Introduction to the Rubidium Frequency Standard using the FE5650A

Michael A. Parker

Angstrom Logic, LLC

Laboratory Series

Introduction to the Rubidium Frequency Standard using the FE5650A

Laboratory Series

Michael A. Parker

Angstrom Logic, LLC

Angstrom Logic, LLC (ALL)
Somerset, NJ

Published in the United States of America

International Standard Book Number: (Soft)
(Digital) 978-1-7350986-0-9

Library of Congress Cataloging-in-Publication Data
Parker, Michael A.
Introduction to the Rubidium Frequency Standard using the FE5650A
Laboratory Series

Contents

Chapter 5: FE-5650A Initial Setup and Tests

Chapter 6: Serial Port and Initial Frequency Tests

Chapter 7: Modifications

APPENDICES

Preface

Atomic clocks and frequency standards/sources have been available for decades especially the Cesium type for Global Positioning Satellites (GPS) and the Rubidium type as the secondary standard. Recently, the Rubidium Frequency Standard (RFS) has become very affordable at $100 for surplus units such as the FE-5650A manufactured by Frequency Electronics Inc. (FEI). This book uses the pre-2005 FE-5650A Option 58 unit as an example (refer to Figure 1.2 to identify the unit). Home workshops, tech shacks and professional laboratories can gain access to highly affordable, accurate atomic frequency standards (0.001 Hz accuracy at 10MHz output frequency) – an accuracy that exceeds by orders of magnitude some of the best Oven Controlled Crystal Oscillators (OCXO). The numerous applications include calibration of electronic equipment, highly accurate frequency sources to replace the simple crystals in microcontroller circuits along with frequency generation and measurement equipment, and highly accurate timing sources. Some hobbyists have used the Rubidium Frequency Standard (RFS) to build highly accurate 24 hour clocks while experimenters have used the cesium variety to test predictions of time dilation of general relativity. Still others have suggested uses for range finders and frequency multiplier circuits. Although these RFS units are not quite as accurate/stable as the GPS Disciplined Oscillators (GPSDO), their utility might surpass the GPSDO in that they do not require an antenna nor do they require clear access to the sky – a big advantage to laboratory sub-basement dwellers.

This book delivers the physical principles of the Rubidium Frequency Standard (RFS) along with the modifications, tests and software. The first several chapters introduce the origin and role of the hyperfine transitions, optical pumping, frequency detection, servo electronics, Allan Deviation, sources of error, and the basic functional blocks as exemplified by the FE5650A. The remaining chapters describe auxiliary construction to implement the FE5650A unit, modifications for wider bandwidth (100Hz-15MHz), methods for special-purpose calibration, and construction of an external controller to set the RFS frequency. As mentioned, the book resources include demo-quality software for communications, an external controller and for calculating and plotting the Allan Deviation. The external controller software is written in C/C++ and the PC based software is written with Microsoft Visual Basic/C#.

This book addresses RFS science and technology in two basic parts. The first four chapters will be of most interest to those readers with some previous knowledge of the RFS and its physical mechanisms although Chapter 3 regarding the Direct Digital Synthesizer circuits should be of interest to most readers. The first several chapters contain discussion of the physical principles including the atomic physics for the hyperfine transitions and methods of optical pumping. The remaining chapters develop circuit design, modifications and programming that can best be appreciated by those having previous experience with electronic circuit construction. The chapters guide the reader through programming a microcontroller. The home enthusiast will find the process of bringing the FE-5650A to life relatively easy. However, keep in mind, the FE-5650A units can have varying design depending on the manufacture date and options; the reader should refer to Figure 1.2 of Chapter 1 to identify the unit. Much of the information on the FE-5650A operation can be found on the internet per various included recent links. Enough credit cannot be given to the intrepid explorers (in the references) who determined the functions of various circuits in the unit and then posted their findings and their various modifications.

Some readers will wonder why so much source code has been included especially for Visual Studio software when only an internet address for download need be included. As an answer, anyone reading the book then has direct permanent access to the code whereas internet pages come and go. The temporary nature of internet links also poses a problem for references in the book. An online book most naturally uses online references; those references are transitory by nature of internet websites unlike libraries and journal collections. For the present book, some references offer two links to the same article or electronic component. If the reference can no longer be found, search the title or author.

The author did not initially intended to write and distribute a how-to-do book for the FE5650A but something 'funny' happened along the way. The author purchased two RFS units in order to calibrate some lab equipment. During the testing of the serial communications, the RFS units appeared to transmit encrypted data in response to the status inquiry. After notifying the EBay vendor, he unexpectedly and graciously sent three more units without charge. So a set of test and setup notes could be made available in exchange in order to help the next person needing an RFS. Well, writing the notes went fast but then a decision was made to add information on the physics/engineering of the Rubidium Physics Package and then an introduction to Allan Deviation, an external controller, additional circuits, calibration techniques, and demo-grade software. Overall, the value-added topics increased the notes to a book and delayed the release by a few months.

By way of background, the author has researched at university and government laboratories in areas including novel quantum well laser devices, optical process in bulk and quantum structure materials, instrumentation, and applications of quantum optics. He has taught graduate and undergraduate university physics, engineering, and mathematics courses and has written several textbooks in the topical areas of condensed matter physics, quantum theory, laser physics (matter-light interaction) and mathematics.

The author thanks Serey Thai, Ph.D., for early discussions related to integrated rubidium frequency standards, Ron Schmidt (WA5QBA) for helpful information on the WWV and crystal oscillators and Carol Parker for helpful discussions and assistance.

Michael A. Parker, Ph.D.
Angstrom Logic, LLC

Chapter 1: Introduction

Atomic Frequency Sources (AFS) [1.1-6] offer unprecedented accuracy and stability, and surplus units can be found at a fraction of the cost of the original new ones. These Atomic Frequency Sources offer many orders of magnitude better stability and accuracy than the various crystal oscillators. The output frequency of an 'atomic clock' is set by electronic transitions between atomic energy levels. The Rubidium Frequency Standard RFS is the least expensive of the atomic clock genre with widespread commercial, military, laboratory and test bench applications. Some readers will undoubtedly want to use the RFS for equipment calibration and research experiments while others will use it for a home lab [1.7-10] and still others for something as simple as a highly accurate clock using the 1 pulse per second (1pps) output [1.11].

This first chapter briefly introduces the AFS with special focus on the Rubidium Frequency Standard (RFS) FE-5650A produced by Frequency Electronics Inc. (FEI). The RFS is an electronic oscillator (i.e., 'atomic clock') that offers vastly improved accuracy over mechanical and crystal-based oscillators [1.2, 12] using vaporous rubidium atoms. Wikipedia [1.13-14] defines the rubidium clock as follows.

> "A **rubidium standard** or **rubidium atomic clock** is a frequency standard in which a specified hyperfine transition of electrons in rubidium-87 atoms is used to control the output frequency. It is the most inexpensive, compact, and widely produced atomic clock, used to control the frequency of television stations, cell phone base stations, in test equipment, and global navigation satellite systems like GPS. Commercial rubidium clocks are less accurate than cesium atomic clocks … "
>
> All commercial rubidium frequency standards operate by disciplining a crystal oscillator to the rubidium hyperfine transition of 6834682610.904 Hz. The intensity of light from a rubidium discharge lamp that reaches a photodetector through a resonance cell will drop by about 0.1% when the rubidium vapor in the resonance cell is exposed to microwave power near the transition frequency. The crystal oscillator is stabilized to the rubidium transition by detecting the light dip while sweeping an RF synthesizer (referenced to the crystal) through the transition frequency.

Subsequent chapters discuss the physical principles (i.e., physics) of the RFS, the electronics for deriving a signal and disciplining a crystal oscillator, the operation of the Analog Devices AD9830A Direct Digital Synthesizer (DDS), the methods for determining oscillator instability using the Allan Deviation, and then tests and modifications of the FE5650A. First-time readers can probably skip the chapters on the physics module and the Allan Deviation. The majority of the information regarding the circuits and modifications was found at a variety of websites and we summarize that information in the chapters as noted by the relevant references.

Figure 1.1: The FE-5650A Option 58 as viewed from the connector side. The unit is designed and manufactured by Frequency Electronics Inc. FEI.

Section 1.1: FE-5650A Opt. 58

The discourse primarily focuses on the FE-5650A Option 58 RFS manufactured by Frequency Electronics Inc. (FEI). The units were purchased through EBay from China [1.15] and were in operation from approximately 2000 through 2005. Riley [1.2] states the FE-5650A units were used in Lucent cell phone towers/base stations. Actually, the Riley article discusses the comprehensive history of Rubidium Standards. The FEI rubidium standard FE-5650A provides a portable, highly stable frequency source – for the history of FEI, refer to the company's article on their webpage [1.16]. The Option 58 unit consists of a compact aluminum structure (Figure 1.1) containing four double sided printed circuit boards (PCBs, Figure 1.2) and the so-called 'physics module' (metal structure at upper right in Figure 1.2). The 'physics module' shown without the metal cover in Figure 1.3 contains a vessel of Rubidium vapor, which provides the atomic frequency source, along with an optical pump and sensor, microwave interrogator, and a Helmholtz coil that singles out a specific frequency. Subsequent chapters discuss modifications to both (i) the Direct Digital Synthesizer DDS board, which in the ideal case can produce a sinusoidal signal with frequency in the absolute maximum range of DC through 25MHz, and (ii) the back side of the regulator board attached to the front aluminum plate (Figure 1.1), which has a comparator and counter to convert the DDS sinewave to a divided-down square wave and pulses.

Figure 1.2: The DDS board.

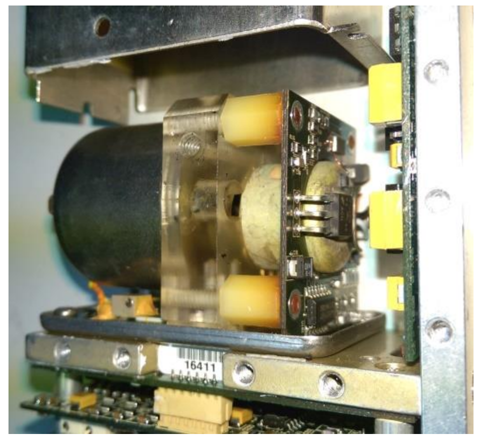


Figure 1.3: Physics module exposed

Section 1.2: Comments on Accuracy and Stability

The accuracy and stability of the Rubidium Frequency Standard RFS (along with its compact size, low weight, low cost) make it highly relevant for the both the professional and amateur laboratory. Table 1.1 shows the typical accuracy for a variety of oscillators/clocks [1.14, 17, 18] but does not specify the averaging time (refer to Ch. 4) The Rubidium Frequency Standard has approximately 10,000-fold better accuracy than the Oven Controlled Crystal Oscillator (OCXO) but 10-fold less than the Cesium clock. The accuracy is measured in part-per-million ppm or parts-per-billion and it is typically calculated using the Allan Deviation (Chapter 4).

Type	Frequency (Hz)	Accuracy
Crystal	Cut to user specification	10^{-5} (10ppm)
TCXO – Temp Cont. Xal	Cut to user specification	10^{-6} (1ppm)
OCXO – Oven Cont. Xal	Cut 5-10MHz	$2\ x\ 10^{-8}$ (0.02 ppm)
^{87}Rb Rubidium (RFS)	6 834 682 610 .904 324	10^{-12} (0.001 ppb)
^{133}Cs Cesium	9 192 631 770 .0	10^{-13} (0.0001 ppb)
^{1}H Hydrogen Maser	1 420 405 751 .7667	10^{-15} (10^{-6} ppb)
^{87}Sr Optical Clock	429 228 004 229 873 .4	10^{-17} (10^{-8} ppb)

Table 1.1: The table displays the accuracy of the various types of oscillators but without the requisite type period. Aging and drift are not shown. The frequency for the atomic clocks refers to the atomic frequency not the output frequency of a commercial unit typically in the neighborhood of 10MHz.

The possible/expected error after a period of time T could be calculated by $Error = T\ A$ where A is the accuracy listed in Table 1.1. As a couple of examples, after 24hours, the OCXO can be in error by 1700 uS and the RFS can be in error by 86.4 ns. The expected error for the cesium clock would be 8.6 nsec which is 10-fold less than the RFS. Oscillator instabilities can be divided into several categories [1.19-20] including (i) short term such as on the order of seconds resembles frequency noise, (ii) long term such as on the order days or months attributable to aging and (iii) environmental such as due to temperature changes. The manufacturer will generally provide spec sheets showing the various effects. Interestingly, RFS and GPSDO discipline a crystal (TCXO, OCXO) based on the rubidium frequency (6.834+ GHz) and the GPS signal, respectively, since such crystal oscillators have better short term stability than that of the RFS and GPS. The crystals can provide a type of 'holdover' function where the unit makes primary use of the crystal oscillation for short times where the RFS (or GPS) do not have good stability or in case the RFS temporarily stops operating.

Section 1.3: Some Applications

Some readers will undoubtedly want to use the RFS for equipment calibration and research experiments while others will use it in a home lab [1.7-10] and still others for something as simple as a highly accurate clock using the 1 pulse per second (1pps) output [1.11].

As mentioned, a prime reason to invest in a RFS is to calibrate electronic equipment. Much of the general and laboratory test equipment specify the accuracy to approximately 0.1Hz to 1Hz or perhaps 0.1ppm to 1ppm (or larger) especially when they incorporate a crystal, TCXO or OCXO. Very often, the better equipment such as digital oscilloscopes, RF spectrum analyzers, frequency counters and function generators will have provisions to set the frequency calibration. Some will allow the user to key-in the correction while others might have an option for an external time-base and some others might require the user to adjust capacitors. Needless to say, the calibration procedure will require a reference with greater accuracy (and probably lower drift). For this purpose, the RFS or a GPS Disciplined Oscillator (GPSDO) provides the required accuracy at a cost under $100 on EBay at the time of this writing. They can be used to calibrate a crystal, TCXO, and OCXO although the crystal will exhibit some temperature dependence which tends to negate the benefits of calibration.

Similarly, it is often possible to replace a crystal with the more accurate RFS although it might be necessary to tailor the voltage range or the frequency. The out-of-the-box FE-5650A has provisions to adjust the output frequency in the range of roughly 6MHz to 14MHz without modifying the circuitry beyond adding appropriate interfacing for the RS232 port. The frequency range can be expanded to approximately 30Hz – 14MHz can by changing some capacitors as will be discussed. Some function generators optionally provide connectors for an external frequency source. It is sometimes possible to replace the crystal (even if it's OCXO or TCXO) in an inexpensive 8 digit frequency counter to obtain higher accuracy. As a matter of fact, a newer frequency counter FA-2 priced at approximately $110 on EBay at the time of this writing, has a rear input for a reference frequency as well as a USB connection to transfer frequency measurements. It features accuracy to approximately 0.001 – 0.0001 Hz. An interesting example concerns Software Defined Radio SDR which might use a crystal without sufficient temperature control [1.21]. A more accurate time base can decrease signal drift but maybe more interestingly, it could also increase the accuracy of the RF Spectrum Analyzer function found in the software such as SDR Airspy. However, some units use a 28MHz crystal and so it might be necessary to use a frequency synthesizer in conjunction with the RFS output or a microcontroller such as the XMEGA128A4U with a built-in PLL.

There are interesting experiments and applications involving atomic clocks. Archeologists have used the Rubidium Frequency Standard RFS as a magnetometer to reveal hidden underground sites [1.22]. Similarly, they have been employed as gradiometers [1.23]. The magnetometer can be used to find underground water, pipes and various ore. Even the rubidium without the other RFS technology can be used to estimate age of objects or sites based on the ratio of rubidium 87 to 85 [1.24]. Various experiments use the RFS (either commercial or a disassembled unit) to explore quantum phenomena and other various other effects such as concerning the nuclear magneton, optical pumping, spin and magnetic effects.

College students and professors [1.25-28] have joined the fun and now use atomic clocks (such as cesium atomic clocks) to test the predictions of General and Special Relativity. For General Relativity, time runs slower at positions of larger gravitational field which is consistent with gravity viewed as the warping and intermixing of space-time. Stated in another way, time runs faster further from a massive object. The references provide the relation for the fractional change in the flow of time verses the change in height δh as

$$\frac{\delta T}{T} = \frac{g}{c^2}\delta h = 10^{-16}\delta h \tag{1.1a}$$

where $g = 9.8\, m/s^2$, $c = 3 * 10^8\, m/s$. The change in time δT can be seen to be linear in the height and observation time T

$$\delta T = 1.1 * 10^{-16}\delta h\, T \tag{1.1b}$$

So as a rule of thumb, for each increase in height of $\delta h = 1km$ (i.e., 1000meters = 3280ft = 0.62mi) when observed for a full day T=24 hour time period (i.e., 86400 seconds), the clock at the higher altitude runs faster by $\delta T = 9.4\ \text{ns}$ compared to the lower starting altitude. Notice that a cesium clock might be able to detect the height of 1km but the rubidium would require an increase in height of 10km. By the way, the students used the GPSDO as the reference since the time has been corrected for sea level.

Early experiments in 1971 (and later years) flew cesium clocks on a jet. In this case, both the gravitational and motional effects on time needed to be included. For the Special Theory of Relativity, a clock in motion will tick more slowly than the one at rest with respect to an observer as is consistent with a type of space-time mixing. At low speeds v, the fractional change of time for the moving observer compared with the stationary one is approximately

$$\frac{\delta T}{T} = \frac{1}{2}\left(\frac{v}{c}\right)^2 \tag{1.2}$$

For 10m/s (i.e., 22.4mph), the fractional change is

$$\frac{\delta T}{T} = 5.6 * 10^{-16} \tag{1.3a}$$

and so, over T=24 hours (i.e., 86400 seconds), we find

$$\delta T = 5.6 * 10^{-16} T = 0.05 \text{ ns} \tag{1.3b}$$

The calculations for jet experimental results did account for the rotation of the earth and the experiments agreed well with theory.

As is well-known, the RFS finds a number of commercial and military applications. For example, TV stations use them to provide accurate carriers and synchronization [1.29] . GPS satellites incorporate the cesium frequency source as primary standard and the RFS [1.30] for both a type of back up and for anti-jamming. As another example, incoming and outgoing cell phone signals need to be synchronized especially for larger numbers of calls, since otherwise the service would degrade such as by cross talk. Apparently, the base stations incorporate rubidium clocks that are disciplined by GPS clocks in the sense that the rubidium signal is synchronized/matched to that of the GPS [1.31]. If the GPS signal is lost, then the rubidium clock has sufficient stability to provide a substitute clock for extended periods – the function of 'holdover'.

Section 1.4: Next on the agenda

Chapter 2 introduces the physical operating principles (i.e., physics) of an RFS; it can be skipped if desired. The chapter describes the rubidium electronic energy levels, the origin of those levels, the pumping mechanism that maintains the atomic signal, and the method of interrogation.

Chapter 3 continues with some of the electronic circuits for the FE5650A/FE5680 including the Direct Digital Synthesizer DDS, the interface with the synthesizer, the specifics of the amplifier and filter electronics, and the function of the embedded microcontroller. The discussion of the DDS related circuits focuses on those electronic components ripe for modification including those to improve the drive capability of the sinewave output, the externalization of the square wave and serial port, and to extend the bandwidth of the unit. The setup, modifications, tests, calibration are fairly simple and appear in Chapters 5-9. Most if not all of the RFS internal circuits have been gleaned from a variety of internet websites. One of the best is the Time-Nuts on www.leapsecond.com [1.32] with the signup listed in [1.33]. The website provides daily emails on topics of interest. For those planning to use a GPSDO, consider looking into the Lady Heather software for the PC used to control and monitor the GPSDO [1.24].

Section 1.5: Availability of Software and Preprogrammed Microcontrollers

The book includes access to two groups of demo-grade (i.e., alpha version) software: (i) utility programs written using Microsoft Visual Studio (Windows) for Allan Deviation, a PC-based controller, and an FA-2 Frequency Counter converter; (ii) a C/C++ program for a Microchip-Atmel ATMEGA328P microcontroller used in an external controller for the FE-5650A. Chapter 10 and Appendix 6 provide the source code for the C/C++ program and Appendices 7-9 provide the PC software which can be copied transferred to the relevant platform. Compiled and uncompiled versions of the software/firmware may be available on a flash drive or as a download [1.35]. The PC software might be available as (i) a Visual Studio solution/project, (ii) an MSI installation file, and (iii) an executable .exe file which can be run without installation. The microcontroller program can be found as an Atmel Studio solution/project or a HEX file that both require a programmer. The C/C++ program is transferred to a Microchip-Atmel ATMEGA328P microcontroller for the external FE5650A controller. However, a preprogrammed ATMEGA328P might be available [1.35]. The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the words) and the contents are copyrighted and cannot be transferred from the original media/form for any purpose other than for personal use (in the strictest sense) without prior explicit written approval by the author.

Section 1.6: References

[1.1] F. G. Major, "The Quantum Beat: Principles and Applications of Atomic Clocks", 2nd Ed., Springer, NY, 2010.
[1.2] W. J. Riley, "A History of the Rubidium Frequency Standard", IEEE UFFC-S History, July 2019
http://www.wriley.com/A%20History%20of%20the%20Rubidium%20Frequency%20Standard.pdf
or at http://ieee-uffc.org/about-us/history/a-history-of-the-rubidium-frequency-standard.pdf
[1.3] W.J.Riley, "Rubidium Atomic Frequency Standards for GPS Block IIR"
http://www.wriley.com/ION%2792.pdf
[1.4] D. A. Allan, N. Ashby, C. C. Hodge, "The Science of Timekeeping", HP Application Note 1289
http://www.allanstime.com/Publications/DWA/Science_Timekeeping/TheScienceOfTimekeeping.pdf
[1.5] 1.8.0C intro material
http://indico.ictp.it/event/a08148/session/39/contribution/25/material/0/0.pdf
[1.6] G M Saxena, Rubidium Atomic Clock: The Workhorse of Satellite Navigation, World Scientific Publishing, 2019. https://www.worldscientific.com/worldscibooks/10.1142/11249
[1.7] Video: every maker should have a Rubidium Frequency Standard
https://www.youtube.com/watch?v=zW5ffFuEQsw
[1.8] Video: Turning a Rubidium Standard into a proper tool:
https://hackaday.com/2013/08/05/turning-a-rubidium-standard-into-a-proper-tool/
[1.9] https://www.wired.com/2007/12/time-hackers/
[1.10] https://hackaday.com/2015/05/27/measuring-accuracy-of-rubidium-standard/
Also see https://www.tinaja.com/glib/WWVBexps.pdf
[1.11] Rubidium disciplined real time clock:
https://hackaday.com/2016/09/25/rubidium-disciplined-real-time-clock/
[1.12] V. Corey, Frequency Electronics Inc. "Precision Oscillator Overview", Presentation, 1998
https://www.researchgate.net/profile/Vince_Corey/publication/325999270_OCO_VCO_Rb_FS_97/links/5b32832ca6fdcc8506d1153a/OCO-VCO-Rb-FS-97.pdf?origin=publication_detail
[1.13] https://en.wikipedia.org/wiki/Rubidium_standard
[1.14] https://en.wikipedia.org/wiki/Atomic_clock
[1.15] FE5650A on Ebay from I.Fluke:
FEI FE-5650A Option 58 programmable Rubidium Oscillator (change Freq on pc)
[1.16] History of FEI:
https://www.referenceforbusiness.com/history2/54/Frequency-Electronics-Inc.html
[1.17] Example accuracy for various oscillators:
https://www.best-microcontroller-projects.com/ppm.html
[1.18] https://www.meinbergglobal.com/english/specs/gpsopt.htm
[1.19] general discussion of stability
https://www.febo.com/pages/stability/
[1.20] brief long and short term noise:
https://endruntechnologies.com/products/time-frequency/oscillators
[1.21] replacing the crystal in sdr with Rb std
https://www.radiohobbyist.org/blog/?p=468
[1.22] S. Breiner, "The Rubidium [Cesium] Magnetometer in Acheological Exploration" Science Vol. 150, No. 3693, pp. 185-193 (1965)
http://www.breiner.com/sheldon/papers/Rubidium%20Magnetometer%20in%20Archeological%20Exploration.pdf

[1.23] C. Osgood, “Design and use of a gradiometer connected rubidium magnetometer,” Rev. Phys. Appl. (Paris) 5, 113-118 (1970).
https://hal.archives-ouvertes.fr/jpa-00243342/document (use updated web browser) Alternate site:
https://rphysap.journaldephysique.org/articles/rphysap/abs/1970/01/rphysap_1970__5_1_113_0/rphysap_1970__5_1_113_0.html
[1.24] L. Georgescu, “Rubidium round-the-clock” Nature Chemistry, Vol. 7, p.1034 (2015)
https://www.nature.com/articles/nchem.2407.pdf?origin=ppub
[1.25] B. Patterson et. al., “An undergraduate test of gravitational time dilation,”
https://arxiv.org/pdf/1710.07381.pdf
[1.26] T. Van Baak, “GPS, Flying Clocks and Fun with Relativity” 2018 presentation, SCPNT/Stanford
http://web.stanford.edu/group/scpnt/pnt/PNT18/presentation_files/I08-VanBaak-GPS_Flying_Clocks_and_Relativity.pdf
[1.27] More on gravity
http://www.leapsecond.com/pages/atomic-tom/
details at http://www.leapsecond.com/great2005/
[1.28] Hafele-Keating experiment: https://en.wikipedia.org/wiki/Hafele%E2%80%93Keating_experiment
[1.29] Rb at tv station: https://apps.dtic.mil/dtic/tr/fulltext/u2/a485917.pdf
[1.30] RFS in satellites: https://apps.dtic.mil/dtic/tr/fulltext/u2/a427869.pdf
[1.31] https://www.eetimes.com/document.asp?doc_id=1278627
[1.32] Excellent website for all things timing: http://www.leapsecond.com/time-nuts.htm
[1.33] Signup: http://lists.febo.com/mailman/listinfo/time-nuts_lists.febo.com
[1.34] Lady Heather software: http://www.ke5fx.com/heather/readme.htm
W6AER, “Building a 10MHz GPSDO using the Trimble Thunderbolt
https://w6aer.com/10mhz-gps-disciplined-oscillator-gpsdo-trimble-thunderbolt/

[1.35] Source code, listings, downloads, flash drives, preprogrammed microcontroller:
(A) Chapter 10 and Appendix 6 provide the source code for the C/C++ program
(B) Appendices 7-9 provide the PC software
(C) Check EBay.com in connection with the title of this book: software and preprogrammed MEGA328P
(D) check www.AngstromLogic.com for current offerings, or write to Sales@AngstromLogic.com
Keep in mind that Angstrom Logic, LLC is not setup/able to answer questions on the book or software.
(E) Check Amazon in connection with the title of this book.
(F) Note: The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the word) and the contents are copyrighted with all rights reserved (also see front copyright page) except as noted. The software/firmware is scanned for viruses before posting. Download only from secure sources and rescan for viruses before using.

Chapter 2: Physics Package

The Rubidium Frequency Standard (RFS) relies on the physics package to generate the highly stable and accurate frequency. The physics package includes a rubidium 87 cell and excitation lamp, optical filter, photodetector and c-field coil. Associated components include a Voltage Controlled Crystal Oscillator (VCXO) along with feedback and interrogation electronics. The frequency-generation function takes place within a Rubidium 87 vapor cell via electronic transitions between the hyperfine levels. The rest of the physics package has means to excite the rubidium 87 (pumping), detect the electronic transitions (interrogating), and servo a Voltage Controlled Crystal Oscillator (VCXO) to provide a stable frequency for a Direct Digital Synthesizer (DDS). The DDS makes it possible to accurately set an output frequency for the RFS.

Section 2.1: Overview and Physical Principles

The present section describes the basic components of the physics package, their interrelation, the method of determining the resonant condition using phase sensitive detection (lock-in amplifier) and the basics of controlling a quartz crystal.

Topic 2.1.1: Overview

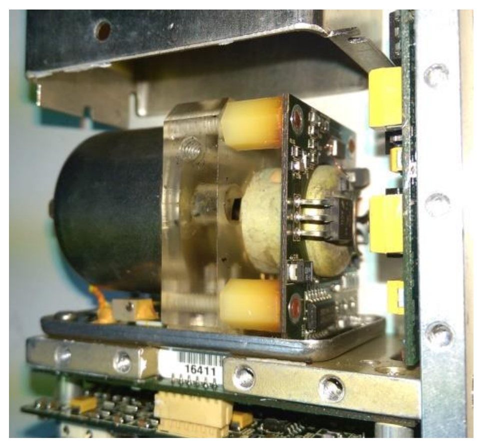


Figure 2.1: The FE5650A physics package without the outer metal enclosure.

The FE-5650A implements myriad technologies to generate and monitor the electron 'hyperfine' transitions in rubidium 87 (denoted by ^{87}Rb) to produce the highly accurate microwave frequency of 6.834,682,61+ GHz [2.1-5]. The '+' after the number refers to additional digits not shown. The word 'hyperfine' refers to an exceedingly small energy difference between the two involved ^{87}Rb energy levels. In particular, for the produced microwave at 6.834+ GHz (wavelength 43.8um), the energy difference is approximately $3.45 * 10^{-5}$ ev which can be compared with that for green light (532nm) corresponding to 2.33ev (where ev refers to electron volts and is a relevant measure of energy for optical transitions). Loosely speaking, the 6.834+GHz is the 'natural resonant' frequency between the two atomic levels but quite different from the resonance in a clock with a pendulum or flywheel. In the case of the Rb, when an electron makes a (hyperfine) transition from a higher to the lower energy level, a photon with energy 34.5 micro-ev (i.e., frequency 6.834+GHz) is produced. One might think that the 6.834+ GHz microwave signal comprises the primary signal of interest to discipline a crystal. It is essential but not exactly primary since the RFS does not have sensors to directly convert that particular GHz signal to a lower frequency for use by other electronic circuits. Rather the absorption of light by the Rb cell for the pump process provides the servo signal via a photodetector.

The physical construction of the RFS Physics Package appears in Figure 2.1 – the 'time-keeping' takes place there. The corresponding block diagram of the package can be seen in Figure 2.2. Light from the lamp passes through the optical filter and enters the Rb Cell which contains rubidium 87 in gaseous/vapor form (along with an inert gas). Apparently the literature refers to the Rb as a vapor meaning the vessel also has Rb solid/liquid as a reserve. Heaters raise the temperature beyond 39C (the melting point) so some of the Rb exists in the gaseous state. The filtered light serves two purposes. First, it acts as a 'pump' to excite electrons in the ^{87}Rb from a lower level to the slightly higher energy level; these electrons form a population inversion in the sense that the higher energy level contains more electrons than the lower one. The actual process involves at least a third energy level as will be discussed in a subsequent topic. And second, the light provides a type of indicator or monitor for the microwave signal since, once the ^{87}Rb hyperfine electron transitions produce the microwave signal, the light level reaching the photodetector will decrease when the electrons (that transition to the lower level) re-absorb the light to repopulate the upper level. Again, this requires a third energy level. A signal derived from the resulting decrease in photodetector-current (on the order of 0.1% to 1%) servos the Voltage Controlled Crystal Oscillator (VCXO) – the signal derived from the photodetector output will be smallest when the received light level is the lowest as discussed in Section 2.2 below.

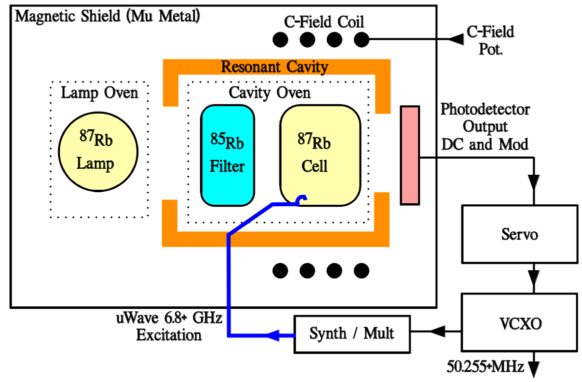


Figure 2.2: Block diagram for an RFS similar to the FE5650A / FE5680.

The question arises as to the method of inducing the hyperfine transition and its relation to a signal from the VCXO. The hyperfine transition is stimulated by introducing an RF signal into ^{87}Rb cell with a frequency that exactly matches the hyperfine transition frequency of 6.834+GHz. The more the RF signal mismatches the ^{87}Rb hyperfine resonant frequency, the lower will be the stimulated emission from the ^{87}Rb and the larger will be the amplitude of the servo signal from the photodetector (Section 2.2). As just mentioned, the stimulated microwave signal itself does not provide the monitor signal although with modern sensors it might be possible to directly sense the strength of the 6.834+GHz signal. A feedback signal is derived from the photocurrent output using a lock-in amplifier technique as discussed later. The photodetector output servos the voltage that controls a VCXO which has a nominal frequency of approximately 50.255MHz. The VCXO output frequency is multiplied by a factor of 136 and applied to the ^{87}Rb cell using a step-recovery diode. So now the voltage to the VCXO can be scanned and the photodetector will produce the smallest signal amplitude when the multiplied VCXO frequency matches the ^{87}Rb resonant frequency. Once the servo stabilizes, the RFS indicates the 'lock' condition. The VCXO signal, as the signal-of-interest, (i) feeds the AD9830A Direct Digital Synthesizer DDS IC (from Analog Devices) as a reference so that the DDS can then synthesize sinewaves with highly accurate frequencies theoretically from approximately DC to 25MHz, and (ii) when divided by 6, provides a clock with frequency 8.376MHz for a Microchip microcontroller. Note that the 8.3876MHz clock is not the same as the typical default frequency from the DDS chip of 8.388608MHz.

Continuing the overview of the RFS, consider again Figure 2.2. Ovens heat the lamp, filter and the ^{87}Rb in the glass/quartz cell to ensure the components contain sufficient Rb gas molecules (temperature generally over 39C) – the FE5650A initially draws a total current of 1.8 Amps but that drops to approximately 0.6 Amps after the heater stabilizes. As mentioned, the lamp serves the dual purpose of pumping the ^{87}Rb gas in the cell to establish a population inversion and also to provide a monitor signal to servo the Voltage Controlled Crystal Oscillator (VXCO) through the photodetector. A synthesizer/multiplier increases by a factor of 136 the VCXO frequency of 50.0255+ MHz which then matches and thereby stimulates the hyperfine transition frequency used to stabilize the VCXO. The resonant cavity, albeit low-Q, strengthens the stimulation of the hyperfine transitions in the ^{87}Rb gas.

The hyperfine transition energy and hence the ^{87}Rb microwave frequency has some slight sensitivity to magnetic fields. The physics package employs mu-metal to prevent external fields from influencing the frequency and stability of the hyperfine transitions.

The magnetic field produced by a Helmholtz coil (i.e., c-field coil or cell-field coil) isolates the two electronic states participating in the hyperfine transitions from other states with exceedingly close energy. Although, not shown, the FE5650A includes a c-field adjustment potentiometer to slightly adjust the magnetic field which also slightly adjusts the hyperfine microwave frequency and hence the output frequency of the RFS. The Helmholtz coil carries DC current and produces a magnetic field across the ^{87}Rb gas cell. As mentioned, in addition to the energy levels sustaining the hyperfine electronic transitions that produce the desired ^{87}Rb microwaves, other levels exist with roughly the same energy. Left to their own wiles, these extraneously levels would interfere with the desired microwave process but fortunately, the energy of these levels are fairly sensitive to magnetic fields, and their energy levels can be moved outside the range of the desired hyperfine transition. The coils provide the magnetic field to change the relative energy between the states. Fortunately, the desired states for the hyperfine transitions are only weakly affected by the c-field. And yet, this slight change does produce a slight change in the hyperfine transition microwave frequency and also therefore the VXCO frequency for the lock condition. The c-field potentiometer can therefore be used for slight, but significant changes to the lock frequency and thereby correct for slight digitization errors in the DDS AD9830A synthesized frequency. Depending on the transfer characteristics between the magnetic field and the energy levels, the lock frequency might be less stable for different c-field currents. If contemplating changing the c-field, it might be a good idea to develop a method to return the c-field back to its original setting such as by measuring the magnitude of the field outside the cell, or measuring the current through the coil, or maybe simply marking the potentiometer angular position. We have not observed any inaccuracy beyond about 0.005 while the best was approximately 0.001 Hz. As previously mentioned, given that magnetic fields affect the accuracy of the Rb standard, the cell and coil are wrapped in mu-metal to shield the Rb states from external magnetic fields. It should be mentioned that the c-field does not need to be adjusted (and should not be) for every frequency since it is the purpose of the reference frequency and the Fcode to obtain the exact frequency required to within the range of 0.001 to 0.01Hz.

Topic 2.1.2: Phase Sensitive Detection (Lock-in Amplifier)

As previously mentioned, the FE-5650A and other RFS units maximize the rate of hyperfine transitions by monitoring the light absorbed by the ^{87}Rb cell and not directly monitoring the 6.84+GHz microwave signal. Those electrons having been induced to transition from the higher to lower energy levels can then be promoted back to the upper level (through a third level) by absorbing photons from the ^{85}RB lamp and thereby decreasing the light intensity reaching the photodetector. It is necessary to optimize the driving/interrogation frequency f from the synthesizer-crystal circuits (Figure 2.2) to best match the hyperfine resonant frequency f_o=6.84+GHz so as to maximize the transition rate and maximize the lamp absorption. FEI as many other companies and researchers in the past make use of Phase Sensitive Detection PSD (i.e., a lock-in amplifier).

Figure 2.3 shows various dispersion curves where the horizontal axis refers to the deviation of the driving/interrogation frequency f from the center of the hyperfine resonant frequency of f_o = 6.84+ GHz – notice the units of Hz for the plots. The ^{87}Rb in the cell absorbs the most light when the interrogation frequency matches the hyperfine resonant frequency. The Lorentzian curve in Figure 2.3a shows the intensity of light passing through the ^{87}Rb versus the interrogation frequency difference $f - f_o$ according to the following equation

$$I_{pd} = I_o \left[1 - \frac{u}{1+4\frac{(f-f_o)^2}{W^2}}\right] \tag{2.1}$$

where f is the driving frequency (i.e., interrogation frequency *near* 6.84+GHz) from the synthesizer-crystal circuits, f_o is the hyperfine resonant frequency (6.84+ GHz), I_o is the intensity of light entering the gas and I_{pd} is the intensity striking the photodetector, u is the maximum decrease in intensity (a fraction), and W is a measure of the bandwidth. Reference 2.5 states a range of values for u and W. Figure 2.3a uses u=0.01 and $W = 256\,Hz$ and the horizontal axes show the quantity $f - f_o$. The bandwidth is in part controlled by Doppler broadening and molecular collisions although an inert gas can decrease the effects of the later.

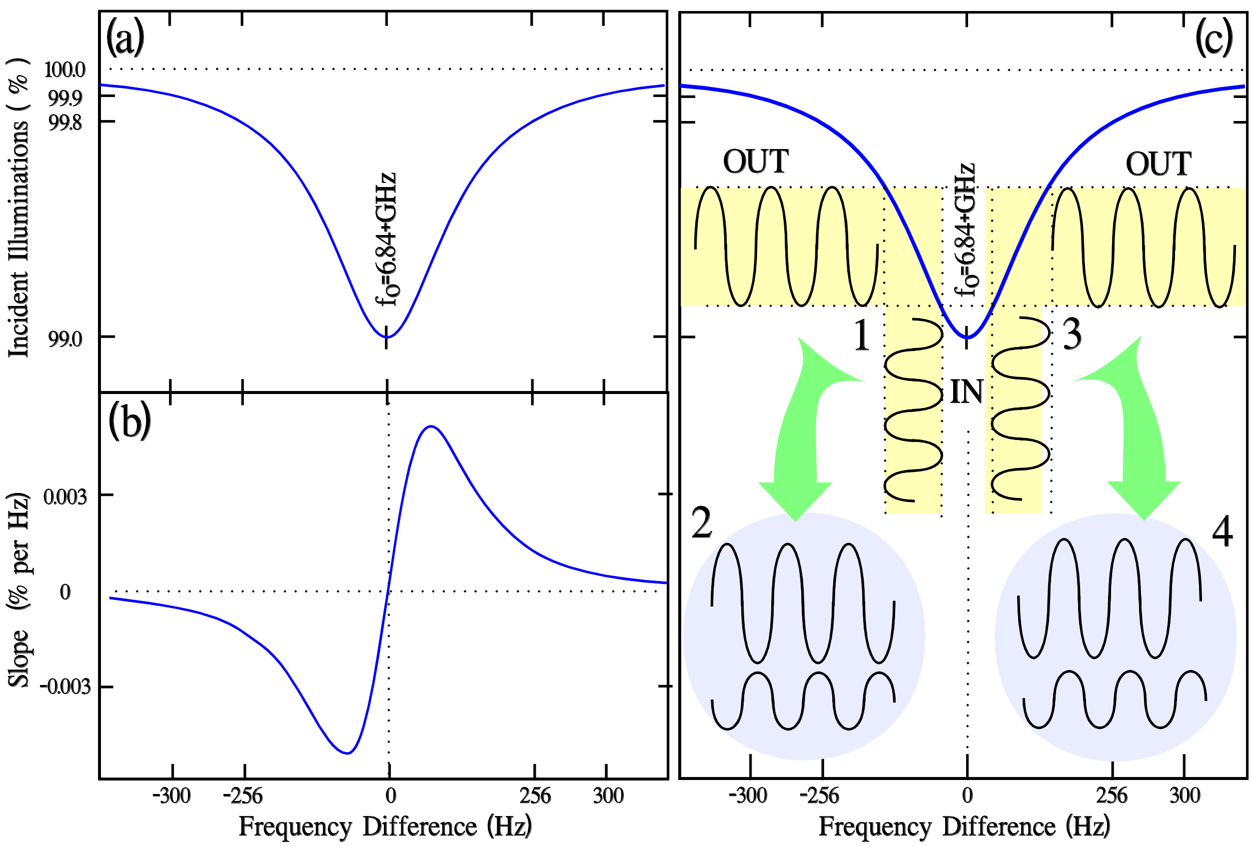


Figure 2.3: All of the curves plot a response to a driving frequency that varies from the resonant frequency f_o=6.84+GHz for the hyperfine electronic transitions in ^{87}Rb. The horizontal axes represent the actual driving frequency as a deviation from the hyperfine resonant frequency. (a) The amount of light from the lamp-filter assembly incident on the photodetector after passing through the Rb cell for the values of u=1% and $W = 256Hz$. (b) The slope of the photodetector Illumination curve in the top left panel. The slope is used to determine the center of the hyperfine resonance. (c) A small FM frequency (marked 'IN') is added to the average interrogation frequency produces an amplitude modulation of the light intensity reaching the photodetector. (1) The FM modulation has a center frequency below the resonant frequency which (2) shifts the output phase by 180° compared to the input. (3) The FM modulation has a center frequency above the resonant frequency which (4) shifts the output phase by 0° compared to the input.

The method of controlling the VCXO comes down to the method of sensing the dip in the transmitted-light (Figure 2.3a). The simplest method consists of "adjusting the driving microwave frequency f until a photodetector produces the smallest signal" won't work for reasons that include (i) the photodetector cannot determine if the driving frequency needs to be increased for the case of $f < f_o$ or if it needs to be decreased for the case of $f > f_o$, and (2) the signal-to-noise ratio is low. However, if the slope of the Incident Intensity curve can be determined as in Figure 2.3b, then the exact center resonant frequency can be seen to produce zero slope. Further as seen in Figure 2.3b, for frequency f near to f_o, the curve approximates a straight line that rises from left to right which means the correction voltage to the VCXO can be expected to have the form of roughly a straight line $V = k_1\,(f - f_o) + k_2$ where $k_1 \neq 0$ and $k_2 \sim 0$ are constants. So regardless of whether $f < f_o$ or $> f_o$, a correction voltage can be determined. Of importance for further discussion is the fact that the slope is negative for $f < f_o$ and positive for $f > f_o$ over the frequency range of interest.

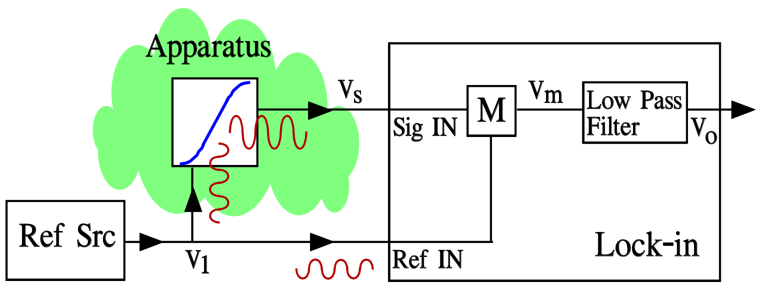


Figure 2.4: Basic setup for a lock-in amplifier. The Apparatus Under Test (AUT) uses the same signal as applied to the reference input of the lock-in. The modified signal from the AUT is essentially multiplied by the input signal at M which produces a DC and frequency-doubled signal. The filter passes the DC signal to the output as $V_o = g\ Cos(\varphi)$.

The slope (Figure 2.3b) can be found by applying a lock-in amplifier (Figure 2.4) which typically has an input for the signal of interest (Sig-IN) and another for a reference signal (Ref-IN) and an output for an RMS voltage and a phase [2.6-8]. The lock-in determines the amount of the input signal Sig-IN that has the same frequency as the signal supplied to its reference input (Ref-IN). The 'amount' is expressed as an RMS voltage and phase. The 'phase' refers to the phase difference between the input signal and the reference signal. Essentially, the lock-in amplifier finds the Fourier coefficient (Fourier Amplitude) of the incoming signal at the frequency of the reference and the phase measured with respect to the reference input. As will be seen, the output of the lock-in can be interpreted as being proportional to the slope (i.e., transfer curve slope or transfer incremental 'gain' g).

Suppose the 'apparatus' in the Figure 2.4 manipulates the amplitude and phase of an incoming signal. Suppose $V_1(t)$ is a signal produced by a signal generator

$$V_1(t) = A\ Sin(\omega t) \tag{2.2}$$

which provides both a reference source for the lock-in and an input signal for the apparatus. As shown in the figure, the apparatus alters the amplitude and phase of the signal (but not the frequency) to produce

$$V_s(t) = gA\ Sin(\omega t + \varphi) \tag{2.3}$$

where 'g' refers to the 'gain' (i.e., transfer curve slope) that changes the amplitude. That is, 'g' represents the slope of the transfer curve for the apparatus. Recall that the angular frequency ω (in radians/sec) is related to the frequency f by $f = \omega/(2\pi)$ (with units of Hz or cycles per second). Among other operations, the lock-in multiplies Sig-IN with Ref-IN at the multiplier M

$$V_s(t)V_1(t) = gA^2\ Sin(\omega t + \varphi)\ Sin(\omega t)$$

and then filters out all but the DC signal (plus some noise) to find the desired result of

$$V_o = \frac{gA^2}{2} Cos(\varphi) \rightarrow g\ Cos(\varphi) \tag{2.4}$$

where the last term obtains by normalizing the input amplitude to the value $A = \sqrt{2}$. Without going through the calculations, Equation 2.4 can be understood by recalling the multiplication of two sinusoidal signals produces the sum and difference frequencies such as f_2+f_1 and f_2-f_1. For the situation in Figure 2.4 where f_1=f_2=f, the sum and difference will be 2f and 0 (DC). The low-pass filter removes the 2f component and allows the DC component of Equation 2.4 to pass [2.6-8]. So essentially, the lock-in can be used to find the slope as in Figure 2.3b; the zero slope will correspond to the minimum of the transmission spectrum shown in Figure 2.3a.

Return to Figure 2.3c and realize the RFS can scan the voltage to the VCXO which means it can scan the driving frequency across the hyperfine resonance fo=6.84+GHz. The RFS adds a small modulation to the voltage applied to the VXCO so as to add a small frequency modulation (FM) to the interrogation signal. Consider the example in Figure 2.3c that shows two cases of where the average voltage to the VCXO sets the driving frequency f slightly below and above the resonance. The input signals are designated 'IN' and the amplitude of the FM deviation can be measured on the horizontal axis. The resulting AM (amplitude modulated) signals are labeled 'OUT' and the amplitude can be measured on the vertical axis on the far left side of the figure. For the case of f < f_o, the small FM signal has a peak-to-peak value of roughly 100Hz as shown near the negative slope of the transfer curve (i.e., gain g) at Region 1 of the figure. The output due to the Rb absorption process produces a peak-to-peak illumination change of roughly 0.4%. The transfer 'gain' is then roughly (0.4%)/(100Hz)= 0.004 (%/Hz) which can be compared with the value of roughly 0.005 in Figure 2.3c. The case for f>f_o is similar. In each case, the lock-in would read a magnitude of roughly 0.005. What about the phase? The inset drawings 2 and 4 show the phases. For f<f_o, the phase is roughly 180^o and the two signals are out-of-phase because of the negative slope on the transfer curve. For f>f_o, the two signals are in phase. Consequently, as the driving frequency scans from left to right, the lock-in will provide an output voltage ranging from -0.005 to +0.005 where the minus sign comes from the phase of 180^o. Zero corresponds to the center of the resonant frequency [2.6].

Section 2.2: Energy States

The Physics Package contains the main components of the rubidium frequency standard (RFS) including the ^{87}Rb cell and lamp, ^{85}Rb filter, and photodetector. The present section discusses the concept of atomic energy levels, photon emission and absorption, as well as the shifting and splitting of energy levels in the presence of magnetic fields, and the basic physics of the rubidium hyperfine states and transitions.

Topic 2.2.1: Basic Concepts of Energy States and Photon Emission

The present topic provides insight into atomic states for electrons, the basic visualization of photon emission/absorption for electron transition between those states and how magnetic fields can affect states.

The energy levels associated with the hyperfine transitions have origins a bit different than might usually be imagined when speaking of energy levels. Most commonly, people imagine a 'planetary' arrangement for which the electrons orbit about the nucleus (i.e., the Bohr model) under the influence of electric and magnetic fields. The cartoon in Figure 2.5 shows an example of two different 'orbits' of an electron in a *hydrogen* atom. The states, meaning the orbits or levels, are typically labeled as 'kets' | > that contain within them the energy of the state such as $|E_1>$ and $|E_2>$. Notice that the states exist even though an electron is not actually present in the state (i.e., level). The states differ in energy as can be easily ascertained since work must be done to separate an electron (negative charge) from the proton (positive charge); hence energy must be added to the electron to move it to an outer orbit. Another example of energy levels, albeit on a very macroscopic scale, would be the rungs of a ladder where the rungs are the states and the potential energy above the ground would be the energy of each state for an object of mass m. The mass might be in any one of the states but the other states still exist independent of whether or not a mass occupies them. Returning to the hydrogen, the orbits have energy (units of electron volts ev)

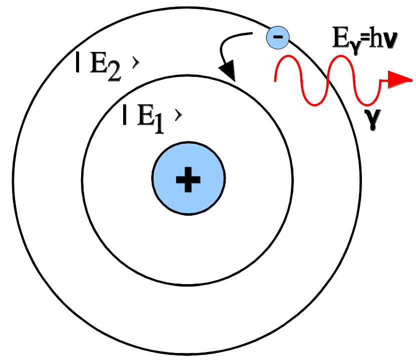


Figure 2.5: Cartoon showing two of the orbital energy states in a hydrogen atom. An electron transitions from a higher energy to a lower one and emits a photon.

$$E_n = -E_I/n^2 \quad \text{n=1,2,...} \qquad \text{(Hydrogen, } E_I = 13.6ev\text{)} \tag{2.5}$$

where the ionization energy E_I is the amount of energy required to remove (i.e., ionize) an electron from level n (a.k.a., the principle quantum number) – see reference [2.9]. The electron volt is the energy to move an electron through a potential difference of 1volt. Each eV corresponds to $1.60217662 \times 10^{-19}$ joules. Reference [2.10] reports $E_I = 4.177ev$ for ^{87}Rb.

When an electron at a higher energy level E_b transitions to a lower one E_a, the difference in energy will be delivered elsewhere. We will be primarily interested in transitions involving photons. The energy lost by the electron $\Delta E = E_b - E_a$ reappears as the energy of the emitted photon E_γ that is, $E_\gamma = \Delta E$.

Several relations are helpful. The energy E_γ of the photon is related to the frequency ν of the light or radio wave according to

$$E_\gamma = h\nu \tag{2.6}$$

where Plank's constant h has the value of

h = 6.62607015E-34 Joule Sec or h = 4.13566773306E-15 eV Sec

The wavelength λ or frequency ν can be calculated from the usual relation of

$$\nu\lambda = c \tag{2.7}$$

where the speed of light in vacuum has the value

$c = 2.99792458 * 10^8$ m/s or $c = 2.99792458 * 10^{17}$ nm/s

Finally, combining Equations 2.6 and 2.7 results in a convenient relation between photon energy E_γ and wavelength λ

$$\lambda = \frac{1239.841995}{E_\gamma} \tag{2.8}$$

where the wavelength has units of nm and energy has units of eV. Often people remember the numerator as 1240 for 4 digit accuracy. As a note, the present chapter lists the various constants with quite a few significant digits because atomic positioning and timing systems, such as GPS and atomic frequency standards, typically requires the calculation precision as will be seen in subsequent chapters.

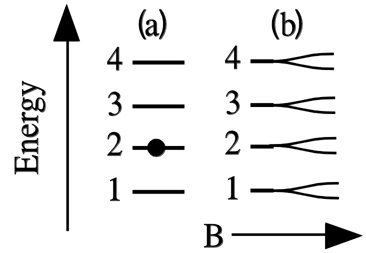


Figure 2.6: Energy levels. The left side (a) shows an electron in energy level #2 similar to the hydrogen atom. The right side (b) shows the energy levels can shift or split such as when the magnetic field B increases. Note: the arrow indicates the value of B increases from left to right. The arrow does not mean vector nor is it a direction for the field itself.

Most often the energy levels are shown without reference to the physical system giving rise to those states. For example, the levels are drawn without showing the atomic orbitals (etc.) as in Figure 2.6a. For rubidium, the energy levels of interest can be shifted/split (Figure 2.6b) by applying an external magnetic field. Hence the physics package must be encased in mu-metal to prevent stray magnetic fields including that of the earth from affecting the frequency of the emitted photon (i.e., the microwave at 6.834+ GHz). We will return to Figure 2.6 in ensuing topics.

Example 2.1: Suppose a hydrogen electron transitions from n=3 to n=2 and emits a photon. Determine the energy, frequency and wavelength of the emitted light.

Solution: The energy difference

$$\Delta E = E_3 - E_2 = (-1.51) - (-3.4) = 1.89 \text{ eV}.$$

will be given to the photon and so $E_\gamma = 1.89\, eV$. Using this E_γ as the photon energy, we find the frequency from Equation 2.7 to be

ν = 456963 GHz or 456.963 THz

Again using E_γ as the photon energy, we find the wavelength from Equation 2.8 to be

$$\lambda = 656\, nm$$

This wavelength corresponds to visible red as can be seen in Figure 2.7.

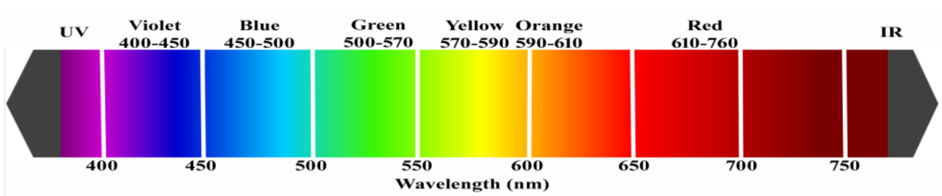


Figure 2.7: Visible spectrum. Image after [2.11]

Example 2.2: Suppose a photon could be emitted from Rubidium 87 for a transition from n=6 to n=5, what would be the frequency?

Solution: The energy difference is on the order of $\Delta E = 0.0511$ev and so the wavelength is

$\lambda = 24266$nm and then $\nu = 12.4$ GHz.

Topic 2.2.2: Spin, Magnetic Fields and Energy Levels

For systems involving electron motion (such as spin and orbital motion), the electron produces a magnetic field. As described next, the energy levels depend not only on the orbit of the electron but also on any external magnetic field B.

Energy levels associated with magnetic fields can be seen to arise by reference to the left side of Figure 2.8. Assume for simplicity the bar magnet pivots around its center. As is well known, north poles repel each other as do south poles. Consequently, an external agent (i.e., person) needs to do work (i.e., add energy) on the bar magnet to swivel it around so that the two north poles are adjacent and the two south poles are adjacent (as shown). In other words, energy must be added to bring the two magnetic fields B and B1 *antiparallel* to each other. Therefore, for the situation shown on the left side, the bar magnet is in a HIGHER energy state than when it is swiveled around by roughly 180 degrees. Further, it should be obvious that the larger the field B then the greater will be the energy required to make the magnetic fields antiparallel – the magnetic field B shifts the energy levels. Without the magnetic field B, the energy of the bar magnet would be independent of its angular position. For the situation in Figure 2.8a, the energy of the bar magnet can be written as

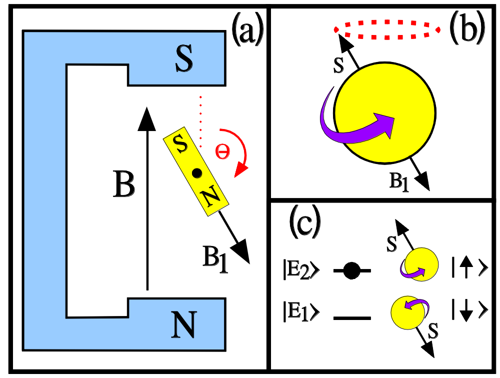


Figure 2.8: (a) Energy must be added to rotate the bar magnet so that B1 is antiparallel to B. (b) Cartoon showing electron spin produces a magnetic field B1 similar to that for the bar magnet but the spin axis processes. (c) Four methods of showing the energy states.

$$E = a - bB\ Cos(\theta) \tag{2.9}$$

where B is the magnetic field external to the bar magnet, ‘a’ is an offset energy, ‘b’ is a related to B1, and θ is the angle between B and B1. For the bar magnet, the offset can be set to zero when other sources of energy are ignored so the energy of the bar varies between –bB and +bB as the angle θ varies between 0 and 180°.

Figure 2.8b shows a classical view of a spinning electron (i.e., the electron continuously spins to produce *spin* angular momentum $\vec{S}$). This view is meant to help with visualizing the situation and does not conform to quantum mechanics beyond showing a relation between spin and the electron’s magnetic field related to that spin [2.9]. The direction of the spin $\vec{S}$ defines the axis of electron rotation (and its length, in this cartoon view, might be intuited as the rotation speed in combination with the mass distribution). Because the electron has charge and because the spin puts the charge in motion, the electron produces a magnetic field B1. Note that the symbol ‘S’ in the figure was originally meant to represent the Spin Angular momentum and not the south pole but a happy coincidence occurs for the electron in that the spin and south pole can be drawn at the same location and pointing in the same direction. As with the bar magnet, when B1 parallels B, the electron occupies the lower energy state.

If the electron is part of an atom, then the orbital energy (such as in Equation 2.5) needs to be added to the energy due to the magnetic field orientation which is usually stated in terms of spin up and down. For example, the ‘a’ in Equation 2.9 would then be related to the energy in Equation 2.5. In quantum mechanics, the electron *spin* has only two energy states ‘up’ and ‘down’ but the electron spin axis never actually coincides with the magnetic field B. Instead the spin axis points similar to that shown in Figure 2.8b for spin 'up'. The figure also shows the electron spin axis precesses around the direction of the B field (recall how the spin axis of a child's toy 'top' precesses/rotates). The energy does not change along the precession path. Finally, Figure 2.8c shows four different methods of labelling the energy levels for the spin. Of particular importance for ensuing topics are the two horizontal lines. The electron with the spin pointing upward in Figure 2.8b is shown as the solid dot on the higher energy level E2.

To see how a magnetic field might cause energy levels to split, consider the case of a single electron with two possible states of up and down. First suppose the external magnetic field is zero B=0 so that the second term on the right-hand side in Equation 2.9 (repeated as Equation 2.10)

$$E = a - bB\ Cos(\theta) \tag{2.10}$$

becomes zero, specifically $bB\ Cos(\theta) = 0$. For B=0, rotating the electron from spin 'down' to 'up' (Figure 2.8) does not require any effort at all (i.e., no work and hence no added energy) as previously mentioned. Consequently for B=0, the electron energy becomes 'E=a' and the spin up and spin down energy levels coincide as shown for the low B situation in Figure 2.6b (i.e., left side of the forked/bifurcated curves). Next, suppose the external magnetic field B increases to a nonzero value, then the second term on the right-hand side of Equation 2.10 adds or subtracts energy from the value 'a' depending on whether the spin is up or down through the term $bB\ Cos(\theta)$. This means that the spin energy levels must separate in energy (i.e., split) similar to that shown in Figure 2.6b in order to produce the two levels shown in Figure 2.8c.

The spin has only one 'rate of revolution', which means the length of $\vec{S}$ cannot be changed, although the component along the B-field can be either 'up' or 'down'. Quantum mechanics provides the length of $\vec{S}$ as

$$S = \sqrt{s(s+1)}\ \hbar \tag{2.11}$$

where the index s (i.e., the 'quantum number') has only the single value of s = ½, and so the length is $S = \left(\sqrt{3}/2\right)\hbar$ and the z-component (i.e., the component along B) is $S_z = \pm\hbar/2$ where $\hbar = h/(2\pi)$ and h is Plank's constant. The electron is called spin ½ because the only acceptable value of s is s = ½.

Topic 2.2.3: Orbital Angular Momentum

Don't assume that *only* the spin of the electron affects the electron energy levels. The fact that the electron moves in relation to the nucleus (i.e., orbital motion) means the orbital motion also produces a magnetic field similar to electrons moving in a loop of wire. Figure 2.9 (Left) shows a concept diagram of an electron orbiting a nucleus along with the magnetic field B2 produced by that motion. As before, the external magnetic field is denoted by B. And as before, energy must be added to the electron-system to swivel B2 antiparallel to B. The *orbital* angular momentum vector $\vec{L}$ gives the axis of rotation for the orbit [2.12-14, 2.9]. The vector $\vec{L}$, and hence the axis, will precesses around the field B (Larmor precession) similar to that for the electron spin $\vec{S}$. As it turns out, the length of $\vec{L}$ (i.e., its magnitude) can only have certain values (i.e., quantized)

$$L = \sqrt{\ell(\ell+1)}\,\hbar \qquad \ell = 0, 1, \dots, n-1 \qquad (2.12)$$

where the index ℓ is an angular momentum quantum number, and $\hbar = h/(2\pi)$ and h is Plank's constant and n is the shell number. Notice that when the literature indicates an angular momentum of say '2', for example, *it refers to the index ℓ in Equation 2.12 and not the length L*; however, obviously ℓ and L are interrelated. Classically speaking, the length L primarily provides information on the rotation speed and also the mass distribution from the center of rotation. For our purposes, we are interested in the length L because it relates to the number of possible states (i.e., directions of L) for the electron orbitals. And as it turns out, the direction for *each* $\vec{L}$ (i.e., the angle between $\vec{B}$ and $\vec{L}$) can only have certain values as labelled by integers

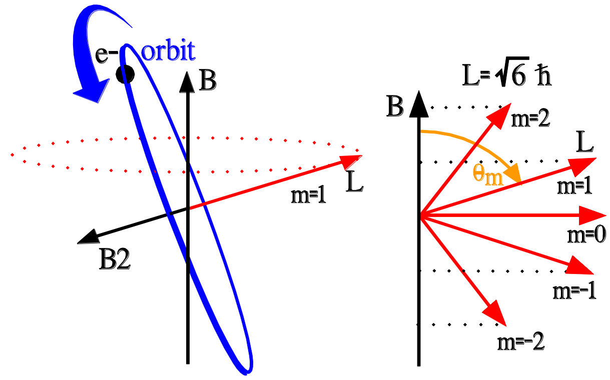


Figure 2.9: Right: Five possible angles (states) for a D orbital (ℓ=2) of the 5th shell (n=5). The symbol 'm_ℓ' has been abbreviated to 'm' for visual clarity. Left: A cartoon representation (Bohr Atom) of an electron in a circular orbit (ignoring electron spin) for m=1; notice the orbit precesses around the magnetic field direction B. The left side is a concept diagram and bears no visual resemblance to the accurate quantum mechanical pictures. Notice $m_\ell = 0$ is perpendicular to B and so the angle is 90°.

$$m_\ell = -\ell, -(\ell-1), \dots, +\ell \qquad (2.13a)$$

which also label the components (a.k.a,. states, direction) of $\vec{L}$ along the B-axis; that is, m_ℓ labels the z-components of $\vec{L}$ – the location where the dotted lines in the figure intersect the B axis corresponds to the z components. Consequently the components can only have certain values given by

$$L_z = m_\ell \hbar \qquad (2.13b)$$

as do the angles θ_m in

$$Cos(\theta_m) = \frac{L_z}{L} = \frac{m_\ell}{\sqrt{\ell(\ell+1)}} \qquad (2.13c)$$

Comparing the left and right sides of Figure 2.9 shows the orbital on the left is the $m_\ell = 1$ state.

Although not shown, it should be especially noted that the quantum mechanical representation of the electron orbitals do not 'look' anything like that in Figure 2.9 – see References [2.15-18] as just a few examples out of many.

Other names exist for the various orbital angular momentum states and those names are usually associated with the periodic table of the elements. The present case in Figure 2.9 corresponds to the orbital for the valence electron in one of the D orbitals. Some readers might recall from the periodic table that the first four shells for rubidium are full and the fifth has only 1 electron. In the 5th shell, the following orbitals are available: S (i.e., ℓ=0) has no orbital angular momentum state, P (i.e., ℓ=1) has three orbital angular momentum states, D (ℓ=2) has five orbital angular momentum states and F (i.e., ℓ=3) has seven orbital angular momentum states. However, each orbital state can have two spin states namely 'up' and 'down'. So in total there are 2 states associated with S (i.e., ℓ=0), 6 states with P (i.e., ℓ=1) and 10 states with D (ℓ=2) and 14 states with F (ℓ=3). Note, the 'F' notation used here for the orbitals should not be confused with the later use of 'F' for the total angular momentum – too many people want to use the same single letter. The possible orbital angular momentum D states appear in the right panel of Figure 2.9. See References [2.9-10,12-14]. Keep in mind that the letter S has many definitions so far: S for south pole, S for spin, and S for an orbital – the text will need to make clear on the usage. As a point worth emphasizing, each additional possible motion (i.e., degree of freedom) for the electron makes possible additional energy levels. Some of the levels might have the same energy (usually termed degenerate levels) until an external influence separates them such as the external magnetic field for the electron spin (Zeeman Effect).

Example 2.3: For the right side of Figure 2.9, find the length L, and the z-component L_z, and the angle for $\ell = 2$ (i.e., D orbital) and $m_\ell = 1$.
Solution: The length of L from Equation 2.13a is $L = \sqrt{6}\,\hbar$. The z component from Equation 2.13b is $L_z = \hbar$. Finally, Equation 2.13c provides the angle $\theta_1 = 66$ degrees.

Example 2.4: Repeat the previous example for the electron with spin up except define θ as the angle between $\vec{S}$ and $\vec{B}$.
Solution: The length of spin vector $\vec{S}$ can be found from Eq. 2.11 (using s=1/2) to be $S = \left(\sqrt{3}/2\right)\hbar$. The z component for spin up is $S_z = \hbar/2$ and so angle can be deduced from $Cos(\theta) = \frac{S_z}{S} = \frac{\hbar/2}{(\sqrt{3}/2)\hbar}$. The angle is $\theta = 55^o$.

Topic 2.2.4: Spin-Orbit Coupling

We are not interested in a complete course in atomic physics but it is important to realize the electrons occupy energy levels, and those electrons can make transitions between the levels by either absorbing or emitting photons, and further, external magnetic fields can split levels (Zeeman Effect) and shift their energy. At this point in the discussion, we have seen the orbital angular momentum and spin describe, in part, the dynamics of the atomic electron. Interestingly, the magnetic field produced by the orbital motion of the electron does not modify the energy levels for the electron but rather a magnetic field produced by the nuclear charge of the atom does modify the energy levels through a relativistic effect. Here, from the point of view of the electron (i.e., an observer riding on top of the electron), the nucleus with its charged protons appear to move about the electron and because the moving nucleus then has charge in motion, a magnetic field is produced at the position of the spinning electron in its orbit. And hence, this magnetic field originating from the nucleus splits/shifts the spin energy levels of the electron [2.19-21, 2.17, 2.14] – the so-called spin-orbit coupling. The effect can be 0.001 eV. The energy for spin-orbit coupling has the form

$$E_{LS} \sim S\,L\,Cos(\theta) = \left|\vec{S}\right|\,L\,Cos(\theta) \qquad (2.14a)$$

where θ is the angle between the vectors $\vec{L}$ and $\vec{S}$, and S is the length of $\vec{S}$. In this case, an S orbital, which means L=0 (i.e., $\ell = 0$ in Equation 2.12), has no energy shift due to spin-orbit effects according to Equation 2.14a. However, the P orbital states (i.e., $\ell = 1$ in Equation 2.10) will experience a split/shift in energy according to Equation 2.14a for the situation when the angle is *not* 90 degrees (i.e., $m_\ell = 0$ produces $\theta = 90^o$ … see Figure 2.9 and the caption). Notice that Equation 2.14a can be restated using $L\,Cos(\theta) = L_z = m_\ell\hbar$ and so

$$E_{LS} \sim m_\ell\, S = \; m_\ell \left|\vec{S}\right| \qquad (2.14b)$$

Equation 2.14b shows $m_\ell = 0$ produces $E_{LS} = 0$.

The shift/split in energy levels for rubidium 87 appears in Figure 2.10 (after Reference [2.22-23]). The left most two states marked 'unperturbed' represents the S and P states, (i.e., $\ell = 0$ and $\ell = 1$ respectively) for the n=5 shell – for these states, the L-S coupling and external magnetic field have not been included. The first question should be "Why are 5P and 5S separated in energy?" since without a magnetic field and LS coupling, changing the orientation of the revolving electron (i.e., L) should not require any energy. The answer resides in quantum mechanics in that electrons having the various angular momenta L experience different electrostatic attraction from the positively charge nucleus because of the shape of the orbital – the 'shape of the orbital' actually shows where the electrons can be found when in the given orbital. [2.9,15,17]. P orbitals are on-averrage slightly further from the positive nucleus than S orbitals. Now consider the LS coupling. As previously mentioned, the 5S state does not split/shift under the LS coupling since, by definition of the S orbital, the angular momentum is zero L=0, and hence Equation 2.14a shows the LS energy is zero (see Figure 2.10). On the other hand, the 5P state is affected by the LS coupling since L is not zero. The degenerate 5P states split into two as shown by the states above the label "LS Coupling" in the figure.

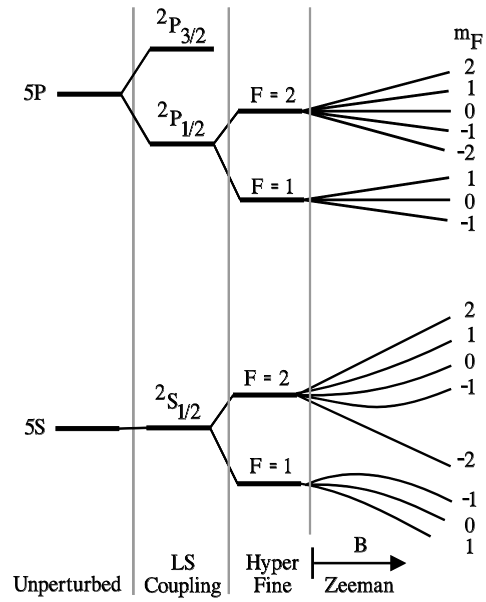


Figure 2.10: The splitting/shifting of the 5S and 5P orbitals for ^{87}Rb, (not to scale). Unperturbed: no LS coupling, no hyperfine; LS Coupling: Nuclear electrostatic field produces magnetic field at position of electron through relativistic effects (S has L=0 and so no LS split); Hyperfine: Nuclear magnetic fields split electron levels; Zeeman: External magnetic field splits sublevels – the 5S quadratic terms are important for RFS. After Steck [1.2.13a]

At this point in the discussion, we yet need to explore the origins of the Hyperfine states and their splitting under the influence of an external magnetic field.

Topic 2.2.5: Total Electron Angular Momentum and Spectroscopic Notation

We divert the discussion to the total *electron* angular momentum $\vec{J} = \vec{L} + \vec{S}$ and its components (labeled as m_j) along a given axis in order to make sensible the notation (such as $^2S_{1/2}$) used in a typical figure similar to Figure 2.10 and because the hyperfine splitting mechanism makes use of $\vec{J}$. The discussion can be skipped if desired although the last paragraph has some bearing on Figure 2.10.

The quantity $\vec{J} = \vec{L} + \vec{S}$ represents the total angular momentum for the electron as the sum of its orbital rotation and its spin (i.e., the total amount of electron rotary motion). The left side of Figure 2.11 shows the uncoupled orbital angular momentum and the electron spin. Each independently precesses about the B axis (i.e., z axis). While quantum mechanics makes it possible to know the length of each angular momentum vector (L and S) and their z-components (L_z and S_z), the x and y components cannot be known. The left side of the figure shows the z-components can be found by projecting L and S onto the z-axis. Similarly, it suggests the x and y components are positioned 'somewhere' around the dotted circles and cannot be known. In quantum mechanics, the observability has to do with commutation relations and only the length and one component of the angular momentum can be

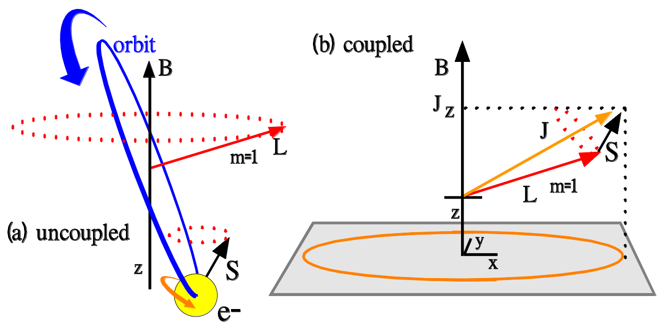


Figure 2.11: (a) Cartoon representation of the *uncoupled* electron spin S and orbital angular momentum L – both independently precess around B. (b) J is the sum of L and S and both precess around J. The Quantum Mechanics prevents knowing the x and y components as can be surmised from the classical view of precession – the location is not known of the projection of J into a horizontal plane – see circle. The same holds for the uncoupled L and S on the left.

known at any given time. Once L and S form J as in Figure 2.11b [2.24], it is the *total* electron angular momentum $\vec{J}$ that precesses around the B axis and both the length of $\vec{J}$ and its z-component can be simultaneously observed but not in conjunction with its x and y components which are positioned 'somewhere' around the circle in the x-y plane. On the other hand, because $\vec{L}$ and $\vec{S}$ must add to $\vec{J}$, any precession of $\vec{L}$ and $\vec{S}$ must be around $\vec{J}$ as shown. Also notice that the precession around J means that the L_z and S_z components (i.e., $\vec{L}$ and $\vec{S}$ projected onto the B axis) change and they are also unknown even though the length of L and S are known.

As mentioned, $\vec{J}$ represents the total amount of electron rotary motion and, although more complicated, one can picture the direction of $\vec{J}$ to be opposite to the direction of the magnetic field produced by the total electron motion. The total electron angular momentum then leads to energy levels in a manner similar to that discussed in connection with L and S in previous topics.

The quantum mechanics shows that the length of $\vec{J}$ can have the values

$$J = \sqrt{j(j+1)}\,\hbar \tag{2.15a}$$

where the index j, an angular momentum quantum number, is a non-negative integer or half integer. As with S and L, the angular momentum J is identified by reference to the index j appearing under the square root. The index j is easier to handle and J can be deduced by reference to j. The allowed z-components are

$$J_z = m_J\hbar \qquad m_J = -j,\ -j+1, \dots ,\ j-1,\ j \tag{2.15b}$$

Notice m_j=0 might not be included such as when j=3/2 since then successively subtracting 1 gives the range of m_J= 3/2, 1/2, -1/2, -3/2. The next topic will discuss the addition of angular momenta and spin.

Now return to Figure 2.10 and examine the spectroscopic notation next to the lines in the "LS Coupling" column. The symbols have the form of

$${}^{M}L_{J} \tag{2.16}$$

where M is the multiplicity M=2S+1, S is the total spin, and J is the total angular momentum *index* (i.e., quantum number) for L+S, and L is the orbital angular momentum written as S, P, D, or F [ref. 2.25]. Notice that the letters L and J ***refer to the indices and not the length*** of $\vec{L}$ and $\vec{J}$. Confusing yes, those spectroscopists are an interest bunch. For the figure, Table 2.1 decodes the symbols. Notice M=2 in all cases because there is only one electron in the 5th shell – only one valence electron. It is this electron that allows the RFS to operate. The last column in the table concerns the addition of angular momenta which is covered in the next topic.

Table 2.1: The symbol provides information on L, S, J.

Symbol	Orbital Ang. Mom. L	Multiplicity → Spin	Total Electron Ang. Mom. J
$^2S_{1/2}$	Orbital S → L=0	M=2=2S+1 → S = ½	J=1/2 agrees with (L+S)
$^2P_{1/2}$	Orbital P → L=1	M=2=2S+1 → S = ½	J=1/2 agrees with (L+S-1)
$^2P_{3/2}$	Orbital P → L=1	M=2=2S+1 → S = ½	J=3/2 agrees with (L+S)

Topic 2.2.6: Adding Angular Momenta and Spin

As it turns out, to add angular momentum, add the maximum quantum numbers (i.e., indices) and then sequentially subtract 1 to find the allowed values while keeping the result positive. In particular let j1 and j2 be two angular momenta (actually the quantum numbers a.k.a. indices) then the allowed values for the quantum number (indice) for J=J1+J2 are given by [2.26]

$$J = j1 + j2,\ \ j1 + j2 - 1,\ \ \ldots,\ \ | j1 - j2 | \tag{2.17}$$

Example 2.5: Find allowed angular momenta for the addition of two spin ½ particles
Solution: Spin is angular momentum. Here s1=1/2 and s2=1/2 are the two quantum numbers. Then we have

$$J = ½+½,\ \ ½+½ -1 = 1, 0$$

Notice the subtraction of 1 stops when the result would be less than zero. And so the total spin S can be either 1 or 0.

Example 2.6: Find the allowed J=L+S for L=1 and S=1/2. Also find the allowed z-components of J.
Solution: Equation 2.17 provides J = 1 + ½, 1 + ½ - 1 = 3/2, 1/2
Again the subtraction of 1 stops so as not to make J less than zero. The z-components (i.e., along B) are calculated from Equation 2.15b as

$$m_{3/2} = 3/2,\ \ 3/2 - 1,\ \ 3/2 - 2,\ \ -3/2 \qquad \text{and} \qquad m_{1/2} = ½,\ \ ½ -1$$

and so we find

$$m_{3/2} = 3/2,\ \ ½,\ \ -½,\ \ -3/2 \qquad \text{and} \qquad m_{1/2} = ½, -½.$$

Topic 2.2.7: Hyperfine Energy Levels

We are not yet finished with magnetic fields and angular momentum. The situation becomes more complicated when one realizes that nuclear angular momentum, denoted by I, produces a magnetic field that affects the electron energy and sets the stage for the *hyperfine transitions*. The Benumof writings [2.27] provide a very clear exposition on the hyperfine states. The nuclear angular momentum I refers to the spin of nucleons and to their orbital motion within the nucleus and it couples with J. The total *atomic* angular momentum associated with the atom becomes $\vec{F} = \vec{I} + \vec{J} = \vec{I} + \vec{L} + \vec{S}$. (Is there no end to the capital letters?!? F is not the orbital F!) For each angular momentum F, there will be allowed angles somewhat similar to those for L shown in Figure 2.9. The length of F can take on the following values

$$F = \sqrt{f(f+1)}\hbar \tag{2.18a}$$

where the index f (i.e., quantum number) can be integers or half integers that are not negative . The z-components of $\vec{F}$ are given by $F_z = m_F\hbar$, where the quantum number m_F has the range

$$m_F = -f,\ -f+1,\ \ldots,\ f-1,\ f \tag{2.18b}$$

and where f is decremented by one until reaching –f. Notice that an integer value of f can attain the value of zero but a half-integer f does not attain the value of zero. Sometimes F will be symbolically substituted for f and one must keep in mind that *the reference is to the quantum number and not the length*.

Table 2.2: The Atomic Angular Momentum and states m_F for Figure 2.10. The closed shells (n = 1 through 4) have total angular momentum of zero (J=0) and so only the last electron Rb can contribute nonzero angular momentum.

Rb	I Nuclear	J Atomic				Atomic F = I + J	m_F
		Symbol	S	L	J		
87	3/2	$^2S_{1/2}$	1/2	0	1/2	$\frac{3}{2}+\frac{1}{2}=2$	2, 1, 0, -1, -2
						$\frac{3}{2}-\frac{1}{2}=1$	1, 0, -1
		$^2P_{1/2}$	1/2	1	1/2	$\frac{3}{2}+\frac{1}{2}=2$	2, 1, 0, -1, -2
						$\frac{3}{2}-\frac{1}{2}=1$	1, 0, -1
		$^2P_{3/2}$	1/2	1	3/2	$\frac{3}{2}+\frac{3}{2}=3$	3,2,1,0,-1,-2,-3
						$\frac{3}{2}+\frac{3}{2}-1=2$	2, 1, 0, -1, -2
						$\frac{3}{2}+\frac{3}{2}-2=1$	1, 0, -1
						$\frac{3}{2}-\frac{3}{2}=0$	0

Refer to Figure 2.10 regarding the F states. The nuclear angular momenta quantum number I has the value of 3/2 for ^{87}Rb and 5/2 for ^{85}Rb. So using the addition of angular momenta we find the results in Table 2.2.

The various states in Table 2.2 can be seen in Figure 2.10 except for the $^2P_{3/2}$ sublevels. It should be pointed out that we will only be interested in the hyperfine levels corresponding to the ground state 5S (top line). The 5S levels have L=0 as previously stated (by definition of the S orbital) and so these levels do not participate in the spin-orbit (LS) coupling. The m_F levels are degenerate in energy until an external magnetic field B is applied. The right hand side of the figure shows the field B splits the m_F levels; notice that the ground state $^2S_{1/2}$ F=1 state has m_F=-1 at the higher energy [2.27] while some references reverse it. So now we have the contribution of the nuclear angular momentum to the states.

Similar to the other angular momenta, the F_z hyperfine states split according to the externally applied magnetic fields. For weak magnetic fields, the energy of the hyperfine levels corresponding to each m_F behave similar to the following

$$E_F \sim B\, F\, Cos(\theta) = B\, F_z \sim m_F\, B \tag{2.19}$$

where the tilde '~' should be taken to mean 'goes as' or 'is proportional to', B is the external magnetic field strength, and the angle θ is between the z-axis (i.e., B) and the F vector. The energy increases or decreases linearly with increasing B which means that there is a linear split of levels with increasing B. Figure 2.10 shows this linear splitting for energy levels for slight increases of B. The figure also shows the states with larger m_F have the larger energy except for the F=1 case for 5S. The reason concerns the Atomic Angular Momentum and the magnetic fields as discussed in [2.27].

The RFS uses the sublevels (m_F) with energies E_F that are quadratic [2.5] in B as shown on the far right hand side of Figure 2.10. The *linear* splitting (a.k.a., linear Zeeman effect) occurs for weak applied magnetic fields B. The *quadratic* splitting (a.k.a, quadratic Zeeman effect) occurs for larger B fields and predominates for certain hyperfine states (i.e., certain m_F). The Breit-Rabi formula describes the quadratic splitting [2.27-30].

Keep in mind that for rubidium 87 there is a single valence electron that can occupy any of the states shown and through various means that electron can make transitions between the states. The state $^2S_{1/2}$ is often termed the ground state. The ground state really refers to the lowest possible electron state of the atom (i.e., n=1 in Equation 2.5); that is, for the atom as a whole, it would be the 1S states that form the ground states. But the valence electron in Rb cannot occupy any level with less energy than the n=5 level $^2S_{1/2}$ since under typical conditions, all of the shells n=1 through n=4 are fully filled. That means for the valence electron, the state $^2S_{1/2}$ is the lowest energy state available to it and so it is called the ground state.

For the RFS, we are most interested in the optical transitions between the hyperfine Zeeman-split states. We call them 'optical' even though we might be in the RF region of the spectrum. The word 'optical' means the transition involves a photon that can have wavelengths from visible to RF. Another important point is that only certain transitions are possible even with correct energy. The photon is a spin 1 particle and requires the electron transition occur between states of $L = \pm 1$ and $m_L = \pm 1, 0$. The photon is given either right or left circularly polarization.

Section 2.3: Pumping and Hyperfine Transitions

The present section describes the basics of optical pumping of rubidium 87 to create a 6.83+GHz RF signal in a ^{87}Rb cell (Figure 2.12 reproduced from Section 2.1) that indirectly, through an absorption mechanism, servos a Voltage Controlled Crystal Oscillator (VCXO). The Rubidium Frequency Standard (RFS) employs an ^{87}Rb Lamp and ^{85}Rb Optical Filter for the pump to produce a population inversion within the ^{87}Rb cell. The section shows the basics of how the light transitions electrons between energy levels and the effects of the magnetic field produced by the Cell Field (C-Field) Coil.

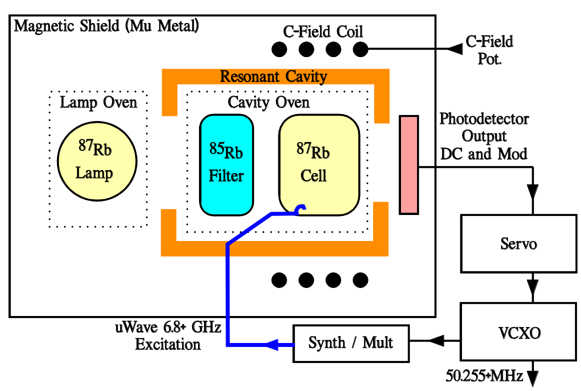


Figure 2.12: Block diagram for an RFS similar to the FE5650A / FE5680.

Topic 2.3.1: Pumping

The rubidium 87 (^{87}Rb) provides a 6.83+ GHz microwave signal that is indirectly used to discipline a Voltage Controlled Crystal Oscillator (VCXO) as discussed in previous sections. The ^{87}Rb lamp (in conjunction with the ^{85}Rb filter) optically pumps the ^{87}Rb atoms (i.e., supplies energy to the atoms) in order to sustain the transitions between energy levels that produce the desired 6.83+ GHz signal [2.5,10,31].

As will be seen, the 6.83+ GHz signal requires the ^{87}Rb valence electrons (Figure 2.13) be excited to higher energy levels such as $^2P_{1/2}$ or $^2P_{3/2}$ by applying 795nm light (for D1 transitions) or 780nm light (for D2 transitions). The D1 or D2 upward transitions require an optical source providing the D1 or D2 wavelengths. The ^{87}Rb lamp in the RFS provides both D1 (795nm) and D2 (780nm). For simplicity, the discussion centers on D1 transitions.

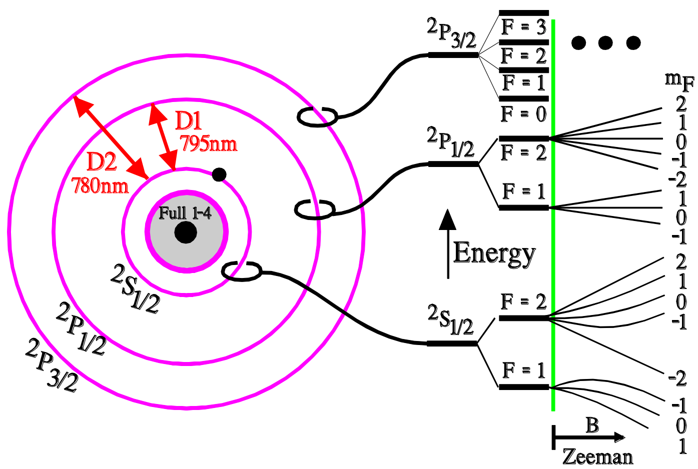


Figure 2.13: A cartoon showing a Bohr-style ^{87}Rb atom with the transitions D1 and D2 used for pumping ^{87}Rb. The cartoon shows the full inner core for the first 4 levels as well as the valence electron in the 5S state. The energy levels on the right side correspond to the lassoed orbitals on the left side (not to scale). Light from the lamp excites electrons from the hyperfine $^2S_{1/2}$ F=1 sub levels (far right denoted by m_F) to the sublevels in $^2P_{1/2}$ (D1 transition) or $^2P_{3/2}$ (D2 transition). The excited electrons then relax to the $^2S_{1/2}$ F=1 and $^2S_{1/2}$ F=2 m_F sublevels. Through continuous action of the lamp, eventually the $^2S_{1/2}$ F=2 levels become more populated than the $^2S_{1/2}$ F=1 sublevels (population inversion).

A conceptual cartoon view of the ^{87}Rb atom appears in Figure 2.13. The 6.83+GHz microwave signal is produced by transitions from the $^2S_{1/2}$ F=2 to F=1 level (actually, their m_F sublevels) as will be seen. But the first question concerns how to place the electrons in the $^2S_{1/2}$ F=2 level for the downward transition to the F=1 level. The left side of the figure shows the D1 transition from one collection of closely-spaced states to another. A transition upward from $^2S_{1/2}$ to $^2P_{1/2}$ requires a photon in the infrared with wavelength of approximately 795nm. The D1 transition can likewise be seen on the right hand side in terms of energy levels. The $^2S_{1/2}$ and $^2P_{1/2}$ states actually consist of F=1 and F=2 levels which are very close in energy and therefore sometimes viewed as a single state such as $^2S_{1/2}$. However, in turn, the F=1 and F=2 states really consist of the m_F sub-states that are ultra-close in energy (ultra-fine states). These F and m_F states are created by the coupling of the spin of the charged nuclear constituents with that of the valence electron (through the magnetic fields). Although we discuss the D1 line as the pump, it is also possible to use D2.

The concept of pumping and emission appears in Figure 2.14 as a stepwise process even though it is really continuous in nature. The pumping process removes electrons from one level and raises them (in energy) to a higher level. Those electrons can spontaneously relax to the lower levels. Therefore, the process ultimately moves electrons from the $^2S_{1/2}$ F=1 to the $^2S_{1/2}$ F=2 level (actually moves them between sublevels of F but not shown). The use of the ^{85}Rb filter will be discussed below. *Panel 1* shows the initial configuration of 4 electrons (out of trillions and trillions…). Because the energy levels are so close in energy, the electrons essentially equally populate them (the difference will be less than roughly 1 ppm) due to thermal energy from the environment. For the present scenario, light with approximate wavelength of 795nm (the D1 transitions) and with specific polarization (right handed) promotes electrons from the $^2S_{1/2}$ F=1 level to the much higher energy levels grouped into $^2P_{1/2}$.

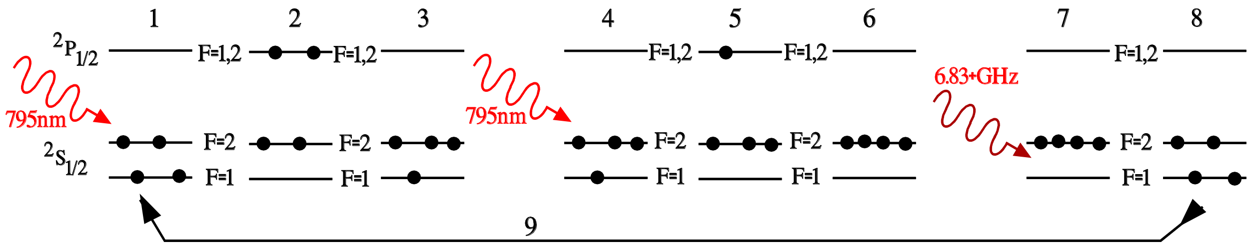


Figure 2.14: The energy levels appear as the straight lines and electrons as the solid black circles. The pumping in this case requires the application of 795nm light as depicted by the red waves at panels 1 and 4 whereas the stimulation of the 6.83+GHz emission requires the application of 6.83+GHz RF as depicted by the dark red wave at panel 7. The emission of the 6.83+GHz RF signal occurs at panels 7 and 8. Notice the electron configuration is the same for panels 1 and 8.

These higher levels have roughly the same energy and collisions between electrons, molecules and sidewalls cause the electrons to distribute uniformly across the various $^2P_{1/2}$ F=1 and 2 sublevels. *Panel 2* shows the excited electrons. The excited electrons have essentially equal probability of dropping to the $^2S_{1/2}$ F=1 and $^2S_{1/2}$ F=2 levels. *Panel 3* shows the resulting configuration of the $^2S_{1/2}$ electrons. *Panel 4* shows the 795 nm incident on the Rb atoms and the $^2S_{1/2}$ F=1 electron can absorb a photon to transition it to the levels at $^2P_{1/2}$ F=1 and F=2. *Panel 5* shows the result of absorbing a photon of 795nm light. Keep in mind that the incident 795nm light absorbs only when there are electrons in the $^2S_{1/2}$ F=1 levels. Therefore a photodetector measuring the 795nm light passing through the ^{87}Rb cell will register maximum light intensity when the electrons occupying $^2S_{1/2}$ F=1 are at a minimum and conversely, it will register minimum light intensity for the maximum number of electrons in $^2S_{1/2}$ F=1. The excited electrons in *panel 5* distribute across all available closely spaced energy levels such as the hyperfine sublevels (not shown). Again the excited electrons spontaneously drop to the $^2S_{1/2}$ levels in equal numbers. The process continues until the $^2S_{1/2}$ F=1 are empty similar to *panel 6*. In such a case, a population inversion exists since the number of electrons in the $^2S_{1/2}$ F=2 level is larger than the number in the $^2S_{1/2}$ F=1 level.

Panel 7 of Figure 2.14 shows the application of a 6.83+GHz RF signal of to the ^{87}Rb atoms in the cell. The applied 6.83+GHz RF signal stimulates the excited atoms (i.e., electrons in the $^2S_{1/2}$ F=2) to emit photons with RF frequency of 6.83+GHz. For the example shown in the figure, the change from *panel 7* to *panel 8* requires the emission of two photons with frequency 6.83+GHz. The *total* RF signal at 6.83+GHz will be larger than the applied RF signal because of this process of *stimulated* emission which is similar to that for a maser or laser. *Panel 8* shows the electron distribution after the RF emission and *step 9* indicates the process starts over. Keep in mind that the sequential view of Figure 2.14 isn't correct since the individual steps simultaneously occur which means the pumping process keeps the $^2S_{1/2}$ F=1 level mostly empty, and the $^2S_{1/2}$ F=2 level mostly full, and the balance is maintained by the continuous application of the 6.83+GHz RF signal. Keep in mind that the 6.83+GHz signal does not directly servo the VCXO. Rather the $^2S_{1/2}$ F=1 electron population has the closer relation with the VCXO servo signal since the photodetector produces the minimum photocurrent when that population absorbs the 795nm light.

A magnetic field applied to the ^{87}Rb cell causes the sublevels (i.e., m_F levels) of $^2S_{1/2}$ F=1 to separate (i.e., Zeeman splitting) and the applied 6.83+GHz RF signal can be controlled to single out the transitions between the two different sublevels ($^2S_{1/2}$ F=2 and F=1) to produce the desired unique microwave frequency ultimately related to the VCXO controller.

Topic 2.3.2: The ^{87}Rb Lamp and the ^{85}Rb Filter

The D1 upward transitions described in the previous topic require an optical source providing the D1 wavelength of 795nm. The ^{87}Rb lamp provides both D1 (795nm) and D2 (780nm). According to a number of references [2.5, 31], the FEI units use D1 to pump the ^{87}Rb Cell and the ^{85}Rb filter removes the unwanted D1 components (refer to Figures 2.13 and 2.15). The D1 light from the ^{87}Rb lamp actually consists of light produced by transitions from the $^2P_{1/2}$ hyperfine levels to the $^2S_{1/2}$ hyperfine levels. Keep in mind that the S and P levels actually consist of the hyperfine levels as shown on the right side of Figure 2.15; that is, strictly speaking, the states S and P do not exist as single states, but rather, given the exceeding small energy difference between the F levels, the collection of F levels give the appearance of a single level on a gross scale. As an example of energy difference in terms of frequency, the figure shows the frequency of the D2 light for a transition between the ^{87}Rb $^2P_{3/2}$ F=2 level and the $^2S_{1/2}$ F=1 level would be

$$384THz - 73MHz + 4.27GHz$$

and from the ^{87}Rb $^2P_{3/2}$ F=3 level to the $^2S_{1/2}$ F=2 level

$$384THz + 194MHz - 2.56GHz$$

The figure does not include the requisite number of digits in the frequencies for an accurate calculation of the transition frequency (refer to the Steck references [1.4.5,6] for the numbers).

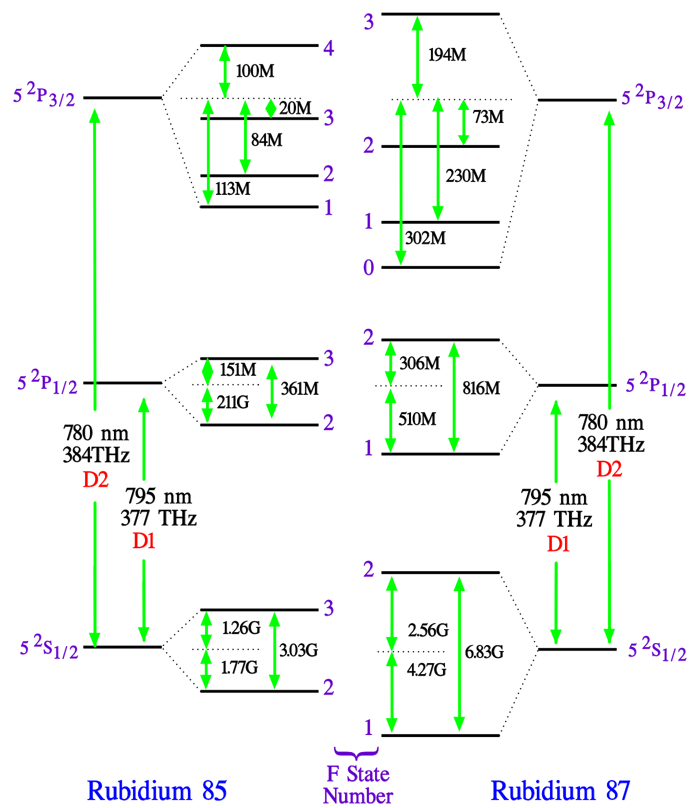


Figure 2.15: Comparison of the ^{85}Rb and ^{87}Rb electron energy levels. The vertical placement has been set to match the D1 and D2 spectral lines on the left and right sides; however the 'the energy between levels' is not to scale. The left side shows the first three orbitals for the valence electron in Rubidium 85 while the right side is that for Rubidium 87. The lines marked as $^2S_{1/2}$, $^2P_{1/2}$, and $^2P_{3/2}$ actually consist of the closely spaced (hyperfine) sublevels marked by the F numbers. Notice the orientation of the right side has been flipped along the horizontal direction to facilitate the comparison of the hyperfine sublevels. The still finer sublevels labelled by m_F do not appear. The frequencies are listed as G = GHz and M=MHz. Keep in mind that the D1 and D2 transitions are really between F hyperfine levels even though they are drawn between the S and P levels. Figure based on Steck diagrams [2.32,33]

Imagine a Rubidium Frequency Standard (RFS) similar to that shown in Figure 2.12 (same as Figure 2.2 of Section 2.1). The ^{87}Rb diagram on the right hand side of Figure 2.15 represents the lamp and the ^{85}Rb on the left side of Figure 2.15 represents the filter. So for Figure 2.15 interpreted as the filter-lamp combination, light emitted from the right side must pass through the energy levels represented by the left side. In what follows, carefully note the use of the ^{87}Rb and ^{85}Rb notation. Suppose we wish to eliminate the 'D2 light'

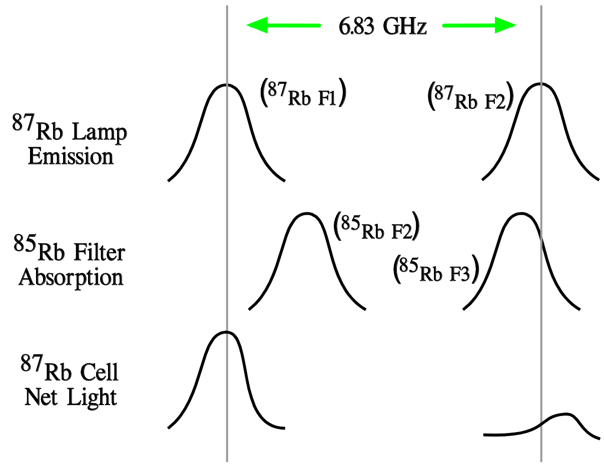


Figure 2.16: A cartoon showing emission and absorption D1 spectra for ^{87}Rb (S F=1 and F=2) and the ^{85}Rb (S F=2 and F=3), respectively. The top portion of the cartoon shows the optical emission from the ^{87}Rb lamp (D1 components). The notation in parenthesis indicates the origin of the optical energy. The middle portion shows the absorption spectrum for an ^{85}Rb filter. Notice the ^{85}Rb absorption has very good overlap with ^{87}Rb emission for ^{87}Rb F=2. The bottom portion shows the optical components of the light entering the ^{87}Rb cell after filtering. The F1 component is still available to pump out the cell F1 level while the lack of the F2 illumination prevents the cell's F2 level from being altered.

emitted by the electron transition from the ^{87}Rb $^2P_{3/2}$ F=3 level to the $^2S_{1/2}$ F=2 level, one would need to have the ^{85}Rb $^2S_{1/2}$ F=3 level line up with the ^{87}Rb $^2S_{1/2}$ F=2 level. The reason is that then the light from the ^{87}Rb transition would be absorbed in making an ^{85}Rb electron transition from the $^2S_{1/2}$ F=3 level to one of the $^2P_{3/2}$ levels. The levels can be made to lineup by adjusting the quantity of inert gasses in the ^{85}Rb filter and the ^{87}Rb lamp and cell.

The function of the filter can be clarified by showing the overlap [1.4.2-1.4.3] between the emission and absorption spectra for the ^{87}Rb lamp and ^{85}Rb filter (Figure 2.16). The top portion of Figure 2.16 shows the *emission* of the D1 spectral components from the ^{87}Rb lamp. Without the filter, these two D1 components could remove electrons from both the ^{87}Rb F=1 and F=2 levels in the ^{87}Rb cell; consequently, the light would not be effective for the pumping mechanism (refer to the discussion in connection with Figure 2.14). The middle portion of the figure shows the *absorption* spectra of the ^{85}Rb at the filter. The portion labelled ^{85}Rb F=3 will almost completely absorb light from the ^{87}Rb F=2 emission. The portion labelled ^{85}Rb F=2 will only negligibly absorb light from the transition marked by ^{87}Rb F=1 because the ^{85}Rb F=2 level is shifted away from the ^{87}Rb F=1 level. The result of the filter absorption of the lamp emission appears at the bottom of Figure 2.16. The lowest two spectra show the light components entering the ^{87}Rb cell. Now there is plenty of optical energy to pump out an ^{87}Rb F=1 level (lower left spectrum) in the cell to produce a population inversion as previously discussed while the lower right spectrum is minimized so as not to pump the F=2 level which would ruin the population inversion.

Reconsider Figure 2.15 but imagine the ^{87}Rb optical source on the left side, the ^{85}Rb filter in the middle and the ^{87}Rb cell on the right. The light from the source propagates from left to right and therefore passes through the filter before pumping the ^{87}Rb cell on the far right. The filter transitions from (^{85}Rb $^2S_{1/2}$ F=3) to (^{85}Rb $^2P_{1/2}$) and (^{85}Rb $^2P_{3/2}$) will remove unwanted components that would otherwise destroy the population inversion between the ^{87}Rb $^2S_{1/2}$ F=1 and ^{87}Rb $^2S_{1/2}$ F=2 states for the ^{87}Rb cell.

Topic 2.3.3: The Transition Requirements

We now consider the actual optical mechanism causing the sublevel selection – the first-time reader can skip this particular digression. The Benumof reference [1.4.1] describes the pump process using right-circularly-polarized 795nm light. The 795nm is the only wavelength (i.e., proper photon energy) able to transition electrons from $^2S_{1/2}$ to $^2P_{1/2}$. The right circular polarization means the electron orbital angular momentum must change by +1, which means the electron must transition from S to P, and additionally the z-component of the angular momentum must change by +1. In such a case, comparing the available $^2S_{1/2}$ and $^2P_{1/2}$ in Figure 2.17, it is clear that there is not any sublevel of $^2P_{1/2}$ F=1,2 with m_F=3. As a result, the 795nm illumination will not remove any electrons present in the $^2S_{1/2}$ F=1 m_F=2 level. Consequently, establishing a population inversion becomes a process of emptying the other sublevels of $^2S_{1/2}$ F=1 (other than m_F=2) although the number in the m_F=2 can and does increase. The electron population created in the $^2P_{1/2}$ states randomize and essentially uniformly populate those states. The downward transitions from these upper levels (i.e., $^2P_{1/2}$) uniformly populate the $^2S_{1/2}$ F=1 and F=2 sublevels including the m_F=2 level. But through continued pumping, as described in connection with Figure 2.14, the population inversion between the m_F=2 state and a lower sublevel will be established. Left-circularly polarized light has a similar effect but it won't empty $^2S_{1/2}$ F=1 m_F=-2 and requires the orbital angular momentum to change by +1 and the z-component (i.e., m_F) to change by -1.

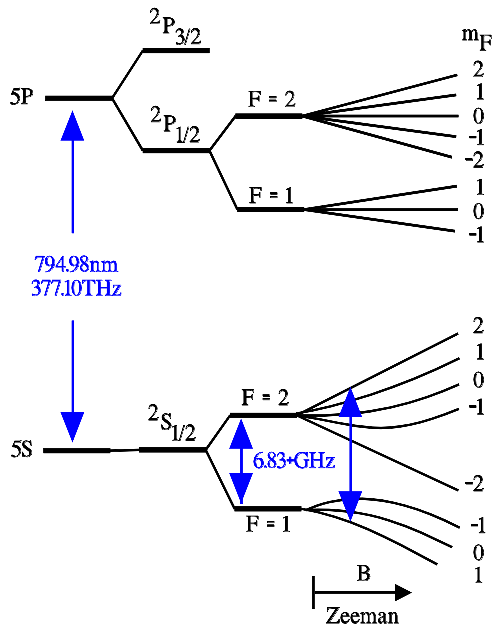


Figure 2.17: (not to scale) The energy levels available to the single valence electron in ^{87}Rb (i.e., 5th shell electron). The D1 line (795nm) corresponds to the energy difference between the 5P and 5S orbitals. The transition between the S F=2 and F=1 creates the microwave 6.83+GHz signal indirectly used to condition the VCXO. Actually the F=2 to F=1 transition occurs between the m_F sublevels indicated on the right side of the '6.83+GHz' label.

Section 2.4: References

[2.1] Frequency Electronics Inc., FEI, "Rubidium Atomic Frequency Standards: FE-5650A, FE-5652A, FE-5660A, FE-5680A,"
http://www.ham-radio.com/wa6vhs/Test%20equipment/FREQUENCY%20STANDARDS/FE-5680A/FEI-5650A,%205652A,%205660A,%205680A%20CATALOG.pdf
Similar but no discussion: http://www.morion.com.ru/uploaded/5650A_Data_Sheet_english.pdf

[2.2] Frequency Electronics Inc. FEI, "Rubidium Frequency Standard Model FE-5650A Series: Operation and Maintenance", Technical Manual TM0107 (1998).
http://www.ko4bb.com/getsimple/index.php?id=manuals&dir=02_GPS_Timing/FEI/FE-5650A
Similar:http://www.wa6vhs.com/Test%20equipment/FREQUENCY%20STANDARDS/FE-5680A/5680%20TECH%20MANUAL.pdf

[2.3] O. Mancini (and J.R.Vig), "Tutorial Precision Frequency Generation Utilizing OCXO and Rubidium Atomic Standards with Applications for Commercial, Space, Military, and Challenging Environments," presentation (slides) at IEEE Long Island Chapter, March 18, 2004.
https://www.ieee.li/pdf/viewgraphs/precision_frequency_generation.pdf

[2.4] E. B. Sarosy, "Output frequency changes in a commercial rubidium clock resulting from magnetic field and microwave power variations," AF Institute of Technology, MS Thesis (1992).
https://apps.dtic.mil/dtic/tr/fulltext/u2/a256466.pdf

[2.5] W. Riley, "Rubidium Frequency Standard Primer"
http://www.wriley.com/Rubidium%20Frequency%20Standard%20Primer%20102211.pdf

[2.6] A. Bennett, "Homodyne Detection in a Laser Locking System", Senior Thesis for the Bachelor of Science at Brigham Young University (2010). Good discussion of lock-in technology.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.592.3207&rep=rep1&type=pdf

[2.7] G. B. Armen, "Phase sensitive detection: the lock-in amplifier" Dept. of Physics and Astronomy, University of Tennessee, Knoxville, TN.
http://server1.phys.utk.edu/labs/modphys/Lock-In%20Amplifier%20Experiment.pdf

[2.8] Bentham Instruments Ltd. "Lock-in Amplifiers" #225.02 or use
http://123.physics.ucdavis.edu/week_4_files/lock-in.pdf

[2.9] R. Eisberg, R. Ressnick, "Quantum Physics of Atoms, Molecules, Solids, Nuclei, and Particles" Wiley, 2nd Ed., (1985) ISBN: 8126508181
Found online at
http://sciold.ui.ac.ir/~sjalali/BSc.Students/modern.physics/Robert_Eisberg,_Robert_Resnick_Quantum_Physics_o.pdf
or
https://www.academia.edu/34441165/Eisberg_R._and_R._Resnick_-_Quantum_Physics_Of_Atoms_Molecules_Solids_Nuclei_And_Particles

[2.10] E. B. Sarosy, "Output frequency changes in a commercial rubidium clock resulting from magnetic field and microwave power variations," AF Institute of Technology, MS Thesis (1992).
https://apps.dtic.mil/dtic/tr/fulltext/u2/a256466.pdf

[2.11] A.Prattis, "LED Wavelength vs. LED Colour", RS-Online 2015.
https://www.rs-online.com/designspark/led-wavelength-vs-led-colour

[2.12] Angular momentum
https://en.wikipedia.org/wiki/Angular_momentum

[2.13] Discussion of Angular momentum
Washington University Physics 432 "Zeeman Effect in Mercury"
http://courses.washington.edu/phys432/zeeman/zeeman_effect.pdf

[2.14] good view of singlets and triplets. Also adding ang mom.
University California Santa Barbara, Physics 432 "Zeeman Effect in Mercury"
https://web.chem.ucsb.edu/~devries/chem218/Term%20symbols.pdf
Correction: remove sqrt from bottom entry in table on page 6.

[2.15] Wikipedia.org, "Atomic Orbitals," https://en.wikipedia.org/wiki/Atomic_orbital

[2.16] Chemguide.co.uk "Atomic Orbitals"
https://www.chemguide.co.uk/atoms/properties/atomorbs.html

[2.17] University of Tennessee, Physics 250 "Multi-electron atoms"
http://electron6.phys.utk.edu/phys250/modules/module%203/Multi-electron%20atoms.htm
And also http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/orbdep.html

[2.18] Vedantu.com "Shapes of Orbitals," https://www.vedantu.com/chemistry/shapes-of-orbitals

[2.19] AK Lectures, "Spin Orbit Interaction (2014)
https://www.youtube.com/watch?v=UI_xLwq_W2U

[2.20] Wikipedia "Angular Momentum Coupling"
https://en.wikipedia.org/wiki/Angular_momentum_coupling

[2.21] Spin-orbit coupling
C. Yip, "Optical Pumping of Rubidium," (2013)
http://home.sandiego.edu/~severn/p480w/OP_CY.pdf

[2.22] D.A. Steck, "Rubidium 87 D Line Data" 2001/2003
https://steck.us/alkalidata/rubidium87numbers.1.6.pdf

[2.23] Advanced Laboratory, Physics 407, Univ. Wisc. "Optical Pumping of Rubidium" (2010)
https://www.physics.wisc.edu/undergrads/courses/spring2018/407/experiments/opticalpumping/opticalpumping.pdf

[2.24] good view of j=L+S
https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Map%3A_Physical_Chemistry_(McQuarrie_and_Simon)/08%3A_Multielectron_Atoms/8.09%3A_The_Allowed_Values_of_J

[2.25] F. M. Walter, "A primer on quantum numbers and spectroscopic notation," Stony Brook University
http://www.astro.sunysb.edu/fwalter/AST341/qn.html

[2.26] L.D. Debbio, "Lecture 15: Addition of angular momenta," University of Edinburgh, Scotland UK, Physics Dept.
https://www2.ph.ed.ac.uk/~ldeldebb/docs/QM/lect15.pdf

[2.27] R. Benumof, "Optical Pumping Theory and Experiments", Am. J. Physics, 33, 151 (1965)
https://www.physics.wisc.edu/undergrads/courses/spring2017/407/experiments/opticalpumping/Benumof_AJP65.pdf
or
https://www.classe.cornell.edu/~hoff/LECTURES/09S_510/S10/Benumof.pdf

[2.28] C. Yip, "Optical Pumping of Rubidium" (2013)
http://home.sandiego.edu/~severn/p480w/OP_CY.pdf

[2.29] T. Dumitrescu, S. Endlich, "Optical Pumping and the Hyperfine Structure of Rubidium 87", Laboratory Manual for Physics 308, Co1umbia University, NY (2007)
http://tesla.phys.columbia.edu:8080/eka/OpticalPumpingLabManual.pdf

[2.30] E. P. Wang, “Optical Pumping of Rubidium Vapor”, MIT Dept. of Physics
http://web.mit.edu/wangfire/pub8.14/oppaper.pdf

[2.31] O. Mancini, Frequency Electronics, Inc. “Tutorial: Precision Frequency Generation Utilizing OCXO and Rubidium Atomic Standards with Applications for Commercial, Space, Military, and Challenging Environments” Presentation for IEEE Long Island Chapter, March 18, 2004.
https://www.ieee.li/pdf/viewgraphs/precision_frequency_generation.pdf

[2.32] Daniel A. Steck, “Rubidium 85 D Line Data,” (2013)
https://steck.us/alkalidata/rubidium85numbers.pdf

[2.33] Daniel A. Steck, “Rubidium 87 D Line Data,” (2013)
https://steck.us/alkalidata/rubidium87numbers.pdf

Chapter 3: Frequency Generation Circuits

The present chapter discusses some of the electronic circuits in the FE-5650A Rubidium Frequency Standard (RFS) from Frequency Electronics Inc. in anticipation of required modifications to allow the units to operate over their full range of frequencies. The first section provides the block diagram and basic circuits [3.1] for the Direct Digital Synthesizer (DDS) board that can produce a desired frequency roughly from DC to near 25MHz although FEI originally shipped the RFS to operate over a smaller range of roughly 6MHz-12MHz. The electronics use fairly standard amplifier and filter designs. The second section details the Analog Devices AD9830A DDS integrated circuit that provides the main functionality for the DDS board.

Section 3.1: Overview of the Relevant Circuit Blocks

The circuits to be modified in the FE-5650A primarily consist of those on the DDS board and a portion of the regulator board as shown in Figure 3.1. The RFS 'physics package' (c.f., Section 2.1) produces the lock frequency (a.k.a., true reference frequency) of 50.255+ MHz where the '+' refers to additional digits. The 50.255+ MHz signal routes to a 74AC161, which divides the frequency by 6 to produce the 8.38MHz clock for a Microchip PIC16F84 microcontroller. Also the 50.255+ MHz signal routes to the AD9830A as a very stable, accurate reference for direct digital synthesis of custom frequencies in the range of 0 to 25MHz (although those above about 15MHz become distorted in the FE-5650A).

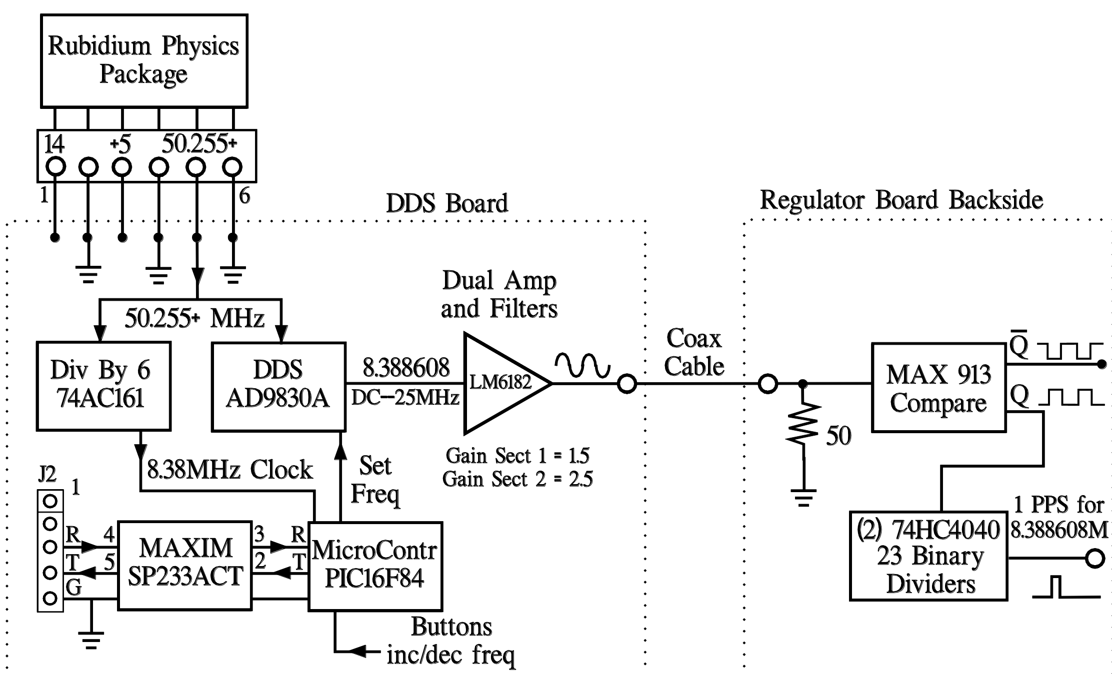


Figure 3.1: Block diagram for the circuits subject to modification for increasing the bandwidth of the FE-5650A Opt 58. After references [3.1-2]. Often FEI sets the AD9830A output to 8.388608MHz in order to produce 1 pulse per second (PPS) at the output of the binary dividers shown on the lower right hand side.

Now consider the functions for the block diagram in Figure 3.1. Keep in mind that the basic idea is to use the 50.255+MHz signal to provide a reference signal for the AD9830A DDS, which produces a sinusoidal signal at a specified frequency Fout, and a clock signal required to clock the PIC microcontroller (uC). The primary purpose of the PIC uC is to set the AD9830A output frequency Fout by transmitting a couple sets of four hex digits as binary over 16 data lines. However interestingly, the uC cannot directly read the reference frequency of 50.255+MHz but instead stores a calibration value in its nonvolatile internal memory along with a representation of the desired output frequency (termed Fcode) – both numbers can be passed to the uC through a serial port. As will be discussed in later chapters, the uC at startup performs a calculation using the stored values to set the AD9830A output frequency.

As mentioned, in order to set the AD9830A output frequency, the PIC uC implements a serial port in order to communicate with equipment external to the RFS. Some versions of the FEI RFS provide connectors on an external panel for the serial port (RS232). The FE-5650A investigated here has the serial port restricted to the DDS board but accessible through connector J2 on the board. A Maxim SP233ACT IC converts between the uC TTL-level serial signals and the traditional RS232 with a voltage range of approximately -10V to +10V. The Maxim chip also inverts the logic levels between the J2 connector and the PIC uC. The TTL-level USART signals can be tapped at pins 2 and 3 for the PIC transmit and receive signals, respectively.

As will be discussed in subsequent chapters, the user sends a HEX number (Fcode) to the serial port at J2 to change the FEI output frequency. Similarly an Fcode and a Reference (Ref or R) number can be sent and the uC can stored them in its own nonvolatile memory. The uC calculates the proper numbers to send to the AD9830A based on the Fcode and the stored Ref. The main purpose of the PIC uC is to send HEX code to the AD9830A to set the desired output frequency. The PIC uC also receives signals from the increment and decrement buttons to increase and decrease, respectively, the output frequency by a single step and saves the new frequency in its own nonvolatile memory. It should be pointed out that only when the FEI unit is instructed, the desired output frequency (HEX Fcode) and any change to the reference frequency (Ref) input through the serial port will overwrite previously saved values in nonvolatile memory. The reference frequency should never be overwritten without very good reason.

The signal synthesized by the DDS AD9830A routes to an LM6182 (dual) op amp and associated filters. To increase the RFS output bandwidth, some of the resistors, capacitors and coils associated with the amplifier need to be modified based on the information found in a well-known, well-written online reference [3.1]. A tap on the DDS board at the output of the amplifier feeding into the coax cable provides a fairly robust sinusoidal signal centered on 0V with drive capability up to about 100mA. The coax cable passes along a milled out region of the front aluminum plate (Figure 1.1) and around to the back side of a regulator board where it is terminated by a 50 Ohm resistor and feeds a Maxim 913 comparator. The Q output feeds a 23 stage binary ripple counter composed of two 74HC4040 ICs to produces a pulse. The pulse rate is Fout/2^{23} which is 1 pulse per second (PPS) when the AD9830A produces the frequency 8.388608MHz. The pulse width is approximately 840nSec. It is possible to tap the other outputs from the counters to obtain different signals with different frequencies. The $\bar{Q}$ output on the comparator can be tapped to provide a square wave at the same frequency as the sinewave from the AD9830A.

Section 3.2: Overview of the DDS Amplifier and Filter

The AD9830A Direct Digital Synthesizer (DDS) produces a sinusoidal wave with a user specified frequency (within the accuracy of its Digital-to-Analog Converter DAC). The DDS AD9830A IC coexists on the DDS PCB with a dual op amp, signal filters, PIC microcontroller (uC) and serial port components among others. Figure 3.2 shows the amplifier and filter circuit for the FE-5680A which, in layout, matches that for the FE-5650A. The diagram is similar to that found in the excellent Reference [3.1] with excellent suggestions and simulations for modifying the bandwidth of the FEI units – a do not miss. We verified the connections shown in Figure 3.2 for the FE-5650A.

The AD9830A feeds the amplifier and filters through coupling capacitors C2, C6, C3 and C5 in addition to other components. The LM6182 dual op amp uses current feedback and thereby holds its gain up to the stated upper limit frequency. The LM6182 operates from a single DC source of Vdd=14 but sets signal ground at roughly half of Vdd by components R12, R13, C9 and C10; the capacitors are sufficiently large to provide a reasonable virtual ground for signals with frequency near 10MHz. The output will be restricted to the range of -7 to 7 volts. The Reference [3.1] also includes the simulated frequency response for each section in the figure. The output from the amplifier-filter circuits connect to a termination resistor, comparator, and ripple counter on the back of the regulator board through a coax cable.

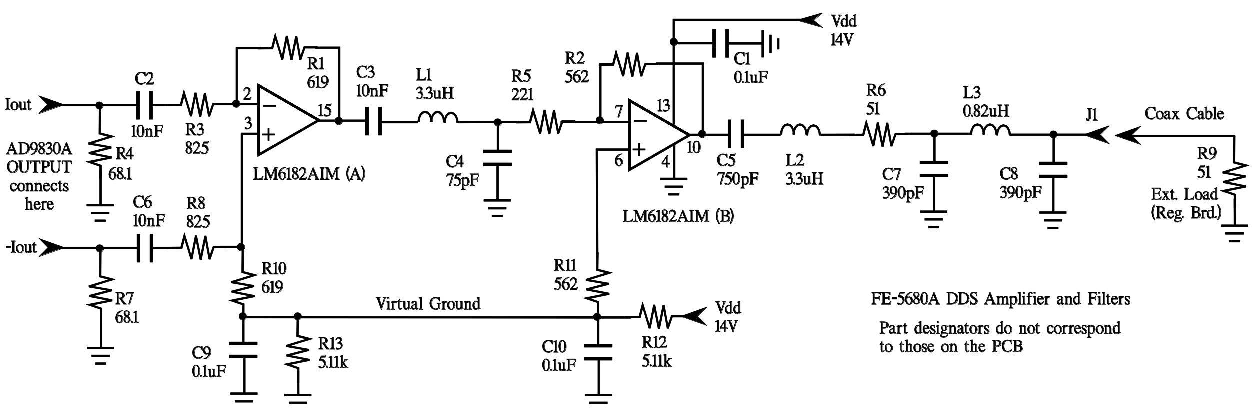


Figure 3.2: Amplifier and filter circuit for the AD-5680A from References [3.1-2]. The layout has been verified but not the component values.

As will be discussed in subsequent chapters, a number of steps can be identified to modify the circuit although not all are required depending on the intended use.

1. The first step in modifying the circuit consists of tapping the connector on the DDS board at the point where the coax cable leaves the board. The amplifier output provides significantly better drive capability than the 'test signal' on the external AMP connector. The references suggest removing the 51 Ohm resistor on the regulator board but then caution to replace the 50 Ohm load at the point where an external circuit uses the signal. We leave the load connected in place. The second step is to increase the ability of the circuit to couple signals.

2. The second step consists of placing 20uF SMD multilayer ceramic capacitors (MCC) across C2, C6, C3 and C5 to provide the greatest increase of capacitance and hence bandwidth while

maintaining the same form factor as the capacitors to be replaced; actually, these MCCs can be soldered on top of those on the PCB. These ceramic capacitors increase the bandwidth to the range 200Hz to 15MHz. The online references suggest using tantalum capacitors up to 100uF or as low as 33uF. These are significantly larger than the multilayer ceramic capacitors and one must be careful of the polarity. The references recommend connecting the positive terminal of C2 to R4, C6 to R7, C3 to pin 15, C5 to pin 10.

3. The third step consists of paralleling the C9 and C10 capacitors with the 20uF SMD multilayer ceramic capacitors to improve filtering at low frequencies. The lower frequency is decreased slightly.

4. A fourth step consists of shorting coil L2 with a piece of wire to improve high frequency signal amplitude. Doing so significantly emphasizes the signal amplitude near 10MHz.

5. A fifth modification consists of tapping desired signals at the comparator and ripple counter on the regulator board to provide various frequency square waves.

Section 3.3: The AD9830

As previously mentioned, the Rubidium Frequency Standard (RFS) incorporates the Direct Digital Synthesizer (DDS) AD9830 integrated circuit (IC) produced by Analog Devices Inc. The AD9830 accepts a reference input signal for which the frequency can be programmatically divided down to the desired output frequency. The IC operates from 5V, accepts clock rates to 50MHz (i.e., the reference), controls frequency to 1 part in 4 billion, and produces a sinusoidal signal using a lookup table and 10bit DAC. The present section provides an overview on the basic operation of the AD9830 [3.3].

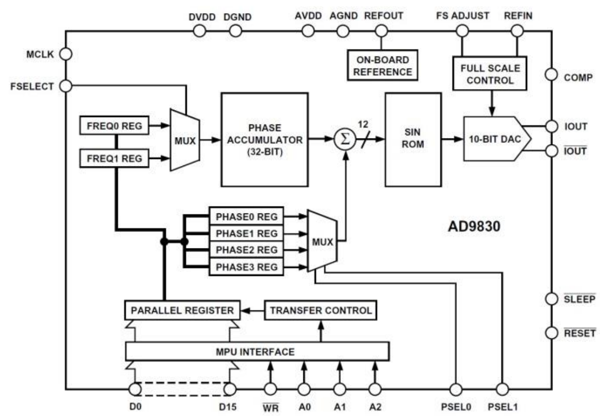


Figure 3.3: Block diagram for the AD9830A Direct Digital Synthesizer (DDS). Diagram from the AD9830 Data Sheet [3.3].

As previously mentioned, the rubidium module produces a signal of approximately 50.255+MHz that provides the true reference frequency R, denoted by F_{MCLK} in the datasheet, to clock the AD9830A. The true reference frequency R is also divided by six to provide the 8.3MHz clock for the PIC microcontroller (uC) that loads the frequency HEX code, *termed phase*, into the AD9830A.

The AD9830A architecture is essentially one of a Numerically Controlled Oscillator NCO [3.4] although the details appear to be masked by the block designated as 'Phase Accumulator' in Figure 3.3. The NCO works with the phase of a sinusoidal wave

$$V(t) = A\,Sin(\omega t) \tag{3.1}$$

where the phase is defined as

$$\varphi = \omega t \tag{3.2}$$

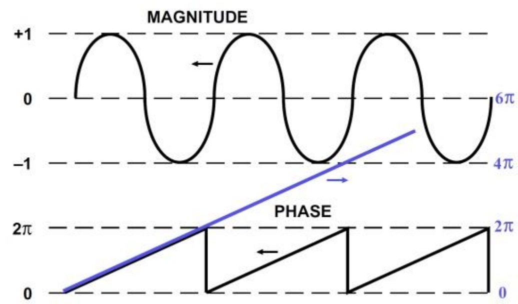


Figure 3.4: Top: A sinusoidal signal. Middle: The phase as a function of time. Bottom: The phase referred to the start of each sinusoidal cycle.

and where the angular frequency ω radians/sec is related to the frequency f in Hz by

$$\omega = 2\pi f \tag{3.3}$$

Figure 3.4 represents the sinewave of Equation 3.1 at the top, the phase in Equation 3.2 as the straight line, and the phase change with respect to the start of each sinewave cycle as the saw wave. The time for the sinewave to complete one full cycle is

$$T = 1/f \tag{3.4}$$

The instantaneous value y of the sine signal ranges between -1 and +1 as shown in top plot of Figure 3.4. Consider the top plot as versus phase $y(\varphi)$ rather than directly versus time. Therefore, if a person knows the phase at time t from the middle plot, then by measuring a distance equal to the phase along the top horizontal axis, the value $y(\varphi)$ can be found by reading the value from the top plot. The process of starting with the phase plot to find the output value closely resembles in concept the use of a 'Look-Up Table' (LUT). Figure 3.3 designates the look-up table as 'Sin ROM'. Either the middle phase plot can be used for the entire sine wave in the top plot or the bottom phase plot can be used with a single cycle of the sinusoid.

The AD9830 (NOC) data sheet describes a single sinusoidal cycle as consisting of smaller 'pieces' referenced by 32 bit numbers (i.e., 4 bytes or equivalently 8 HEX digits). The total number of phase pieces is

$$N = 2^{32} = 4{,}294{,}967{,}296 = (0b)\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111 + 1 = (0x)\ FF\ FF\ FF\ FF + 1$$

where the (0b) near the series of 1's means binary and the (0x) near the series of Fs means HEX digits (see Appendix 1). Each single piece of phase is then a fraction of 2π in the amount of

$$\Delta\varphi = 2\pi/2^{32} \tag{3.5}$$

where of course, 2π refers to one cycle of the sinewave. The current phase is then given by $\varphi_n = n\,\Delta\varphi$ where n is an integer corresponding to the number of elapsed reference clock cycles. With each passing cycle of the reference clock R=f_{MCLK}, the phase number n advances by 1. So after n reference clock cycles, the phase advances to the value

$$\varphi_n = n\,\Delta\varphi \tag{3.6}$$

The NCO accumulator is 32bits wide and holds the cumulative phase value as the number n which has at most 32 bits. If n = (0x) FF FF FF FF (0x denotes hex digits for the 32 bits, refer to Appendix 1), then one full cycle has completed. The next reference clock cycle produces

n = (0x) FF FF FF FF + 1 = (0x) 1 00 00 00 00 => 00 00 00 00

The left most hex digit (1 in this case) corresponds to the number of fully completed cycles and is not important when describing the phase with respect to the start of the cycle (bottom plot in Figure 3.4). The use of a 32 bit register naturally removes the unneeded full-cycle information while retaining the fraction of a cycle which gives rise to the saw wave in the figure. The 32 bit accumulator register can only hold the results of 32 bits (i.e., 8 HEX digits) and therefore rolls over to zero which is 00 00 00 00. The 32 bit register naturally truncates away the number of complete cycles. As an example in HEX, suppose the reference clock R has progressed to n = (0x) 40 00 00 00 then the current phase will be

$$\varphi_n = n\,\Delta\varphi = (0x)40\ 00\ 00\ 00\ *\frac{2\pi}{2^{32}} = 1{,}073{,}741{,}824 * \frac{2\pi}{2^{32}} = \pi/2$$

and so the output sinewave has progressed through a quarter cycle and $Sin(\varphi_n) = 1$. Next suppose the reference clock continues and the accumulator has the number $n = (0x)\ 80\ 00\ 00\ 00$. The phase is now

$$\varphi_n = n\,\Delta\varphi = (0x)\ 80\ 00\ 00\ 00\ *\ 2\pi/2^{32}\ =\ \pi$$

and so the sinewave has progressed through ½ cycle and $Sin(\varphi_n) = 0$. It should be clear that the reference clock advances the phase and hence also the output value of the sinusoidal signal. In concept, n being the number of reference clock cycles, can be thought of as similar to the time. The faster the clock progresses, the faster will the phase $\varphi_n = n\,\Delta\varphi$ cycle through 2π and hence the higher the frequency of the sinusoidal signal.

The NCO must convert the phase to an actual sinusoidal signal. The AD9830A reference clock updates the accumulator which, by the above discussion, can be seen to partly control the output frequency. The signal value (and in part, the frequency) is controlled by a so called look-up table (LUT) labelled as 'Sin ROM' in the block diagram of Figure 3.3. The table consists of cells/registers that hold the output value of the sinusoidal signal. The address n of a memory register corresponds to one of the phase values. So for example, the first entry might be at address 0 (for n=0) and the value at that address would be 0 since then $Sin(\varphi_n) = Sin(n\,\Delta\varphi) = Sin(0) = 0$. Another memory location #m further in the table might have a value such that $\varphi_m = m\,\Delta\varphi = \pi/4$ and so $Sin(\varphi_m) = \sqrt{2}/2$ would be the value stored at location m by the manufacturer. Therefore, as the reference clock proceeds from 0 to m, the address of the memory cell advances from 0 to m (which means the phase proceeds from 0 to $\pi/4$) and the sinewave value is sent from memory (i.e., the LUT) to the output and proceeds from 0 to $\sqrt{2}/2$.

It might appear based on the previous discussion that the lookup table (LUT) has 2^{32} locations but not so. The lookup table has only 2^{12}= 4096 locations which is suitable for a 10 bit DAC. This means that many of the phase numbers n must map to a single LUT cell. This can be handled by using the most significant 12 bits of the 32 bit phase number – the phase number n is truncated to 12 bits.

The Fcode (a.k.a, 'tuning word' or N in the data sheet) also controls the frequency of the output signal; actually Fcode should be considered as the primary control for the frequency. The scheme of advancing the accumulator with each reference clock cycle (R=50.255MHz) would add $\Delta\varphi$ (for each clock cycle) to provide the full 2π cycle in a time of $2\pi/(R\,\Delta\varphi) = 85\,secs$ when $\Delta\varphi$ corresponds to Equation 2.11a. Yikes! Way too slow. The AD9830 adds the 'tuning word' to the accumulator at each clock cycle so as to advance the accumulator at a faster rate – the accumulator skips over some of the numbers n. In fact, using this scheme, if the tuning word is given by $N_{max} = (0\text{x})\text{FF FF FF FF} = 2^{32} - 1$ then the phase at each reference clock cycle will advance according to these simplistic considerations by

$$\varphi_{N_{max}} = (N_{max} + 1)\, \Delta\varphi = (2^{32}) * \frac{2\pi}{2^{32}} = 2\pi$$

which is a full cycle at the rate of the reference frequency. In actuality the maximum output frequency is R/2. As a number-check example, suppose the cycles is divided into two piece instead of 2^{32} pieces. Then the pieces would be indexed by n=0,1 and then N_{max} = 1. One piece of phase would be $[0,\pi)$ and the other $[\pi, 2\pi)$. Then $\Delta\varphi = 2\pi/2 = 2\pi/(\, N_{max} + 1)$.

The AD9830 data sheet provides the following argument to establish a relation among the output frequency Fout, the reference frequency R = F_{MCLK}, and the Fcode = N. In a small time δt, the phase of the output signal advances a small amount $\delta\varphi$ according to Equation 3.2 as

$$\delta\varphi = \omega_{out}\, \delta t \tag{3.7}$$

Substituting Equation 3.3 into 3.7 and then solving for f_{out} provides

$$f_{out} = \frac{1}{\delta t}\frac{\delta\varphi}{2\pi} \tag{3.8}$$

With reference to Equation 3.6, letting n=Fcode provides the phase change as $\delta\varphi = F_{code}\, \Delta\varphi$ for each cycle of the reference clock $\delta t = 1/R$ (from Equation 3.4) where R=F_{MCLK} = 50.255+MHz is the reference frequency for the FE-5650A. Combining these along with Equations 3.8 and 3.5 provides

$$f_{out} = R\ \frac{F_{code}}{2^{32}} \tag{3.9}$$

The next chapters will provide examples for Equation 3.9.

Finally, some of the AD9830 blocks in Figure 3.3 can be discussed. The FE-5650A interfaces an embedded microcontroller PIC15F84 to the MPU Interface block. The microcontroller receives the 32bit Fcode through a serial port in the form of 8 HEX digits 0-F, and it then loads the 32bits into the 'Parallel Register' as 16bits per transfer. The input FSELECT determines whether the FREQ0 or FREQ1 Register will be used for the transfer to the Phase Accumulator. If within the design of the 5650A, the inputs PSEL0 and PSEL1 can be used to transfer an additional phase offset to the output of the Phase Accumulator to offset the scan of the SIN ROM lookup table. The PSEL0 and PSEL1 selects which of the four PHASE REGs will be used as the offset.

Section 3.4: References

[3.1] DDS board: original board schematic and simulations for mods for FE-5680A but DDS same. Includes using serial port, circuit layout on pcb - Very complete.
Matthias Bopp, D.C.Johnson, Deflef, "A precise reference frequency not only for your ham radio station" Rev 1.0 (2013)
http://www.redrok.com/Oscillator_FE-5680A_precise-reference-frequency-rev-1_0.pdf
Vers 0.4 with some Basic programing notes
https://www.fetaudio.com/wp-content/uploads/2009/10/FE-5680A-modifications.pdf

[3.2] Hacking the FE-5650A. Includes information on the power supply, sending and receiving information from the FE-5650A and other notes. Consider making a donation for providing valuable information.
http://www.ko4bb.com/getsimple/index.php?id=manuals&dir=02_GPS_Timing/FEI
http://www.ko4bb.com/getsimple/index.php?id=manuals&dir=02_GPS_Timing/FEI/FE-5650A
Old: can try copy paste: http://www.ko4bb.com/manuals/70.21.206.217/FE_5650A_Opt_58_hack.pdf

[3.3] AD9830 data sheet:
https://www.analog.com/media/en/technical-documentation/data-sheets/AD9830.pdf

[3.4] A variety of links for NCOs and DDS
a. https://www.analog.com/media/en/training-seminars/design-handbooks/Technical-Tutorial-DDS/Section1.pdf
b. https://zipcpu.com/dsp/2017/12/09/nco.html
c. https://www.analog.com/en/education/education-library/technical-tutorial-dds.html
d. https://www.edn.com/design/test-and-measurement/4332832/DDS-design
e. https://zipcpu.com/dsp/2017/12/09/nco.html
f. http://web.eece.maine.edu/~hummels/classes/ece486/docs/NCO_tutorial.pdf
g. https://en.wikipedia.org/wiki/Numerically-controlled_oscillator

Chapter 4: Accuracy, Instability and Allan Deviation

The Rubidium Frequency Standard (RFS) has 1000 fold better accuracy than do the typical crystal based ones. Consequently, the RFS provides a suitable time base to calibrate crystal-based frequency sources. On the one hand for comparison, crystals (XALs) are often considered accurate to within a few parts per million (ppm) and this error can often be decreased with the proper load capacitance. The crystals do age and the frequency does drift even when the initial error is negligible. Further they can be quite sensitive to temperature on the order of 1ppm per degree centigrade (or more). On the other hand, references sometimes cite a fractional instability of 0.01 ppb or less for the RFS that is often interpreted as a measure of the accuracy. The RFS can experience some drift with age as well as some jitter, the latter of which is often more characterized in terms of instability.

The question becomes one of accuracy vs. precision vs. stability. Accuracy tends to focus on a set of samples already collected which can be compared with the true/nominal value whereas stability tends to focus on the deviation of samples in time. Both are then used as predictors of future performance. The chapter first discusses the difference between precision and accuracy, and then how the error is affected by the RFS hardware especially the AD9830A DDS IC digitization and the physics package reference frequency. Afterwards, the chapter discusses the Allan Deviation calculations for ascertaining the instability of the RFS over various time scales (along with other oscillators of course). The normal Allan Dev does not distinguish between drift and jitter. Finally, the chapter describes very handy-to-have software to evaluate the frequency-based ADEV. A variety of well-tested software is available with some capable of using serial and USB ports to directly read from measurement equipment. The software can provide ‘Log-Log’ plots the ADEV or AVAR.

Section 4.1: RFS Accuracy and Error

As mentioned, the Rubidium Frequency Standards (RFS) have 1000 fold better ‘accuracy’ than do the typical crystal based ones. The RFS is often stated to have accuracy near 0.01 ppb. For comparison, crystals (XALs) are often considered accurate to within a few parts per million (ppm) and this error can often be decreased with the proper load capacitance. The crystals do age and the frequency does drift even when the initial error is negligible. Further they can be quite sensitive to temperature on the order of 1ppm per degree centigrade (or more). Temperature changes can be substantially negated by using Temperature Controlled Crystal Oscillators (TCXO) and Oven Controlled Crystal Oscillators (OCXO).

The present section first recalls the difference between accuracy and precision as a precursor to the more quantitative discussion of the types of frequency errors associated with the Rubidium Frequency Standard (RFS). As mentioned, the literature sometimes quotes an inherent RFS inaccuracy on the order of 0.02ppb (i.e., $2*10^{-11}$) which, at 10MHz, is less than 0.001Hz. The digitization processes of the AD9830A DDS chip can produce larger inaccuracies of up to 0.02Hz that are inherently stable at the 0.001 Hz level. As will be seen in the next section, the RFS inaccuracy should be discussed in terms of instability as quantified by the Allan Deviation metric which depends on the averaging time.

Topic 4.1.1: Accuracy and Precision

Consider the contextual difference between the words ‘accurate’ and ‘precise’ for a set of measurements made of some physical parameter. The word ‘accurate’ means the average of the measured values is close to the nominal or standard value. The word ‘precise’ means the measured values are close together (small standard deviation) but the average doesn’t necessarily come close to the nominal/standard value. Notice the word ‘accurate’ also indicates the measurements are fairly reproducible with low variance. For example, a microcontroller clock with measured values in the range 10.000 009 – 10.000 011 MHz has an average value of 10.000 010 MHz that might be considered accurate when the actual value is 10.000 010MHz. The precision might be considered low for these measurements since the values occupy a range of 2Hz. Generally, the standard deviation should be small for all the measurements of clock frequency (small jitter) and the average should be close to the actual/standard value. As another example to help demonstrate accuracy vs. precision, consider an archery target with a bullseye. For accurate archers, the arrows all strike the target close to the center/bullseye with an average position within the bullseye. For precise archers, the arrows all strike close to each other (small standard deviation) but the average position does not necessarily come close to the center/bullseye. The deviation between the average and the desired value refers to a bias. For the target analogy, accurate archers might have large deviation but still the average position of the arrows group close to the center of the bullseye. Frequency jitter produces large standard deviation. The clock can be accurate in the average but quite amiss for the individual measurement – low precision but good accuracy.

Now-a-days, crystal-based timing systems can be found in much of consumer equipment such as computers, cell phones, radios, TV, oscilloscopes, spectrum analyzers, signal generators, frequency counters. Wherever crystal clock systems appear, they will likely need calibration depending on how the frequency drift affects the function of the host equipment. In such a case, the technical user would want to calibrate the clock and would need access to a calibrated standard rather than send the equipment to the factory or service provider for calibration at a cost of a few hundred dollars.

The question becomes what frequency source should be used for the calibration [4.1-2]. As previously discussed, crystals tend to drift with age and have sensitivity to temperature on the order of 1ppm per degree centigrade. This means that if the temperature of a microcontroller circuit changes 3 C, then the crystal frequency changes by 3ppm thereby implying, for example, a 16 MHz crystal would shift frequency by 16 * 3 = 48Hz! The better choice, but at more complexity and higher cost, consists of the Temperature Controlled Crystal Oscillator (TCXO) or better yet, the Oven Control Crystal Oscillator (OCXO). The OCXO consists of a crystal with a heater surrounded by an insulating enclosure that helps maintain constant temperature. The TXCO and OCXO do well to have a precision temperature controller with less than 1 C temperature variation. Despite having temperature control, all of the crystals age and their frequencies drift. Some can be adjusted with internal trimmer capacitors or perhaps by resetting the temperature to a new set point.

More accurate clocks do exist such as the Global Positioning System Disciplined Oscillator (GPSDO), Rubidium Frequency Standard (RFS), or Cesium Frequency Standard (CFS), with inherent errors as small as 0.001 Hz or less. These systems can either be used directly with a microcontroller or frequency multiplier or they can be used to calibrate the crystal based systems (crystals, TCXO, OCXO) employed by the microcontroller. Surplus GPSDO and RFS can be found on EBAY or Amazon in the range of $50 and up at the time of this writing. Similar to the crystal, these oscillators can age whether it’s because the embedded crystal ages or the environment of the working atomic species changes. For example, if the crystal in the RFS physics package feedback loop drifts/ages then locking to the standard will be less probable over time. The rubidium source might lose Rubidium over time causing a problem with locking to the Rubidium signal. The RFS and GPSDO certainly provide calibration to better than the 0.01Hz level (out of 10MHz) at a very reasonable cost for the units.

Topic 4.1.2: The RFS Error/Uncertainty Relations

Consider the AD9830A Direct Digital Synthesizer (DDS) [4.3] integrated circuit that sets the output frequency F_{out} using the following relation from Equation 3.9 in Section 3.3

$$F_{out} = F_{code}\,\frac{R}{2^{32}} \tag{4.1a}$$

where F_{code} is a 4 byte (32 bit) HEX number sent to the AD9830A, R=Ref is the true reference frequency on the order of 50.255+MHz that appears at the RFS Voltage Controlled Crystal Oscillator VCXO. Recall that R (a.k.a. Ref) is derived from the rubidium transition. That is, the RFS servo's the VCXO in such a manner that a multiple of Ref induces electronic transitions (i.e., locks to the rubidium transition frequency) and thereby enables the RFS to output the precision frequency Ref. For future reference, the value of 2^{32} is

$$2^{32} = 4{,}294{,}967{,}296 \tag{4.1b}$$

Suppose one wishes to find the uncertainty in the RFS output frequency F_{out}, denoted by ΔF_{out}, written as $F_{out} \pm \Delta F_{out}/2$. To do so, consider the differential of Equation 4.1a

$$\Delta F_{out} = \Delta F_{code}\,\frac{R}{2^{32}} + \Delta R\,\frac{F_{code}}{2^{32}} \tag{4.2}$$

A small change in F_{out}, namely ΔF_{out}, can be related to small changes in the reference R and to F_{code}, namely ΔR and ΔF_{code}, respectively. According to this last equation, there are two sources of possible error. The first is that Fcode might not be able to take on the exact value required to produce an exact Fout – the value of Fcode can only change by ± 1 at minimum. The second is that the reference frequency R = Ref might not be exactly known especially considering R (when measured in Hz) has six digits beyond the decimal point (i.e., 0.000 001 Hz)

Topic 4.1.3: Digitization and Reference Errors

A couple types of error have relevance with respect to the rubidium standard. As an example, if a typical Allan Deviation Analysis shows the rubidium standard to have an inherent instability of approximately $2*10^{-11}$ which at 10MHz, produces

$$\text{Error} = 2*10^{-11} * 10{,}000{,}000 = 0.0002\text{ Hz}$$

For simplicity, we approximate this error as 0.001 Hz . Notice this error is a percentage of the operating frequency which means the error (Hz) scales with the frequency.

Consider first the DDS digitization/quantization error associated with ΔF_{code}. We are not interested for the moment in the uncertainty of the reference frequency R and so assume $\Delta R = 0$ which means

$$\Delta F_{out} = \Delta F_{code} \frac{R}{2^{32}} \tag{4.3a}$$

We are interested in the change in F_{out} when the frequency-set number F_{code} changes by 1. In such a case, F_{out} would change at minimum by ΔF_{out} and the values between F_{out} and $F_{out} \pm \Delta F_{code}/2$ would not be accessible. So assume that we assign the Fcode uncertainty to be $\Delta F_{code} = 1$ and that the reference frequency is approximately 50255055 Hz, we find the digitization error to be

$$\text{Digitization Uncertainty: } \Delta F_{out} = 0.0117 \text{ Hz} \tag{4.3b}$$

For simplicity, assume the digitization uncertainty is $\Delta F_{out} = 0.02\ Hz$. This means that F_{out} falls somewhere within $\pm 0.01\ Hz$ of the desired frequency. Suppose for example, a person knows R and determines F_{code} to set the output frequency to 10,000,000.000 Hz as close as possible. Then based on the digitization error, the actual output frequency could range from 9,999,999.99 to 10,000,000.01. So it is possible that a person might actually measure the output frequency to be 10,000,000.000 since we can't be sure where in the 0.02Hz interval the actual frequency will be found. Further, given the inherent stability of the rubidium frequency lock, the digitization uncertainty is stable to the 0.001 Hz level. Now if a person dares adjust the c-field potentiometer, it may be possible to remove the discrepancy between the desired and actual frequency.

The digitization uncertainty is not a fixed *percentage* of the operating frequency. For example, if Fout=1Hz, then the digitization uncertainty is roughly 1% (that is, $\Delta F/F = 0.01/1 = 1\%$) whereas at 10MHz, this uncertainty is far less than 1%.

The second source of error resides with the true reference frequency R as expressed through ΔR. We are not interested in the digitization error at this point and so we set $\Delta F_{code} = 0$. Equation 4.2 becomes

$$\Delta F_{out} = \Delta R \frac{F_{code}}{2^{32}} \tag{4.4a}$$

Suppose for the time being, the uncertainty in the reference frequency is $\Delta R = 1\ Hz$ (of course this value is way larger than the actual). Then the uncertainty in the output frequency (at 10MHz) with Fcode ~ 32F0AB00 provides the following results. Refer to the appendices for calculating with hex numbers.

$$\Delta F_{out} = 0.2\ \Delta R \tag{4.4b}$$

So if the reference frequency is known to within $\Delta R = \pm 1$ then the output frequency will be known to within ΔFout = ±0.2 Hz. The error actually scales with the frequency F_{out} because of the term Fcode in Equation 4.4a. So at 5MHz, one would expect 0.1 ΔR.

As a question, what should be the reference error ΔR to make the frequency error in Equation 4.4b equal to that of 0.01 for digitization error? To find the value of ΔR, set $\Delta F_{out} = 0.01$ in Equation 4.4b. Solve for ΔR to see that the reference uncertainty should be less than approximately $\Delta R = 0.05 Hz$. However, if the c-field trimmer potentiometer can be used to offset the digitization error, the uncertainty in the reference frequency should be brought below 0.05 so that the corresponding error decreases to the inherent stability of the Rb mechanism. So to predict the output frequency F_{out}, one needs an accurate measure of Ref or at least some method to work backwards to deduce an accurate value for Ref.

Section 4.2: Allan Deviation Basics

The Allan Deviation ADEV and Allan Variance AVAR are both commonly accepted measures of oscillator instability attributable to drift and various types of noise [4.4]. The behavior of the 'ADEV versus sample averaging time' on a log-log plot can be used to identify the type of noise responsible for the oscillator instability. The present section starts with introductory discussion of the relevant types of noise and how the Allan methods discriminate among these types using the slopes of the log-log plots. After briefly describing the method of frequency averaging, the section then jumps right into the calculation of AVAR and ADEV based on the original Allan formula while providing an intuitive explanation of its meaning through several stylized examples. Next, the section discusses the limitations of using the standard deviation and then describes stationary stochastic processes and their short-comings with respect to the oscillator stability. The section derives the various formulations for the Allan calculations including the Normal (i.e., Classic), Overlapping, and Modified Allan Variance [4.4-13, 4.1].

Topic 4.2.1: Introduction to the Allan Deviation

The Allan Deviation provides a metric for oscillator instability that can be used to compare oscillators. As previously mentioned, the stability of a clock depends on the type of oscillator such as the RC (resistor-capacitor timing) oscillator, crystal oscillator, temperature controlled crystal oscillator (TCXO), oven controlled crystal oscillator (OCXO), rubidium and cesium oscillators (refer to Section 1.2 and Reference [4.2,1]). It should be noted that some oscillators have better short term or long term stability. For example, the crystal oscillator has good short term stability and for this reason, a Rubidium Frequency Standard (RFS) employs a crystal in a feedback loop (i.e., it disciplines the crystal) in order to achieve short and long term stability.

The various Allan Deviation (ADEV) formulations provide the ability to determine various types of noise processes causing the oscillator instability. Generally the oscillator is characterized in terms of frequency or phase. The frequency ADEV that can be calculated from samples of the frequency displayed on a frequency counter. The samples of frequency are labelled as f_i where the i indicates sequential sampling times t_i and $= 1, 2, 3, \ldots$. The frequency based ADEV equations typically use the normalized (i.e., fractional) error frequency and typically denote it by y

$$y_i = \frac{f_i - f_o}{f_o} \tag{4.5}$$

where again, the subscript $i = 1, 2, 3, \ldots$ refers to the first, second, third (etc.) measurement of the oscillator frequency f. The f_o refers to the nominal or expected oscillation frequency. For example, an oscillator designed for $f_0 = 10{,}000{,}000.000\ Hz$ might produce frequency values of 10,000,001 and then 10,000,000 and then 9,999,999 and then 10,000,001 and so on. The fractional error would be

$y_1 = 10^{-7}$, $y_2 = 0$, $y_3 = -10^{-7}$, $y_4 = 10^{-7}$ and so on

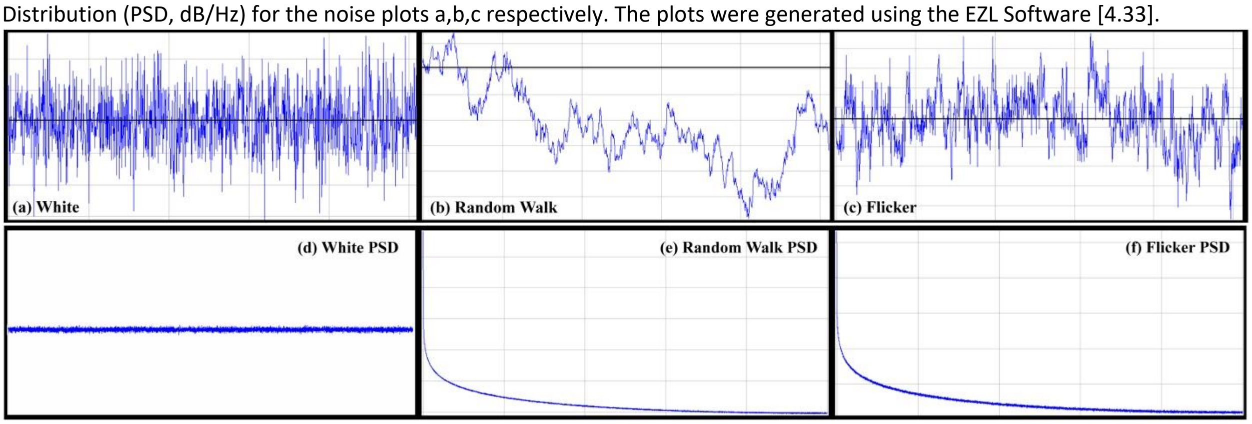
For drift/aging, the fractional frequency can be used to find the expected oscillator error over time by a simple multiplication of the fractional error by a time measure. For example, if an oscillator $f_o =$

$10MHz$ drifts with an expected fractional error of 10^{-10} $per\ year$ then after 1 year, the expected error is 0.001Hz and after 10 years, the error might be 0.01Hz. As will be seen below, the Normal (i.e., Classic) Allan Deviation formalism can be applied to either the error sequence $y_i = f_i - f_o$ or to the frequency sequence $y_i = f_i$ and, because the Normal Allan Deviation involves the difference between adjacent sequence terms, either of these y_i provides the expected error and then if desired, the fractional error can be found by dividing by f_o .

As well known, the cause of oscillator/clock error can be generally attributed to noise and drift (often termed aging). Drift might be reset by internal adjustments such as the load capacitance for a crystal or the c-field for the Rubidium Frequency Standard (RFS). For crystals, the aging can involve a number of mechanisms including the adsorption of environmental molecules or stress and microfractures [4.14-15].

The physical mechanisms responsible for oscillator instability are modelled by several noise sources including those for white and flicker noise and random walk RW [4.16-18]. In Figure 4.1, Panels a, b, and c show these types of noise plotted versus time whereas Panels d, e, and f show representations of the Power Spectral Distribution PSD (dB/Hz) versus frequency as might be seen on an RF Spectrum Analyzer (RFSA).

Figure 4.1: Panels a,b,c: White, random walk and flicker noise plotted versus time. Panels d,e,f show the Power Spectral Distribution (PSD, dB/Hz) for the noise plots a,b,c respectively. The plots were generated using the EZL Software [4.33].



The white noise (Panel 'a') primarily appears in electronic components and minimally in crystals. The RFSA (Panel 'd') shows white noise as essentially independent of frequency over a range of interest. The white noise can be caused by the thermal agitation of charge carriers in electronic circuits [4.16-17]; this motion of charge produces current spikes and can induce voltage fluctuations. Shot noise across semiconductor junctions can produce similar behavior. The white noise in Panel 'a' assumes a Gaussian (i.e., Normal) distribution. White noise is a stationary process in the sense that the average value does not change with time (nor do any of the other moments of the distribution). Consequently, the stochastic quantity (i.e., random variable) will appear to return to the average as the process evolves in time. The Random Walk noise (Panel 'b') for crystals primarily originates in temperature variations; the PSD is expected to drop as $1/f^2$. Notice that the random walk does not necessarily return to the nominal frequency represented by the horizontal line across Panel 'b'. This behavior occurs because the probability that the quantity (i.e., random variable) will take-on a particular value depends upon the previous value (as will be discussed in a subsequent topic). Essentially, the average in the probability distribution changes with time. Behavior such as that in Panel b might be interpreted as poor control/feedback over the crystal temperature (or other parameter). Flicker noise (Panel 'f') appears universally in oscillators but it is not well understood; the PSD decreases as $1/f$. For Flicker noise, notice how the stochastic quantity move away from the average for rather long periods of time but then returns. The behavior of moving away for long periods of time is due to the disproportionately large component of lower frequency components.

The Allan Deviation provides a metric to characterize the instability of an oscillator as well as to discriminate among the noise types. An example of an Allan Deviation log-log plot appears in Figure 4.2 (after Reference [4.5]) for the Normal Allan Deviation ADEV and the Modified Allan Deviation MDEV. For easier viewing, the ADEV plot has been shifted upward compared with the MDEV one. The horizontal axis represents an averaging time τ as determined by the measuring equipment or the software. The greater the averaging, the smaller the (low pass) bandwidth which is on the order of

$$Bandwidth \sim 1/\tau$$

For such a plot, the τ is viewed as a controllable parameter in order that the various types of noise can be filtered out by adjusting τ. Figure 4.2 indicates 'white' noise becomes discernable near $\tau = 4$ on the horizontal axis

$$slope = -\frac{1}{2} \quad near\ \tau = 4$$

and the random walk (RW) near $\tau = 8$ on the horizontal axis

$$slope = +\frac{1}{2} \quad near\ \tau = 8$$

These two regions differ by 4 on the log-log plot which corresponds to a ratio of the two averaging times of 10,000. Interestingly, some circuit designs use the minimum of the log-log plot (if it exists) to deduce the optimum bandwidth for the system to provide the best stability.

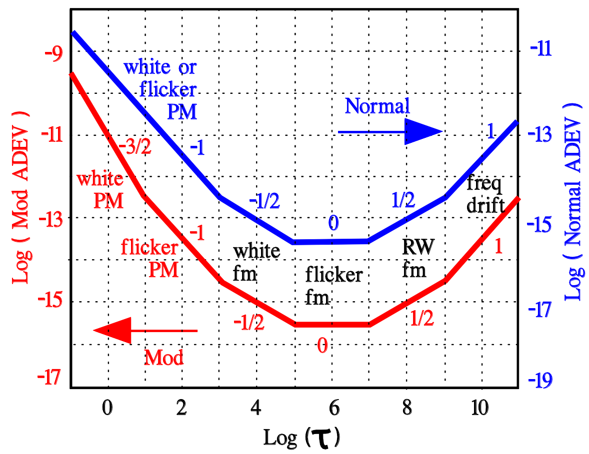


Figure 4.2: Plot of the Modified and Normal Allan Deviation [4.5,9], denoted by MDEV and ADEV, respectively. The MDEV plot discriminates white and flicker phase noise whereas the ADEV does not.

As mentioned, the log-log plot discriminates the various types of noise based on the slope [4.5,21-25]. A constant slope obtains when the ADEV versus τ satisfies a power law of the form

$$A = b\,\tau^m \qquad \text{where A=ADEV} \tag{4.6a}$$

where b and m are constants and m will become the slope of the log-log plot. The log-log plot place the horizontal and vertical grids so as to take into account the logarithms while showing the ADEV and τ without the log function. The question becomes one of finding the slope of a straight line segment on a log-log plot. As is well known, when the vertical and horizontal log decades have equal size such as in Figure 4.2, the slope of a log-log plot is essentially measured with a *ruler* (rise over run). However, in many if not most, the vertical and horizontal decades differ. The slope m can then be found by the following considerations. Take the logarithm (base 10) of Equation 4.6a to find

$$Log(A) = Log(b) + m\,Log(\tau) \tag{4.6b}$$

Find two points that coincide with the line segment say (τ_1, A_1) and (τ_2, A_2) and substitute into Equation 4.6b to find

$$Log(A_1) = Log(b) + m\,Log(\tau_1) \tag{4.7a}$$
$$Log(A_2) = Log(b) + m\,Log(\tau_2) \tag{4.7b}$$

Solving Equations 4.7 for the log-log slope m provides

$$m = \frac{Log(A_2) - Log(A_1)}{Log(\tau_2) - Log(\tau_1)} \tag{4.8}$$

The log-log slope m does not change when the value of τ is scaled by a factor. Suppose, as discussed more in the next section, the measuring equipment requires time τ_o to render a measurement and further suppose software averages over n samples so that the averaging time will be $\tau = n\,\tau_o$ (i.e., the value τ_o is scaled by the factor n). It can be seen that the log-log plot can plot against τ or n without changing the value of the log-log slope m which is required to distinguish the type of noise. Visually, the $\tau = n\,\tau_o$ produces a shift of the horizontal axis between n and τ

$$Log(\tau) = Log(n\,\tau_o) = Log(n) + Log(\tau_o)$$

and hence does not affect the shape and slopes. This can also be seen from Equation 4.8 by substituting $\tau_2 = n_2\,\tau_o$ and $\tau_1 = n_1\,\tau_o$ and noting the resulting two $Log(\tau_o)$ terms in the denominator cancel out.

The following topics discuss the Allan Deviation as applied to the fractional frequency since this will be the quantity of interest for those using a highly accurate frequency counter.

Topic 4.2.2: Introduction to Calculating AVAR and ADEV

Consider the calculation of the Normal Allan Variance AVAR and Allan Deviation ADEV as metrics for oscillator instability [4.4-8, 4.1, 4.11-13]. The present topic calculates a single number for the metric. A subsequent topic extends the calculations to define the Normal Allan Variance and Deviation that depend on an averaging parameter τ. As previously discussed, the values of AVAR and ADEV as a function of τ can be plotted on a log-log plot to deduce the instability of the oscillator and, in particular, the types of noise causing the instability. For now, we show the simplest formulas and provide some examples to gain insight into their meaning and usage.

The Allan Deviation calculations require samples of the frequency over a period of time as might be obtained from an accurate frequency counter. Frequency counters typically admit a sinusoidal signal through a timed 'gate' to cycle-counting circuits. The gates typically have set times of 0.1, 1 or 10 seconds. Often, the counter increments the count when the sinusoidal signal passes through zero. The cycle counting essentially measures a difference in phase truncated to the nearest cycle. This means that one cycle of the incoming signal might not be counted (and generally the frequency counters do not count fractional cycles) and therefore the gate time places limits on the counter accuracy. For example, 1 second gate time for a 10MHz oscillator might cut off 1 cycle out of 10,000,000 which means the best measurement would have an error of approximately 1Hz. A 10 second gate time could reduce the error to 1 cycle out of 100,000,000 which provides 0.1Hz error. Other frequency-measurement instruments might directly compare the oscillator signal phase with that from a highly accurate oscillator to obtain the frequency. Regardless, the equipment requires a finite change in phase to calculate a frequency and hence, a minimum measurement time τ_o. The measured frequency is therefore an average over the time interval τ_o. As a note, we assume the test instrument has much better stability and accuracy than the oscillator under test.

An example plot of an 'instantaneous' frequency versus time appears in Figure 4.3 for an oscillator under test. Notice the use of the symbol y_i to represent the average frequency for the i^{th} sample rather than something more conventional like f_i. The use of y_i appears to be traditional but most often represents a normalized (a.k.a., fractional) frequency commonly written as $(f-f_o)/f_o$ where f_o is the nominal or average frequency; the normalized frequency simply represents the fractional departure from the nominal frequency and it does not have units. For the present discussion, we use the notation y_i to mean the average frequency over the i^{th} interval τ_o and it has units of Hz. We reserve the notation $\bar{y}_i$ for the case when the procedure averages over multiple y_i. The average can also be written with the angular brackets such as $\langle y_i \rangle$. The set of y_i, denoted by $\{y_i\}$, provides the sample sequence for the Allan formalism.

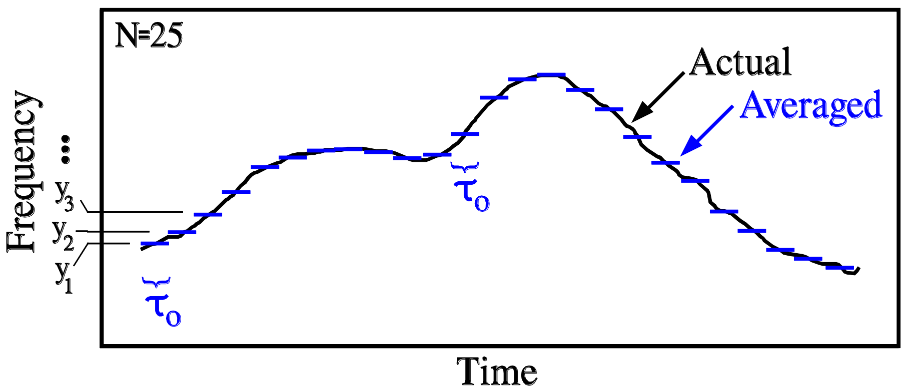


Figure 4.3: Example plot of the instantaneous frequency (solid curve) from an oscillator under test vs. the elapsed time. The instrument averages the signal over time τ_o as shown by the short horizontal lines. The averaged frequencies are labeled as the y_i on the frequency axis. For the example plot, the instrument provides N=25 samples.

The Allan Variance, often denoted by AVAR or alternatively by σ_A^2, is sometimes written as

$$\sigma_A^2 = \frac{1}{2}\langle\ (y_{i+1} - y_i)^2\ \rangle \qquad \text{or} \qquad \sigma_A^2 = \frac{1}{2}\langle\ (\ y(t+\tau_o) - y(t)\ )^2\ \rangle \tag{4.9}$$

where $y_i = y(t_i)$ are the frequency samples taken for each τ_o interval, that is $t_{i+1} = t_i + \tau_o$. Notice there cannot be any dead time between each τ_o for these calculations. The average in Equation 4.9 should be over all times (infinite) but that's obviously impossible. The data must be therefore analyzed as a collection of N frequency samples with the AVAR estimated as

$$\sigma_A^2 = \frac{1}{2}\ \frac{1}{N-1}\sum_{i=1}^{N-1}(y_{i+1} - y_i)^2 \ = \frac{1}{2}\ \frac{1}{N-1}\{\ (y_2 - y_1)^2 + (y_3 - y_2)^2 + \cdots + \ (y_N - y_{N-1})^2\} \tag{4.10}$$

Figure 4.3 shows an example of N=25 samples. The N-1 appears at the top of the summation symbol because N samples provide N-1 adjacent pairs. For example, the two pairs for N=3 can be listed as $y_3 - y_2$ and $y_2 - y_1$.

The Allan Deviation, often denoted by ADEV or alternatively by σ_A, is the square root of the Allan Variance.

$$\sigma_A = \sqrt{\sigma_A^2} = \sqrt{\frac{1}{2}\langle\ (y_{i+1} - y_i)^2\ \rangle} \tag{4.11}$$

INTERPRETATION: **The Allan Deviation is the RMS average of the 'distance' between adjacent frequency values y_i.** Here 'distance' provides a visual cue and really means the 'difference' $y_{i+1} - y_i$.

As previously mentioned, typically the Allan Deviation makes use of the fractional frequency (often called normalized) defined as

$$y_i = \frac{f_i - f_o}{f_o} \tag{4.12}$$

where f_o is the nominal frequency and so $f_i - f_o$ appears as an error term. Also as previously mentioned, AVAR formalism can also be applied to $y_i = f_i - f_o$ or to $y_i = f_i$. For simplicity and ease of interpretation we initially provide examples using $y_i = f_i$.

So now, a few examples are in order. Example 4.1 shows a sequence of numbers obtained from the literature along with numerical results. The calculations, although simple, help solidify the exact nature of the formulas and the type of averaging and maybe just as important, make it possible to check the accuracy of automated procedures such as computer programs. Example 4.2 helps provide interpretation and understanding for ADEV as an RMS (root-mean-square) measure of the distance between successive data points (even though the points are not random). A subsequent section will show how to distinguish between the various behaviors by further averaging techniques and the slope of log-log plots.

Example 4.1: Find the Allan Deviation for the numbers listed in Table 4.1 column 2.
Solution: Equation 4.10 indicates that one should first form the differences $y_{i+1} - y_i$ and then square that difference $(y_{i+1} - y_i)^2$, and then add up all those squared differences. Table 4.1 shows the N=8 data points and the mentioned calculations to obtain

$$AVAR = \sigma_A^2 = \frac{1}{2}\ \frac{1}{N-1}\sum_{i=1}^{N-1} (y_{i+1} - y_i)^2 \ = \frac{1}{2}\ \frac{1}{8-1}\ 450.79 * 10^{-12} = 32.20 * 10^{-12}$$

And the Allan Deviation is the square root of the Allan Variance: ADEV = $\sigma_A = 5.67 *\ 10^{-6}$

Table 4.1: Numbers for Example 4.1

i	**y_i x 10^{-6}**	**$(y_{i+1}$-$y_i)$ x 10^{-6}**	**$(y_{i+1}$-$y_i)^2$ x 10^{-12}**
1	43.6	---	---
2	46.1	2.5	6.25
3	31.9	-14.2	201.64
4	42.1	10.2	104.04
5	44.7	2.6	6.76
6	39.6	-5.1	26.10
7	41.0	1.4	1.96
8	30.8	-10.2	104.04
		Total	*450.79*

Example 4.2: Suppose N samples of the frequency all produce the same value of 'f' for i=1 to N. Find the Allan Variance and the Allan Deviation.

Solution: Since y_i = f for every sample i, we then have

$$y_{i+1} - y_i = f - f = 0$$

and then Equation 4.10 provides

$$\sigma_A^2 = \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1}(y_{i+1} - y_i)^2 \;= 0$$

and then also the Allan Deviation has the value of zero. The zero makes sense because the frequency samples all have the same value – there isn't any deviation between them – no scatter.

Example 4.3: Find the Allan Deviation for the three sets of samples shown in Figure 4.4. Notice how the last two sample sets increase/decrease without bound and yet have the same finite Allan Deviation as does the first one. The reason is that the distance between successive points is the same for all the three cases. The results emphasizes that the Allan Deviation is an RMS measure of the 'distance' between successive points. A subsequent section will show how to distinguish between the various behaviors in Figure 4.4 by multiple sample-averaging, which extends the two point method described here, and viewing the results on a log-log plot.

Solution: All panels have N=13 points with identical point spacing.

Top Panel: Adjacent points differ in frequency by $y_{i+1} - y_i = \pm 1$. Equation 4.10 provides

$\sigma_A^2 = \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1}(y_{i+1} - y_i)^2 \;= \frac{1}{2}\,\frac{1}{12}\sum_{i=1}^{12}(\pm 1)^2 = \frac{1}{2}$

And so the Allan Deviation ADEV becomes $\sigma_A = 0.707$ for the top panel in Figure 4.4.

Middle Panel: Again, adjacent points differ in frequency by $y_{i+1} - y_i = +1$. Equation 4.10 provides

$$\sigma_A^2 = \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1}(y_{i+1} - y_i)^2 \;= \frac{1}{2}\,\frac{1}{12}\sum_{i=1}^{12}(+1)^2 = \frac{1}{2}$$

And so again the Allan Deviation ADEV becomes $\sigma_A = 0.707$ for the middle panel in Figure 4.4.

Bottom Panel: As before, adjacent points differ in frequency by $y_{i+1} - y_i = -1$. Equation 4.10 provides

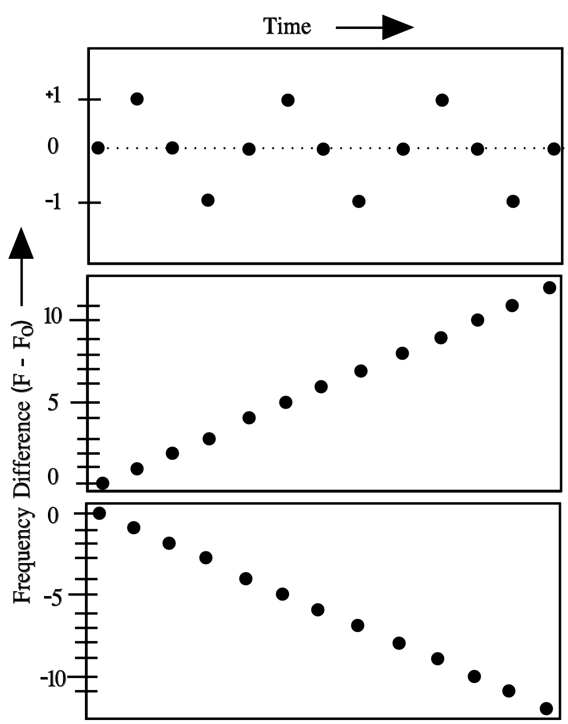


Figure 4.4: Time plots of $f_i - f_o$ which is the frequency difference from the nominal value. The plot uses points rather than horizontal lines for convenience - *do not* consider the time between points to be 'dead time'.

$$\sigma_A^2 = \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1}(y_{i+1} - y_i)^2 \;= \frac{1}{2}\,\frac{1}{12}\sum_{i=1}^{12}(-1)^2 = \frac{1}{2}$$

And so again the Allan Deviation ADEV becomes $\sigma_A = 0.707$ for the bottom panel in Figure 4.4.

We will find that although the three very different panels reduce to the same single number ADEV, the first plot can be distinguished from the last two plots by extending the Allan Deviation to include more than two-point averaging and then viewing the results on a log-log plot. The software section describes available software to simulate the various possibilities [4.43].

Example 4.4: Return to Figure 4.3 and calculate the Allan Variance and Deviation for the 25 samples having the values

1, 2, 4, 6.5, 9.3, 10.5, 11, 11.3, 10.8, 10.2, 10.5, 13, 17,
18.8, 19, 17.5, 15.5, 12.7, 9.6, 7.8, 4.3, 2.3, 0.2, -0.8, -1.8

Solution: Set up a table similar to Table 4.1. The sum of the square calculates to be 96.7. Then the Allan Variance and Deviations from Equations 4.10 and 4.11 become

AVAR = 2.01 and ADEV = 1.42

Topic 4.2.3: The Standard Deviation

Various writers stress the importance of a measure of the sample variation that converges regardless of the duration of the sampling time and number of samples. The Allan Deviation solves the problem by averaging (RMS) the difference between adjacent samples as previous discussed.

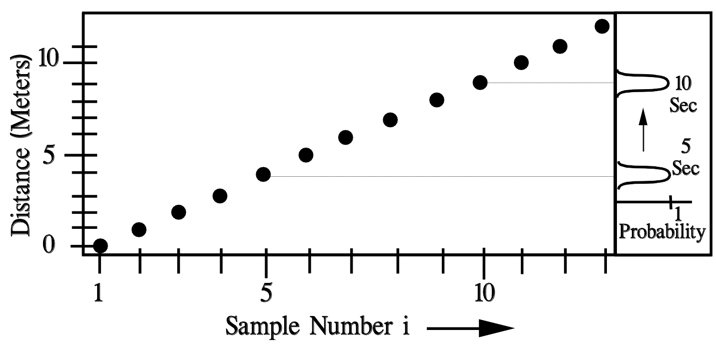


Figure 4.5: A marble rolls along a straight line toward the north. A sample is recorded once each second as shown. The standard deviation of ALL samples will increase with time. The right side shows the probability distribution (sideways view) depends on time: its average moves north but the width does not change.

The present topic shows how the standard deviation can fail to converge with increasing sample times and sample number. The standard deviation is interpreted as the RMS distance between the sample points y and the *average* of the data points $\bar{y}$; that is, the standard deviation describes the scatter of sample points around the average value. The standard deviation is the square root of the standard variance σ_S^2 which is defined by

$$\sigma_S^2 = \langle\ (y - \bar{y})^2\ \rangle \tag{4.13a}$$

For the collection of data points, the best estimate for the standard variance has the form (often called an estimator)

$$\sigma_S^2 = \frac{1}{N-1}\sum_{i=1}^{N}\ (y_i - \bar{y})^2 \tag{4.13b}$$

where

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{4.13c}$$

To see how the standard deviation can increase with time, consider the situation shown in Figure 4.5 for a marble rolling along a straight line toward the north. Assume that each sample requires 1 second to measure and coincidentally the marble rolls at a rate of 1 meter per second just to keep things easy. First calculate the standard deviation (estimator) for all data points collected in the first 5 seconds and then in the first 10 seconds. The average for the first 5 seconds and 10 seconds, respectively, is $\bar{y}(5) = 2.0$ and $\bar{y}(10) = 4.5$. Then the standard variance for 5 seconds and 10 seconds, respectively calculates to be $\sigma_s^2(5) = 2.5$ and $\sigma_s^2(10) = 9.2$ and hence the corresponding standard deviations are $\sigma_s(5) = 1.6$ and $\sigma_s(10) = 3$. The standard deviation increases with time simply because the outlying points move further away from the *average* (despite the fact that the *average* increases/moves too). Here the calculated standard deviation involves all of the samples taken up to some specified time which means the RMS average distance between the marble and the *average position* is increasing. The situation with the marble is somewhat similar to the drift of frequency for an oscillator-under-test; however, generally the oscillator will require months to show appreciable drift and there will be additional noise to make the measured value less certain (a lot less).

As shown by measurement, the *position* of the marble (Figure 4.5) very predictably moves further from the starting position which causes the calculated standard deviation to increase. The right-hand side panel in Figure 4.5 shows the probability distribution for the marble whereby the average moves toward the north but the width of the distribution remains unchanged. 'The' standard deviation is usually defined as related to the width of the probability distribution. In the present case, the probability distribution is shaped such that the probability of finding the marble at position is either 0 or 1 because the width of the distribution is very narrow. (A more mathematical way of saying the same is that the probability of the marble being found at north position j at time t is either 1 or 0 according to P(j,t) = 1 when t=j+1 and zero otherwise where j is the distance.) So for example, at t=5 seconds, the marble will be found at 4 meters north with near 100% probability as shown in the figure. Now here's the issue. The standard deviation related to the width of the probability distribution does not change whereas the calculated one based on the sampled positions does change. So what happened? In some cases, calculating a statistical quantity over time is not the same as knowing that quantity at one particular time. The issue is related to stationary and nonstationary and ergodic processes. In particular, the process of the rolling marble is not ergodic in that the average is not independent of time and the calculation of the average and standard deviation based on the collection of samples does not agree with that for the probability distribution at a particular time.

Topic 4.2.4: Comments on Stationary and Ergodic Processes

People typically tend to think in terms of 'stationary processes' and in particular, ergodic ones where the average and standard deviation can be calculated for the probability distribution at any given time by calculating them from a collection of measured samples as in Figure 4.6 (see Refs [4.26-27] and the previous and next topics). The discussion briefly addresses some concepts necessary for random processes.

A process refers to the sequential measurement of a quantity for which each realized value (for each time) is guided by a probability distribution function. The value of a quantity q at a particular time

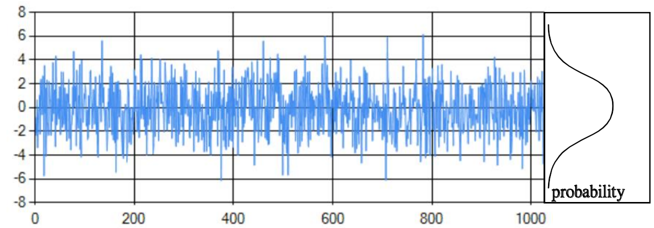


Figure 4.6: A stationary process guided by a normal probability distribution. Each probability distribution is the same for each time (i.e., the probability distribution is independent of time).

can potentially be any one of a variety of values although a measurement at that time will produce a single value. In this sense, for each time t such as t_1, t_2, … , the quantity q is thought of as a different random variable denoted by q_1, q_2, …. The collection of these random variables forms the 'process' which is the long way of referring to the quantity q(t) such as the y for the previous topics. The point of considering random variables such as q_i is that a person can discuss the *average and standard deviation at a particular time* t_i. *At a given time*, the variable $q(t_i)$ can *potentially take on any value in a range of values consistent with a probability distribution*. Therefore $q(t_i)$ will have a corresponding expected value and a standard deviation at the time t_i. For example, the value at t=400 in Figure 4.6 could take on any value between roughly -8 to +8, but the range from -2 to +2 is much more likely. In principle, each different time t_i will have its own distinct probability distribution for q – the average or deviation could differ similar to Figure 4.5 which is unlike that for Figure 4.6. The discussion in connection with Figure 4.5 showed the average value over the collection of samples (i.e., the so called time average) was not the same as the expected value of the distribution at any *particular* time (i.e., the so called ensemble average). The process in the figure is not ergodic otherwise the time average and standard deviation would be the same as the ensemble average and standard deviation (within some small random error). As a note, even for ergodic processes, when dealing with measured values of a quantity such as voltage or frequency, there will be slight random variations of the measured mean and standard deviation because the measured values have some degree of randomness (otherwise there would not be any need for the mean or standard deviation). The calculated mean or standard deviation won't be precisely the expected theoretical values simply because the measured values have some degree of randomness.

An ergodic process is a stationary one (i.e., independent of time) in such a manner that the average and standard deviation (and other moments) are independent of time (Figure 4.6). For the process shown in Figure 4.5, the probability function depends on time and so the average and variance can be expected to depend on time as well. Figure 4.6 shows a hypothetical example of frequency variation (or marble location) for a time-independent Gaussian distribution for 1024 samples taken at a rate of 1 sample per second. The motion stays centered on the position y=0 because the average is independent of time – it's almost as if there is a 'feedback mechanism' or a 'restoring force' that keeps the points centered – the feedback could be part of the system and the time independent probability distribution manifests that feedback. A slight drift would appear as a slight upward (or downward) slope but that would mean the probability distribution has a time-dependent mean. The software [4.43] in conjunction with this book provides an opportunity to see how the stationary distribution produces the Allan deviations and how the multiple-point averaging looks on a log-log plot.

Topic 4.2.5: Random Walk

Aside from the stationary processes such as those guided by a normal or uniform distribution, there are nonstationary ones, besides systematic drift, such as the Random Walk which is sometimes observed in conjunction with frequency measurements. Figure 4.7 shows a hypothetical example for a random walk of the frequency variation although it might be easier to think of a marble on a ladder. The short horizontal lines represent possible frequency steps or ladder rungs. Each state is labeled by a pair of numbers (t,y) where the second entry refers to the frequency increment (or rung of the ladder) and the first number refers to the time. Although the figure shows multiple columns of identical states, in reality, there is only a single set of states (a single ladder for the marble) but that single set is drawn as a column for each tick of the clock (1 second per tick) in order to show an increase or decrease in frequency (or ladder rung for the marble) as the clock ticks. The system starts at t=0 with the frequency F(0)=10,000,000 Hz (or the marble 10 meters above the ground); these numbers correspond to (0,0).

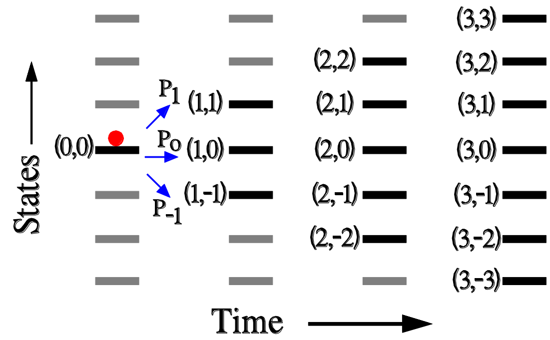


Figure 4.7: An example for a random walk process. The marble can move up or down by no more than 1 step.

Figure 4.7 shows the frequency (or marble) can make a transition to one of three states as determined by probability (Table 4.2). The three probabilities must sum to one as in $P_1+P_0+P_{-1}=1$. As evident from the table for this example, the frequency value or ladder rung can only change by 0, ± 1 for any given initial state at any given time t, regardless of whether the time is t=0 or some other value such as t=100. Notice the gray horizontal lines. For this example, they represent states that definitely cannot be accessed at a particular time because the probability of a transition from a previous possible state to the gray one is zero. So at t=0, only three states can be accessed at t=1 and the others are therefore grayed-out in the figure. Similarly at t=2, the top most and bottom most states have been grayed-out.

Table 4.2: Transition probabilities for the 3 step random walk.

Probability	**Description**
P_0	Probability of transition from (t,y) -> (t+1, y) that is, y(t+1)=y(t) P_0 is the probability that the frequency will remain unchanged (or the marble remains on the same rung) when the clock advances by 1 second.
P_1	Probability of transition from (t,y) -> (t+1, y+1) that is, y(t+1) = y(t)+1 P_1 is the probability that the frequency will increase by +1Hz (or the marble moves up one 1 rung) to produce F(1)=10,000,001 when the clock advances by 1 second.
P_{-1}	Probability of transition from (t,y) -> (t+1, y-1) that is, y(t+1) = y(t)-1 P_{-1} is the probability that the frequency will decrease by 1Hz to F(1)=9,999,999 (or the marble will drop 1 rung) when the clock advances by 1 second.

Based on the fact that the accessible states increase with time (i.e., the boundary between the gray and black horizontal lines moves away from the center) indicates that the standard deviation will depend on time [4.28-29]. Figure 4.8 shows an example random walk for 50,000 seconds (not associated with the previous example). It would appear the system does not have very good feedback to restore the system to equilibrium at y=0.

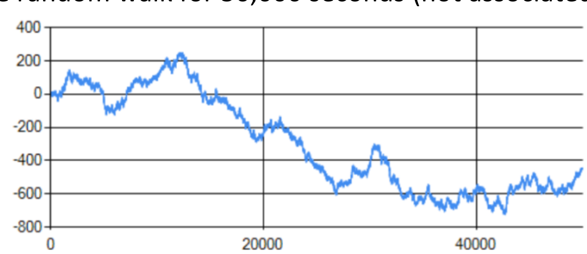


Figure 4.8: An example random walk for 50,000 seconds.

To see the variance for the random walk is proportional to the elapsed time, consider the following statistical model. Let Y_t be the random variable at time t such that when Y_t is measured, it provides one of +1, 0, -1 for the step up, none, down as consistent with the probability distribution. Then the total distance Z_t from the zero base-line will be a sum of these random variables. For example, Z_2=Y_2+Y_1 and so if Y_1=1 and Y_2=0 then the total displacement will be Z_2=1. After t ticks of the clock, the total displacement will be

$$Z(t) = \sum_{i=1}^{t} Y(i) \tag{4.14a}$$

Given that each Y(i) is guided by the same probability distribution, the variance of each will be the same and denoted by $\mathrm{Var}_Y = \mathrm{Var}(\,\mathrm{Y(i)}\,)$, and each Y(i) is independent of the previous in assigning +1, 0, -1 we have

$$\mathrm{Var(Z)} = \sum_{i=1}^{t} Var\{Y(i)\} = \sum_{i=1}^{t} Var_Y = Var_Y \sum_{i=1}^{t} 1 = t\, Var_Y \tag{4.14b}$$

where the first equality follows by the Y_i being independent and the second equality because each Y_i has the same probability distribution. Consequently, one can see that the standard variance is proportional to the elapsed time. By the way, the variance is viewed as the scatter at time t. Because the probability distribution is not stationary, calculating the variance or standard deviation across all samples to time t will not necessarily be the same as the theoretical variance at time t as is easy to see since near t=0, Equation 4.14b shows variance is smaller then. Equations 4.14 can be used to calculate the Allan Variance of an oscillator when the frequency counter has similar instability [4.30].

Example 5.5: (a) Show the value of the Normal Allan Variance σ_A^2 remains bounded for a random walk for the three accessible steps shown in Figure 4.7. (b) What is the expected value for σ_A^2?
Solution: (a) Equation 4.10 provides an upper bound

$$\sigma_A^2(N) = \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1} (y_{i+1}-y_i)^2 = \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1} (\pm 1)^2 = \frac{1}{2}$$

where the maximum displacement of $y_{i+1} - y_i = \pm 1$ has been used to obtain the upper bound on the summation. The lower bound of zero obtains when $y_{i+1} = y_i$ for every i. As a result to part (a), the Allan Deviation for Figure 4.7 remains within the bounds of

$$0 \le \sigma_A \le 1/\sqrt{2}$$

(b) Next, the expected value of σ_A^2 can be found by ensemble averaging the $(y_{i+1}-y_i)^2$ without regard for the particular value of i (i.e., without averaging over time). The probability of $y_{i+1} - y_i = \pm 1, 0$ is independent of time – the average is made over the pairs $y_{i+1} - y_i$. Recall, the average value of a discrete function f(x) can be written as

$$\langle f \rangle = \sum_x f(x)\, p(x)$$

where p(x) is the probability of a particular x. In this case, $x = y_{i+1} - y_i$ and so x can take on the values of +1, 0, -1 with the respective probabilities of p_1, p_0, p_{-1} where $p_1 + p_0 + p_{-1} = 1$. Consequently the expected value of σ_A^2 can be written as follows. Notice the average is applied to $(y_{i+1}-y_i)^2$ without regard to the index i.

$$\langle\, \sigma_A^2 \,\rangle = \langle\, \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1} (y_{i+1}-y_i)^2 \,\rangle = \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1} \langle\, (y_{i+1}-y_i)^2 \,\rangle$$

$$= \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1} \{\, (+1)^2 p_1 + (0)^2 p_0 + (-1)^2 p_{-1} \,\}$$

Make the substitution $p_1 + p_{-1} = 1 - p_0$ and note that the summand is independent of i to find the answer of

$$\langle\, \sigma_A^2 \,\rangle = \frac{1}{2}\ \frac{1}{N-1}\sum_{i=1}^{N-1}\{\, 1 - p_0 \,\} = \frac{1 - p_0}{2}$$

Given that p_0 can range from 0 to 1 means that $\langle\, \sigma_A^2 \,\rangle$ can range from 0 to ½ which matches the bounds found for $\sigma_A^2(N)$. Keep in mind that $\langle\, \sigma_A^2 \,\rangle$ is a single number that will fall within the bounds of 0 to ½.

Topic 4.2.6: Concepts for the Allan Variance

The Allan Variance and Allan Deviation should not necessarily be understood as a single number as was done in Topic 4.2.2 above. The Normal Allan Variance AVAR and Deviation ADEV depend on an averaging parameter τ. The values of AVAR(τ) and ADEV(τ) as functions of τ can be plotted on a log-log plot to deduce the instability of the oscillator and, in particular, its cause in terms of the types of noise such as flicker, Gaussian, random walk, and others. The time parameter τ provides for variable filtering/averaging in the sense of setting the system bandwidth (Figure 4.9). The parameter τ would be the software equivalent, for example, of switching a frequency counter among 0.1, 1, and 10 second gate times. As mentioned, the function $\sigma_A^2(\tau)$ can be viewed on a log-log plot to discern the various types of noise.

The literature shows many different types of Allan Variance and Allan Deviation including the Normal (i.e., Classic) and Overlapping, and Modified variety. The various types primarily differ according to the type of averaging used. The references [4.31-38] provide links to free but highly usable software for calculating the time-dependent Allan Deviation (along with lots of other goodies).

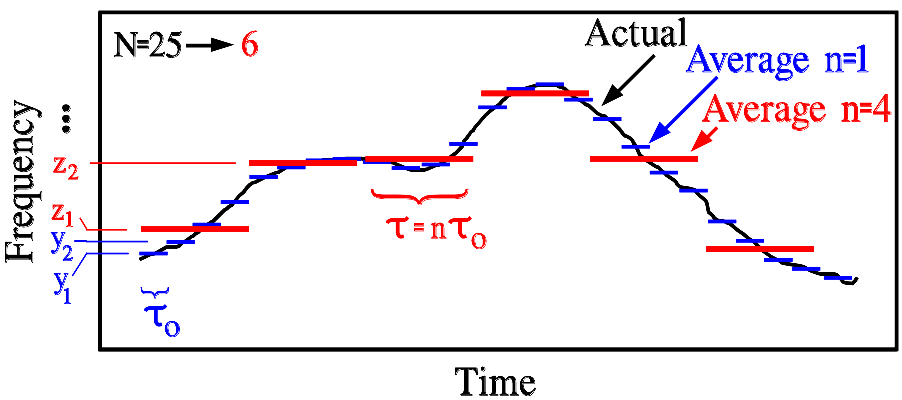


Figure 4.9: Plot of the actual frequency from an oscillator under test vs. the elapsed time. The bandwidth of the measuring instrument causes the instrument to average the signal over time τ_o. The long horizontal lines represent the average of n=4 samples.

As mentioned, the Normal Allan Variance introduces a time parameter τ related to software averaging as a type of bandwidth control. Recall that the measurement of each frequency sample requires a time interval τ_o as shown in Figures 4.3. Now however, we average n of those samples so that the averaging interval becomes $\tau = n\,\tau_o$ where n can be 1, 2, 3 and so on. The equivalent bandwidth for the equipment becomes roughly $F = \frac{1}{n\,\tau_o}$. The long red horizontal lines in Figure 4.7 represent an example of the averaging time $\tau = 4\tau_o$. That is, if $\tau_o = 0.01\ seconds$ then the software will provide an average value for the sampling time of 0.04 seconds and the equivalent bandwidth decreases from a value on the order of 100Hz to 25Hz. Of particular interest and importance, the fast variations of the frequency plot will be removed; that is, those variations with frequencies larger than approximately $\frac{1}{n\,\tau_o}$ will be filtered out. In this manner, the Allan Variance and Allan Deviation as a function of $\tau = n\,\tau_o$ provide a variable low-pass filter which therefore makes it possible to distinguish various types of noise.

The Allan Variance can be denoted by any of AVAR(τ), AVAR(n), $\sigma_A^2(\tau)$, and $\sigma_A^2(n)$ depending on which is most convenient.

Similar to Equation 4.9, the time-dependent Allan Variance will be written as

$$\sigma_A^2(\tau) = \frac{1}{2}\langle\ (z_{i+1} - z_i)^2\ \rangle \qquad \text{or} \qquad \sigma_A^2(\tau) = \frac{1}{2}\langle\ (\ z(t+\tau) - z(t)\ )^2\ \rangle \tag{4.15}$$

where $z_i = z(t_i)$ are the AVERAGE over time τ of the frequency samples y_i taken for each τ_o interval. The present topic will examine the parts of this last equation and cast the equation into something suitable for a sequence of data points. First however, consider some of the symbols. The time between averages is $t_{i+1} = t_i + \tau = t_i + n\,\tau_o$ where n=1, 2, …. Perhaps the better notation would be to set $z_i \to \overline{y_i}$ in order to realize z is an average. The notation appears in Figure 4.9. The average is theoretically over all times (infinite) which is obviously impossible in real situations. Assume we collect a total of N samples of the frequency. Each average z_i consumes n of those points. Figure 4.9 shows the original 25 frequency samples are partitioned into 6 groups of 4 points which enter into each averaged point z (here the averaging time is $\tau = 4\,\tau_o$). Notice point #25 is not used. As will be discussed below, the estimate of the Allan Deviation becomes

$$\sigma_A^2(\tau) = \frac{1}{2}\,\frac{1}{M-1}\sum_{i=1}^{M-1}\ (z_{i+1} - z_i)^2 \tag{4.16a}$$

where $M = Int\left(\frac{N}{n}\right)$ is the number of groups containing n samples, and z_i is based on the figure and can be written as

$$z_i = \frac{1}{n}\sum_{j=(i-1)n+1}^{i\,n} y_j \tag{4.16b}$$

At this point in the discussion, refer to Example 4.6 below to see how the averaging works based on Figure 4.9.

The next topic provides several discrete operators for the sequence { y_i } and then shows how to build the Overlapped and Normal Allan Deviations using these operators. The development helps understand the metrics.

Example 4.6: Using the numbers for the τ_o intervals in Example 4.4, calculate $\sigma_A^2(4\,\tau_o)$ for the longer horizontal lines in Figure 4.7.
Solution: First divide the 25 samples of the original data from Example 4.4 into sets of four:

1, 2, 4, 6.5,	9.3, 10.5, 11, 11.3,	10.8, 10.2, 10.5, 13,
17, 18.8, 19, 17.5,	15.5, 12.7, 9.6, 7.8,	4.3, 2.3, 0.2, -0.8,
-1.8		

Eliminate the last single data point. Average each of the 4 samples in the six sets to find the z_i.

3.4, 10.5, 11.1, 18.1, 11.4, 1.5

Now treat these as 6 samples (N=6) and apply Equations 4.16 to find

AVAR(4) = 24.3 ADEV(4) = 4.9

Topic 4.2.7: Overlapping Allan Deviation

We now develop the Overlapped Allan Deviation using a couple of discrete operations on the sequence { y_i }. The next topic provides similar for the Normal Allan Deviation.

Let the sequence y_i for i=1 to N be the samples provide by the measuring instrument with each sample requiring the time τ_o as usual. The samples are most frequently the fractional frequency but as previous discuss, the frequency or the error can also be used. The symbol y_i can also be written as y(i) to make the sequence easier to read as well as match notation in some references.

Consider a new sequence in index k, denoted by Y(k;n) (note the capital Y), defined as the *average* of n samples y(i) starting at index k. Y is similar to z in Equation 4.16b except Y does *not* skip over sample blocks.

$$Y(k;n) = \frac{1}{n}\sum_{i=k}^{k+n-1} y_i \qquad (4.17)$$

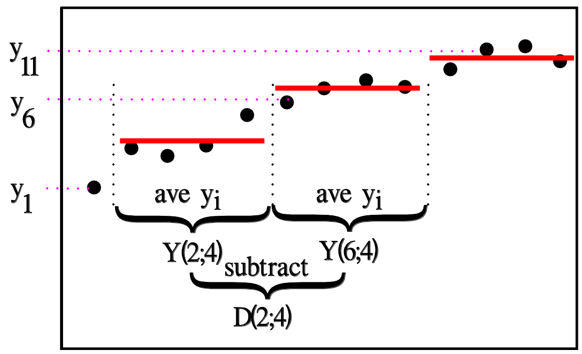


Figure 4.10: An example of D(2;4):
$D(2;4) = Y(2+4;4) - Y(2;4)$

The sequence of Y(k;n) could also be written as Y_k with the n suppressed. Notice especially that Y(k;n) is the average value of the n samples starting at index k. The n should be considered a fixed number for the following discussion until $\sigma_A^2(n)$ is rendered on the log-log plot – most of our examples will consider n=4. Figure 4.10 shows the notation. Notice the number of elements in the sum is

$$1 + \{end - start\} = 1 + \{(n+k-1) - k\} = n$$

where 'end' and 'start' refer to the starting and ending index on the summation in Equation 4.17. *Sometimes a collection of contiguous samples* { y_i } *will be termed a 'block' of sample points*.

Next we define a sequence of numbers D(k,n) that subtracts the difference of two averages.

$$D(k,n) = Y(k+n;n) - Y(k;n) \qquad (4.18)$$

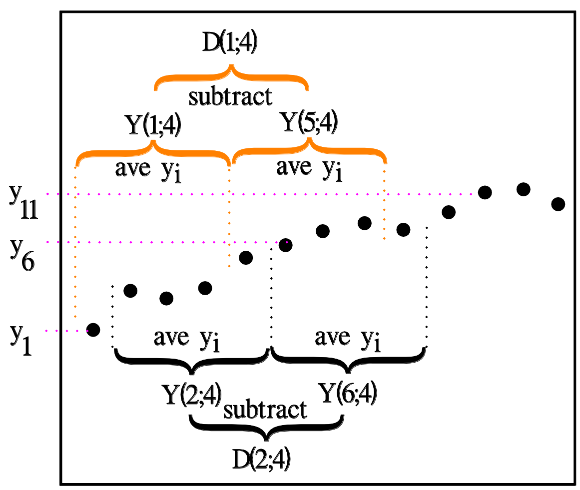


Figure 4.11: The block of samples associated with D(1;4) (red at top) moves to the right by 1 step to D(2;4). The D(1;4) and D(2;4) blocks overlap. The block of samples for D consists of two sub-blocks.

D is the differencing operation between adjacent averages of n samples. $Y(k;n)$ averages a block of n samples starting at index k. $Y(k+n;n)$ averages the next block of n samples starting at k+n (i.e., jumps over the first block). Figure 4.11 shows an example for the case of D(k=2;n=4). Notice for D(k=2,n=4), the relevant block samples { y_i } starts at index k=2 and continues across two sub-blocks each with n=4 samples. If the index k is incremented by 1, then the boundaries for the entire block (along with the two sub-blocks) move to the right by one. Of course the values of Y and D will change. Further, when D is squared, we see that the result begins to resemble Equation 4.15. Notice in particular as shown in Figure 4.11, adding 1 to the index k does *not* cause the computation to jump over the first n samples (i.e., the first sub-block of data) unlike what would be required for the Normal Allan Deviation. To be more specific, Figure 4.9 and Equation 4.16b show for n=4, the averaging operation consumes a block of 4 samples into z and the summation skips to the next sample sub-block; however, to the contrary, Equations 4.17 and 4.18 do not skip over a block when k is incremented. Figure 4.11 shows an example of k=1 (and n=4) at the top. Adding 1 to k=1 causes the operation to select the block starting at k=2 as shown in the bottom portion of the figure. In this manner, the blocks for subsequent indices k can overlap each other and therefore the data points are not necessarily fully independent for different k. It can be seen that the index k can be increased until the block of n samples includes the very last sample point on the right and so maximum value of k is $\mathrm{k}_{max} = \mathrm{N} - 2\mathrm{n} + 1$. For Figure 4.11, the max value of k_{max} would be $\mathrm{k}_{max} = 13 - 2 * 4 + 1 = 6$.

The Overlapping Allan Variance is defined as

$$\sigma_O^2(n) = \tfrac{1}{2} \langle \ (D(k;n))^2 \ \rangle_{ave\ k} \tag{4.19a}$$

where the average < > involves the quantities with index k in Equation 4.18. The estimator for the Overlapping Allan Deviation in Equation 4.19a is

$$\sigma_O^2(n) = \frac{1}{2(N-2n+1)} \sum_{i=1}^{N-2n+1} [\, Y(i+n;n) - Y(i;n)]^2 \tag{4.19b}$$

Now we need to substitute for the Y to obtain a useful result in terms of y_i suitable for a computer. Inserting Equation 4.17 then provides the desired result

$$\sigma_O^2(n) = \frac{1}{2\, n^2\, (N-2n+1)} \sum_{i=1}^{N-2n+1} \left[\sum_{j=i}^{i+n-1} y(j+n) - y(j) \right]^2 \tag{4.19c}$$

Continuing, we next show the Normal Allan Variance denoted by $\sigma_A^2(n)$ or AVAR. The Normal Allan Variance is defined to skip past each block of n samples once having averaged them.

Topic 4.2.8: Normal Allan Deviation

.

Now we can extend the Overlapping Allan Variance to the Normal Allan Variance. One simple method would be to rearrange the indices for the overlapping case so that the blocks move through n points for each increment of the index k. We'll follow another method that has the same effect.

Define the operation G(k;n) on a sequence of averages Y(k;n) (and hence also the y_i) according to the following [4.31-32 manual]:

$$G(k;n) = D(\,(k-1)n+1;n\,)$$
$$= Y(kn+1;n) - Y(\,(k-1)n+1;\;n\,)$$
(4.20)

where Equation 4.18 was substituted for D. The sequence G essentially redefines k to specify the block of n samples to be averaged. So k=1 refers to the difference between the first and second blocks of n averaged samples, and k=2 refers to the difference between the second and third blocks of n averaged samples. Now k essentially labels a block of samples rather than each element in the sequence y_i. Figure 4.12 shows an example. So essentially G provides terms such as $z_{k+1} - z_k = G(k;n)$ in Equation 4.15.

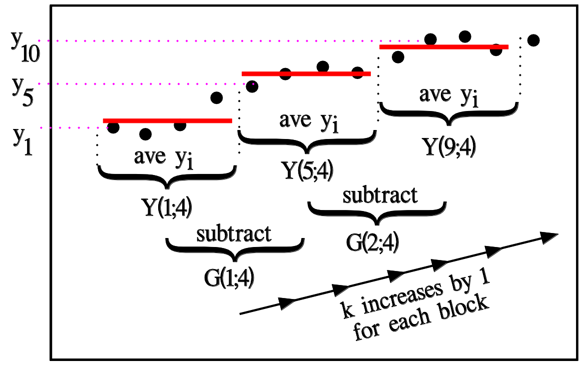


Figure 4.12: The k in G(k;n) labels each block of samples. G provides the difference between the average of adjacent blocks of samples.

The Normal Allan Variance is defined in terms of G as follows

$$\sigma_A^2(n) = \tfrac{1}{2}\,\langle\;(\,G(k;n)\,)^2\,\rangle_{ave\;k} \qquad (4.21)$$

where k is associated with the averaging. Writing the estimator for the average and substituting the expression for G from Equation 4.20 provides (suppress "; n" in Y for easier reading)

$$\sigma_A^2(n) = \frac{1}{2}\,\frac{1}{M-1}\sum_{k=1}^{M-1}[\;Y(kn+1) - Y(\;(k-1)n+1\;)\;]^2$$
(4.22)

where $M = Int\left(\frac{N}{n}\right)$ and N is the total number of samples y_i. The Normal Allan Variance in Equation 4.22 can be written in terms of the samples y_i by substituting Equation 4.17

$$\sigma_A^2(n) = \frac{1}{2}\,\frac{1}{(M-1)}\sum_{k=1}^{M-1}\left[\frac{1}{n}\sum_{i=kn+1}^{(k+1)n} y_i - \frac{1}{n}\sum_{i=(k-1)n+1}^{kn} y_i\right]^2$$
(4.23a)

The indices can be rearranged to yield

$$\sigma_A^2(n) = \frac{1}{2}\,\frac{1}{n^2\,(M-1)}\sum_{k=1}^{M-1}\left[\sum_{i=(k-1)n+1}^{kn}(y_{i+n} - y_i)\right]^2$$
(4.23b)

or as

$$\sigma_A^2(n) = \frac{1}{2}\,\frac{1}{n^2\,(M-1)}\sum_{k=1}^{M-1}\left[\frac{1}{n}\sum_{i=1}^{n} y_{kn+i} - \frac{1}{n}\sum_{i=1}^{n} y_{(k-1)n+i}\right]^2$$
(4.23c)

where $M = Int\left(\frac{N}{n}\right)$ is the integer part of N/n.

Example 4.7: Use Equation 4.23c to show the cases of n=1 and n=4 for N=25 give the correct results.
Solution: Consider the two cases as follows

Case n=1: The value of M is then M=Int(25/1)=25 and the equation reduces as follows:

$$\sigma_A^2(n) = \frac{1}{2(M-1)} \sum_{i=1}^{M-1} \left[\sum_{k=1}^{1} y_{i+k} - \sum_{k=1}^{1} y_{(i-1)+k} \right]^2 = \frac{1}{2(M-1)} \sum_{i=1}^{M-1} [y_{i+1} - y_i]^2$$

which reproduces Equation 4.10.

Case n=4: For this case, M=Int(25/4)=6 and the equation becomes

$$\sigma_A^2(4) = \frac{1}{2(M-1)} \sum_{i=1}^{M-1} \left[\frac{1}{n} \sum_{k=1}^{n} y_{in+k} - \frac{1}{n} \sum_{k=1}^{n} y_{(i-1)n+k} \right]^2$$

$$\sigma_A^2(4) = \frac{1}{10}\left\{ \left[\frac{y_{4+1} + \cdots + y_{4+4}}{4} - \frac{y_{0+1} + \cdots + y_{0+4}}{4} \right]^2 + \left[\frac{y_{8+1} + \cdots + y_{8+4}}{4} - \frac{y_{4+1} + \cdots + y_{4+4}}{4} \right]^2 + \cdots \right\}$$

Checking Figure 4.9, it can be seen the previous Equation reduces to the form

$$\sigma_A^2(4) = \frac{1}{10}\{ (z_2 - z_1)^2 + (z_3 - z_2)^2 + \cdots + (z_6 - z_5)^2 \}$$

as it should to match Equation 4.16a and the procedure in Example 4.6.

Topic 4.2.9: Modified Allan Variance and Time Variance

The Modified Allan Variance and Deviation, respectively denoted by MVAR and MDEV, provides discrimination between white and flicker noise. Figure 4.13 represents the scheme for MVAR for the case of n=4. MVAR(n) is determined as in the following several steps.

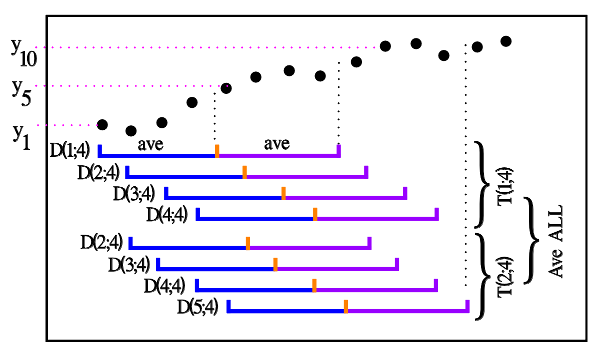


Figure 4.13: Example calculations for the Modified Allan Variance.

First, MVAR requires the differences of the y according to

$$D(k;n) = \frac{1}{n} \sum_{i=k}^{k+n-1} [y(i+n) - y(i)] \quad (4.24)$$

where n has the same meaning as previous. Figure 4.13 shows some of the sample blocks for D(k;n) starting at the first sample and using n=4 samples. Notice how the D(k;n) repeatedly use a subset of the block such as for k=1, 2, 3, 4.

Second, calculate the averages of n of the D(k;n) using Equations 4.18 and 4.17. The averages Y will use n sequential samples y, and the average of the D will use n sequential (in k) calculated values of the D. Figure 4.13 shows, for example, D(1;4) through D(4;4) forming a block of values. The average of the n values of D(k;n) is assigned to the integer j according to

$$T(j;n) = \frac{1}{n}\sum_{k=j}^{j+n-1} D(k;n) \tag{4.25}$$

This is very similar to the Overlapped Allan Deviation but limited to n of the D(k,n). The quantity T(j;n) is the average over n of the D(k;n) starting at the index k=j. For Figure 4.13, for example, we would have $T(1;4) = \frac{1}{4}\sum_{k=1}^{1+4-1} D(k;4)$. Substituting the expression for D(k;n) into Equation 4.25 and using Equations 4.18 and 4.17 provides

$$T(j;n) = \frac{1}{n^2}\sum_{k=j}^{j+n-1}\left[\sum_{i=k}^{k+n-1} y(i+n) - y(i)\right] \tag{4.26}$$

The T represents the average of the averages.

Finally compute the average of $[T(j;n)]^2$ for fixed n to arrive at MVAR(n).

$$MVAR(n) = \frac{1}{2n^4(N-3n+2)}\sum_{j=1}^{N-3n+2}\left[\sum_{k=j}^{j+n-1}\left[\sum_{i=k}^{k+n-1} y(i+n) - y(i)\right]\right]^2 \tag{4.27}$$

As a final result for the present topic, the Time Variance $TVAR(\tau)$ is defined as

$$TVAR(\tau) = \frac{\tau^2}{3} MVAR(n) \tag{4.28}$$

where, as before, $\tau = n\tau_o$.

Example 4.8: Show for n=1, Equation 4.27 reduces to Equation 4.10 namely

$$\sigma_A^2 = \frac{1}{2}\,\frac{1}{N-1}\sum_{i=1}^{N-1}(y_{i+1} - y_i)^2$$

Solution: Set n=1 in Equation 4.27 to obtain

$$MVAR(n) = \frac{1}{2(N-1)}\sum_{j=1}^{N-1}\left[\sum_{k=j}^{j}\left[\sum_{i=k}^{k} y(i+n) - y(i)\right]\right]^2$$

Given the limits on the two inner summations, they drop out of consideration and set $j = k = i$ to find

$$MVAR(n) = \frac{1}{2(N-1)}\sum_{j=1}^{N-1}[y(j+n) - y(j)]^2$$

Topic 4.2.10: M-Sample and 2-Sample Variance

As a side note, references generally comment that the Allan Variance relates to the so-called m-sample variance which is defined as an average

$$mSampVar = \sigma_{MS}^2 = \langle \sigma_j^2(m, T, \tau) \rangle_{ave\ j} \tag{4.29a}$$

where

$$\sigma_j^2(m, T, \tau) = \frac{1}{m-1} \left\{ \sum_{i=j}^{j+m-1} y_i^2 - \frac{1}{m} \left[\sum_{i=j}^{j+m-1} y_i \right]^2 \right\} \tag{4.29b}$$

The classic Allan Variance obtains by setting m=2 and $T = \tau$ so that the $\sigma_j^2(m, T, \tau)$ becomes

$$\sigma_j^2(2, T, \tau) = \sum_{i=j}^{j+1} y_i^2 - \frac{1}{2} \left[\sum_{i=j}^{j+1} y_i \right]^2 = y_j^2 + y_{j+1}^2 - \frac{1}{2} \left[y_j^2 + y_{j+1}^2 + 2 y_j y_{j+1} \right] = \frac{1}{2} \left(y_j - y_{j+1} \right)^2 \tag{4.29c}$$

Consequently, the m-Sample Variance can be seen to reproduce the Normal Allan Variance when m=2

$$mSampVar = \sigma_{MS}^2 = \langle \sigma_j^2(2, \tau, \tau) \rangle_{ave\ j} = \frac{1}{2(N-1)} \sum_{j=1}^{N-1} \left(y_j - y_{j+1} \right)^2 \tag{4.30}$$

Section 4.3: Transforming Distributions for ADEV Software

Calculating the Allan Deviation (as a function of averaging time τ) is best handled by computer especially for the case of hundreds of thousands of frequency samples. The references provide links to a variety of well-tested software [4.31-35]; some of this software can collect frequency samples by directly connecting to the frequency counter through USB or a RS232 Serial Port. Generally the software can calculate the normal, overlapped and modified varieties of the Allan Deviation. As seen in the previous sections, Log-Log plots provide insight into the types of noise affecting the frequency measurements. The references list both free-of-cost and professional software for calculating and plotting the various Allan Deviation types. First time readers can skip this section.

The present section briefly discusses the demo software [4.43] that accompanies this book and then explores the mathematics required for those readers wanting to develop their own Allan Deviation software based on the equations in the previous section. A number of programming languages only implement a random number generator for the uniform distribution and not one for the normal (i.e., Gaussian) distribution. The section discusses the method for transforming uniformly distributed random numbers to normally distributed ones. The process involves inverting the Error Function (ERF) for which algorithms can be found online, and the results must be related to the Gaussian distribution.

Topic 4.3.1: Example Software Description

The best way to understand the Allan Deviation is to work some simple examples by hand. After that, software can plot the same examples while providing real-world results from actual measurement systems. The reader would be best advised to download some of the well-tested software listed in the references [4.31-35]. This book includes access to a variety of software [4.43] primarily meant to be modified by the reader including to some extent the software meant to run the FE-5650A controller. The included free Allan Deviation software (named AllanDev) provides some simulation capability and it can read text files with frequency points separated by CR and LF characters. As will become evident, the present section primarily describes the mathematics required to convert numbers generated by a uniform probability distribution to numbers consistent with a Gaussian probability distribution. We start however, by showing the user interface and basic functionality for the included software.

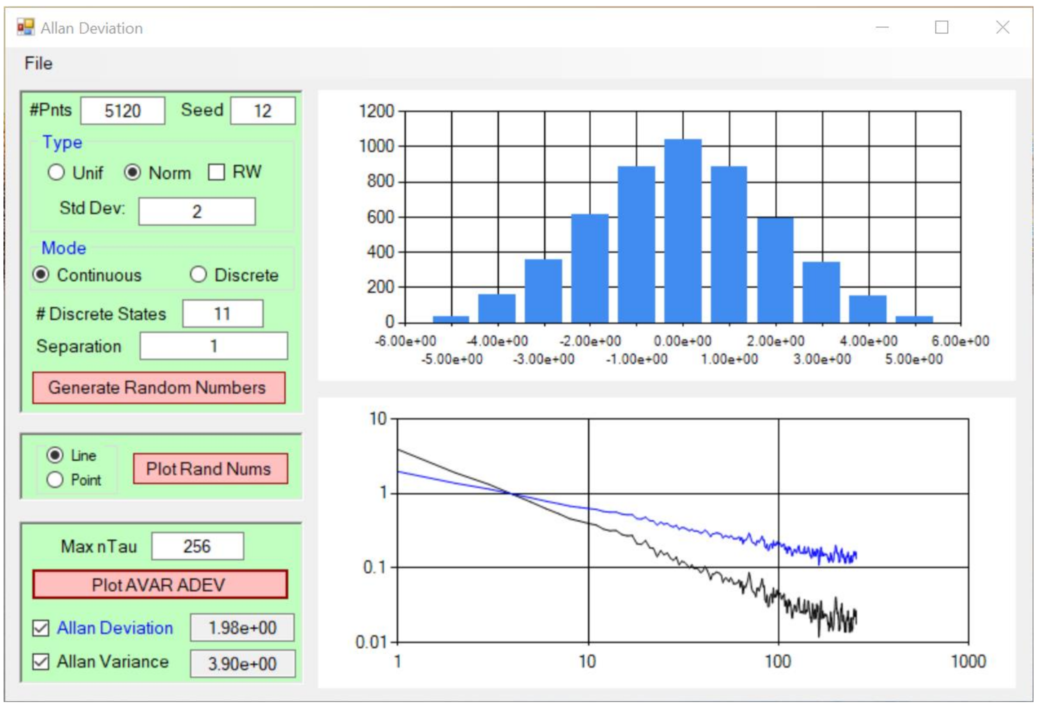


Figure 4.14: User Interface for Allan Deviation software

The Allan Deviation software allows one to observe the Allan Deviation and Variance for various statistical distributions and conditions. Figure 4.14 shows the Graphical User Interface (GUI) for the AllanDev software included with the book. The basic idea is to generate a random process using either a uniform or a Normal/Gaussian distribution. A checkbox allows the program to form a random walk from the selected distribution if desired. The top graph plots either the distribution (such as the Normal distributions shown) upon clicking the button labelled 'Generate Random Numbers', or it plots the actual sequence of random numbers, which looks like noise, upon clicking the button labelled 'Plot Rand Nums'. The bottom graph shows the time-dependent Allan Variance (black) and Allan Deviation (blue) upon clicking the 'Plot AVAR/ADEV' button. The software can read a text file (.txt) containing a custom set of numbers (i.e., entered by hand) using the 'File' menu item. It is also possible for the software to save the software-generated random numbers to a '.txt' file using the 'File' menu item. Keep in mind, the software is only for demonstration. Well tested software should be used for 'the stuff that matters.'

Next examine the three individual panels. The easiest is the bottom panel. The button simply causes the program to use the generated random numbers to calculate the time-dependent Allan Variance (black) and the Allan Deviation (blue) and overwrite any existing plot in the bottom chart. The code associated with the bottom panel is the work-horse for the Allan computations. The middle panel plots the sequence generated random numbers (or random walk) in the top chart. To use either of the bottom two panels, it's only necessary to generate the random numbers using the top panel. The bottom two panels do not depend on each other.

The top panel corresponds to the code for generating the random numbers. At present, the software can generate a uniform or Normal distribution of random numbers. The number of samples can be set in the text box labelled as '#Pnts'. It is possible to enter a seed for generating the random numbers; each set of numbers will be the same for the same seed. This makes it possible to repeat a set of numbers when 'something looks interesting'. The 'Std. Dev.', '# Discrete States', and 'Separation' require additional discussion (see below). Using the 'Mode' selection buttons, the distribution can generate either a continuous range of numbers or one divided up into discrete but adjacent bins.

Consider now the discrete mode. The textbox for the number of discrete states refers to the number of vertical states (i.e., y states) and the states appear as bins across the horizontal axis of the histogram. An even number of bins skips zero. An odd number includes zero. The 'separation' textbox refers to the (vertical) separation of the discrete states and hence also the horizontal separation of histogram bins.

Topic 4.3.2: Brief Review of Density and Cumulative Functions

We briefly review the probability density and the cumulative distribution functions, and their use to find probability. Consider the probability density function for the uniformly-distributed random variable X (notice the capital X), denoted by $f_X(x)$, which appears in Figure 4.15 as does the one for the normally-distributed random variable Y denoted by $f_Y(y)$. Of utmost importance, the density functions are normalized such that the area under the density is equal to one for the purposes of probability. Typically one would want to know, for example, the probability that the random variable X takes on the value x. But recall the probability density refers to the probability per unit interval so one must ask for the probability that X takes on a value x located in a small interval dx:

$$f_X(x)\, dx = \frac{prob}{interval} * interval = probability \qquad (4.31a)$$

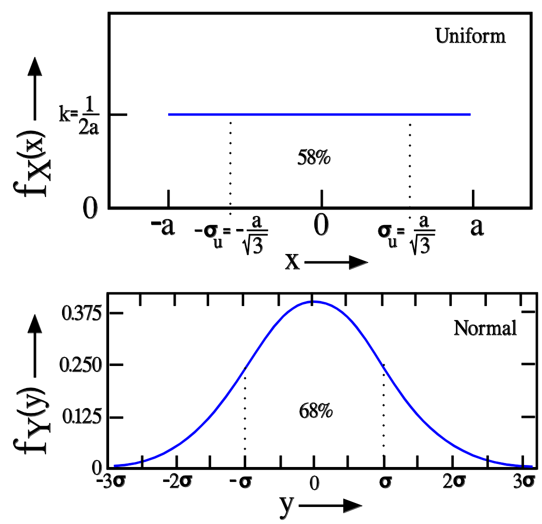


Figure 4.15: Top: Uniform distribution across the interval (-a,+a). The width of twice the standard deviation σ_u has approximately 58% of area under the curve. Bottom: Normal distribution for the interval $(-\infty, +\infty)$. The width of 2σ has 68% of the area. Note: the total area under each curve is arranged to be 1 since the total probability is 1.

Similar considerations provide the probability that Y takes on a value y located in a small interval dy is given by $f_Y(y)\, dy$. The probability of finding x in dx is the area of a small rectangle of width dx and height $f_X(x)$.

The probability that the value x is in any interval of width W will be the area under the curve $f_X(x)$ over the width W (similar statements for y). The probability that x is in the interval (c,d) is then given by

$$P(c < x < d) = \int_c^d dx' \, f_X(x') \tag{4.31b}$$

We have primary interest in the uniform and normal distributions. The density functions for the uniformly distributed random variable X, denoted by $f_X(x)$, and the normally distributed random variable Y, denoted by $f_Y(y)$, both appear in Figure 4.15 and have the following expression

$$f_X(x) = \begin{Bmatrix} \frac{1}{2a} & -a \leq x \leq a \\ 0 & else \end{Bmatrix} \tag{4.32a}$$

and

$$f_Y(y) = \frac{1}{\sqrt{2\pi}\sigma} Exp\left(-\frac{(y-\bar{y})^2}{2\sigma^2}\right) \qquad y \in (-\infty, +\infty) \tag{4.32b}$$

where σ is the standard deviation and $\bar{y}$ is the average which we set to zero $\bar{y} = 0$. The area under the density curve such as in Equation 4.31b is related to the cumulative probability function.

A few comments are in order regarding a *cumulative* distribution. The comments apply to any cumulative distribution but we use those for the uniform and normal distributions. The cumulative distribution monotonically increases within the bounds of 0 and 1 for the purposes of probability. The cumulative distribution function for the uniformly distributed random variable X is denoted by $F_X(x)$ (note the capital F). The probability of finding a value x in the range $(-a, x_o)$ is given by

$$F_X(x_o) = P(-a < x < x_o) = \int_{-a}^{x_o} dx' \, f_X(x') = area\ under\ f_X(x)\ over\ (-a, x_o) \tag{4.33a}$$

Inserting the uniform density function from Equation 4.32a into Equation 4.33a provides the cumulative uniform distribution function in the form

$$F_X(x_o) = P(-a < x < x_o) = \begin{Bmatrix} 0 & x_o < -a \\ \frac{x_o + a}{2a} & -a < x_o < a \\ 1 & x_o > a \end{Bmatrix} \tag{4.33b}$$

The cumulative distribution for the normally distributed random variable Y is denoted by $F_Y(y)$. The probability of finding a value of y in the range $(-\infty, y_o)$ is given by

$$F_Y(y_o) = P(-\infty < y < y_o) = \int_{-\infty}^{y_o} dy \, f_Y(y) = area\ under\ f_Y(y)\ over\ (-\infty, y_o) \tag{4.33c}$$

The Normal Cumulative Distribution function can be found in many books of tables – people don't enjoy integrating the Normal Density Function although computers seem ok with it. The normal cumulative distribution function is related to the Error Function ERF as will be used later. Since areas can be added/subtracted from each other, it is possible to use the cumulative distribution to calculate the probability, for example, of y in the range $(y_o < y < y_1)$

$$P(y_o < y < y_1) = P(y < y_1) - P(y < y_o) = F_Y(y_1) - F_Y(y_o) \tag{4.34}$$

As an example for the uniformly distributed random variable X in the top panel of Figure 4.15, the probability of finding a measured value in the interval $(-\sigma_u, \sigma_u)$ is given by the area under the curve in that interval as

$$P(-\sigma_u < x < \sigma_u) = \int_{-\sigma_u}^{\sigma_u} dx' \, f_X(x') = \frac{1}{2a} \, 2\sigma_u$$
$$= ht * wid = 0.58 \tag{4.35}$$

Similarly, the area under the Normal Distribution for Y over an interval provides the probability that the value y will be in that interval. For example, tables provide

$$P(-\sigma < y < \sigma) = \int_{-\sigma}^{\sigma} dy \, f_Y(y) = 0.68 \tag{4.36}$$

The numbers σ_u and σ refer to the standard deviation of the uniformly and normally distributed random variables respectively.

Topic 4.3.3: Convert from Uniform to Normal Distributions

The previous topic, by way of Figure 4.15, shows the uniform density for X appears as a horizontal straight line meaning X is equally likely to take-on a value anywhere in the range. The normal density for Y appears as a 'bumped curve' so that Y most likely will take-on values closer to the average at y=0.

The pertinent question for the section can be framed as follows. If a uniform random number routine generates a set of values { x }, then what mapping $x \rightarrow y$ will give the set { y } a normal distribution?

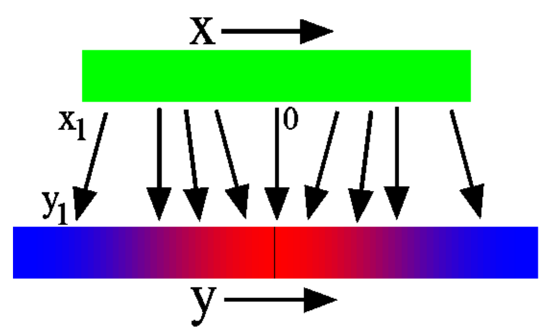


Figure 4.16: A cartoon showing how the uniformly distributed x values map to the normally distributed y values. The red region corresponds to the higher density (i.e., higher probability) and the blue region to the lower density.

A view of how the x values might be seen to correspond to the y values can be seen in Figure 4.16. The solid green coloring of the top x-range shows all values of x are equally-likely to be taken-on by X. The gradated color of the bottom range shows the normally distributed Y would be found to produce values more closely grouped near the average at the center than does X. The red near the center indicates the higher density (i.e., higher probability) and the blue indicates the lower density. Just to say the same but in another way, Figure 4.16 shows how the uniformly distributed values of x map into those for y(x) such that the density of points y increases for y near the center (i.e., the average) and decreases for y near the outside. Basically, the green-colored range for the uniformly distributed variable must be warped to form the normally distributed one. One way of specifying the map is to say the desired value of y corresponding to the value of x is found when the area under the normal curve over $(-\infty, y)$ is equal to the area under the uniform curve over (-a,x). For example, x=0 and y=0 give the same area of ½ and so x=0 corresponds to y(x)=0. As the primary example, if x_1 produces the probability $P_X(x < x_1) = 0.05$ then the value y_1 is the one giving $P_Y(y < y_1) = 0.05$. This last equation would need to be inverted to solve for y_1. As another example, although not so obvious and not so important, the exactly vertical arrows in the figure correspond to very roughly $x = \pm 1.1a$ which very roughly maps to $y = \pm 0.9\sigma$.

Consequently, the way to describe the transformation of the uniformly distributed values x to normally distributed values y is to find the value y for the given value x that makes the following true

$$F_Y(y) = F_X(x) \tag{4.37}$$

where $F_X(x) = \frac{x}{2a} + \frac{1}{2}$ by integrating $f_X(x) = \frac{1}{2a}$ over the interval (-a,x). So we need to find y in

$$F_Y(y) = \frac{x}{2a} + \frac{1}{2} \tag{4.38}$$

where x corresponds to the number from a uniform random number generator over the interval (-a,a). Of course that means we need the inverse function of $F_Y(y)$. Let's symbolize the inverse by $InvF_Y$. The inverse has the property of $InvF_Y(\ F_Y(y)\ ) = y$. Then applying the inverse to both sides of Equation 4.38, we find

$$y = InvF_Y\left(\frac{x}{2a} + \frac{1}{2}\right) \tag{4.39}$$

The inverse function of the normal distribution can be either a look-up table or an algorithm. As it turns out, the algorithm is short and efficient and preferred but it does require the cumulative normal distribution to be written in terms of the Error Function ERF [4.40].

Topic 4.3.4: Relation between ERF and Probability

As previously discussed, some programming languages offer only uniformly-distributed random number generators and therefore, must be augmented with further code to generate Normally-distributed random numbers. It is possible to algorithmically integrate the Normal Density or to use a look-up table to obtain the inversion in Equation 4.39. However, the internet offers a nice algorithm [4.41-42] for inverting the Error Function. So it would be advisable to relate the ERF to probability and then, in the next topic, to the Cumulative Normal Distribution.

First relate ERF to probability starting with the definition

$$ERF(z) = \int_0^z \frac{2}{\sqrt{\pi}} e^{-\tilde{z}^2} d\tilde{z} \tag{4.40}$$

where z > 0 is a real number and the twiddle ~ simply denotes the integration variable. Now change variables to make the previous look more like a Normal Distribution. Let

$$y = \sqrt{2}\sigma\ z \tag{4.41}$$

where we assume for now that $y \geq 0$. The ERF becomes

$$ERF(z) = 2 \int_0^{y=\sqrt{2}\sigma z} \frac{1}{\sqrt{2\pi}\sigma} Exp\left(-\frac{\tilde{y}^2}{2\sigma^2}\right) d\tilde{y} \tag{4.42}$$

Comparing Equations 4.42 and 4.32b shows this last equation is an integral over the Normal Density Function.

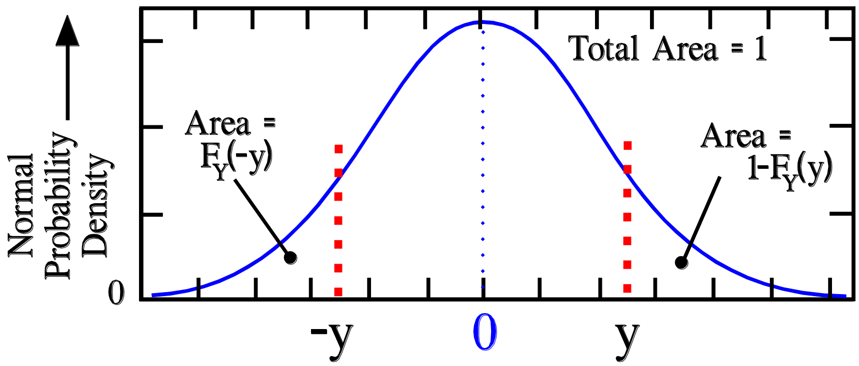


Figure 4.17: The Normal Probability Density $f_Y(y)$ is symmetric about the average value which is 0 in this case. The density function is normalized in such a manner that the area under the curve is one.

The previous equation can be written in terms of the normal cumulative distribution function F_Y as

$$\begin{aligned} ERF(z) &= 2(\ F_Y(y) - F_Y(0)\ ) \\ &= (\ F_Y(y) - F_Y(0)\ ) + (\ F_Y(y) - F_Y(0)\ ) \\ &= (\ F_Y(y) - F_Y(0)\ ) + (\ F_Y(0) - F_Y(-y)\ ) = F_Y(y) - F_Y(-y) \end{aligned} \tag{4.43}$$

where the second term in the third line comes from adding areas under the Normal Density Function (Figure 4.17) and the fact that the Normal Density Function is symmetric about y=0. So then the connection with probability can be deduced although it will not be required for further consideration

$$ERF(z) = Prob_Y(-\sqrt{2}\,\sigma\, z\ < Y < \sqrt{2}\,\sigma\, z) \tag{4.44}$$

Topic 4.3.5: Relation between ERF and Normal Distribution

Next the Cumulative Normal Distribution needs to be written in terms of the ERF so that the inverse of the Normal Distribution can be developed in the next topic. Starting with Equation 4.43 in the form

$$F_Y(y) = F_Y(-y) + ERF(z) \tag{4.45}$$

where $y \geq 0$. Figure 4.17 and subtracting/comparing areas shows $F_Y(-y) = 1 - F_Y(y)$ and provides

$$F_Y(y) = \tfrac{1}{2}\big(1 + ERF(z)\big) \tag{4.46a}$$

where Equation 4.41 is repeated here to show

$$y = \sqrt{2}\sigma\, z \qquad y,\ z \geq 0 \tag{4.46b}$$

Next look at the case of ‘ $-y$’. Returning to Equation 4.43 in the form

$$F_Y(-y) = F_Y(y) - ERF(\,|z|\,) \tag{4.47}$$

The absolute value was added to emphasize z>0 for later when the constraint on $y \geq 0$ is relaxed. Based on Figure 4.17 and by comparing areas, $F_Y(y) = 1 - F_Y(-y)$ so the previous equation can be rewritten as

$$F_Y(-y) = \tfrac{1}{2}\big(1 - ERF(\,|z|\,)\big) \tag{4.48}$$

Combine Equations 4.46a and 4.48 and allow y to be positive or negative, to find

$$F_Y(y) = \tfrac{1}{2}(\ 1 \pm ERF(\,|z|\,)\ ) \quad \text{Use '+' for } y \geq 0 \text{ and use '–' for } y < 0 \tag{4.49a}$$

The relation between y and z can be substituted (Equation 4.46b) to find

$$F_Y(y) = \tfrac{1}{2}\Big(\ 1 \pm ERF\left(\tfrac{|y|}{\sqrt{2}\,\sigma}\right)\Big) \quad \text{Use '+' for } y \geq 0 \text{ and use '–' for } y < 0 \tag{4.49b}$$

Topic 4.3.6: The Inversion and Solution

Finally the results of the last several topics can be combined to find a value of y (from Equation 4.38) such that

$$F_Y(y) = \tfrac{x}{2a} + \tfrac{1}{2} \tag{4.50}$$

References [4.40-41/1.9.6, 1.9.7] provide an algorithm to invert the ERF and the previous topic provides the necessary relation between the Normal Cumulative Distribution $F_Y(y)$ and the ERF. Now Equations 4.49 with 4.50 provides

$$ERF(\ |z|\ ) = \pm\tfrac{x}{a} \qquad \text{Use '+' for } x \geq 0 \text{ and use '–' for } x < 0 \tag{4.51}$$

Finally, applying the inverse function InvERF and using $y = \sqrt{2}\sigma\, z$ provides

$$y = \pm\sqrt{2}\sigma\ InvERF\left(\ \tfrac{[x]}{a}\ \right) \qquad \text{Use '+' for } x \geq 0 \text{ and use '–' for } x < 0 \tag{4.52}$$

Topic 4.3.7: The Code

The C code for inverting the ERF can be found in Reference [4.41]. The Visual Basic code appears below and it can easily be changed to other languages as desired.

```
' algorithm from https://stackoverflow.com/questions/27229371/inverse-error-function-in-c
' meaning of sqrtf at https://en.cppreference.com/w/c/numeric/math/sqrt means float argument
' meaning of logf at https://en.cppreference.com/w/c/numeric/math/log means float argument

Friend Function ErfInv(ByVal x As Double) As Double
        Dim tt1, tt2, lnx, sgn As Double
        sgn = If((x < 0), -1.0F, 1.0F)
        x = (1 - x) * (1 + x)
        lnx = Math.Log(x) 'was logf(x)
        tt1 = 2 / (3.1415 * 0.147) + 0.5F * lnx 'math.pi
        tt2 = 1 / (0.147) * lnx
        Return (sgn * Math.Sqrt(-tt1 + Math.Sqrt(tt1 * tt1 - tt2))) 'was sqrtf
    End Function
```

Section 4.4: References

[4.1] Accuracy for various oscillators appear in Ch. 6 in
S. Bregni, "Synchronization of Digital Telecommunications Networks," 1st Edition, Wiley (2002)
Available from Amazon.com or download free of charge at
https://www.academia.edu/34111764/Bdqxa.Synchronization.of.Digital.Telecommunications.Networks.by.Stefano.Bregni

[4.2] Nice guide on accuracy for various oscillators
https://www.meinbergglobal.com/english/specs/gpsopt.htm

[4.3] AD9830 data sheet:
https://www.analog.com/media/en/technical-documentation/data-sheets/AD9830.pdf

[4.4] D. W. Allan, "Statistics of Atomic Frequency Standards," Proceedings of the IEEE, 54, 2 (1966)
https://tf.nist.gov/general/pdf/7.pdf

[4.5] W.J.Riley, "Handbook of Frequency Stability Analysis," NIST Special Publication 1065 (2008)
Available from Amazon.com or download free of charge at
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication1065.pdf

[4.6] IEEE Standards Committee, "IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology – Random Instabilities" IEEE Std 1139-1999 (1999)
https://ieeexplore.ieee.org/document/807679

[4.7] A. Makdissi, ALAMATH documentation, https://www.alamath.com/alavar/

[4.8] Wikipedia "Allan Variance" Shows M-Sample Variance, background of Allan Variance and need for convergence
https://en.wikipedia.org/wiki/Allan_variance

[4.9] Wikipedia "Modified Allan Variance"
https://en.wikipedia.org/wiki/Modified_Allan_variance

[4.10] D.W.Allan, J.A.Barnes "A Modified 'Allan Variance' with Increased Oscillator Characterization Ability", Proc. 35th Ann. Freq. Control Symposium, USAERADCOM, Ft. Monmouth, NJ 07703 (1981)
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.207.2479&rep=rep1&type=pdf

[4.11] D.A.Howe, D.W.Allan, J.A.Barnes, "Properties of Signal Sources and Measurement Methods," Proceedings of the 35th Annual Symposium on Frequency Control (1981). Available as part of the NIST Technical Note 1337 edited by D.B.Sullivan, D.W. Allan, D.A.Howe, F.L.Walls. (gives sample calculation for AVAR and ADEV)
https://nvlpubs.nist.gov/nistpubs/Legacy/TN/nbstechnicalnote1337.pdf
Another one but mislabeled online: https://tf.nist.gov/general/pdf/868.pdf

[4.12] Phidgets Inc, "Allan Deviation Primer"
https://www.phidgets.com/docs/Allan_Deviation_Primer

[4.13] Jerath, "Noise Modeling of Sensors: The Allan Variance Method" Power Point Slide Presentation: https://eecs.wsu.edu/~taylorm/16_483/Jerath.pptx Nice introduction to Allan Variance and slope methods.

[4.14] Everything RF, "What is aging of a crystal oscillator?"
https://www.everythingrf.com/community/what-is-aging-of-a-crystal-oscillator

[4.15] Wikipedia "Crystal Oscillator"
https://en.wikipedia.org/wiki/Crystal_oscillator

[4.16] C.D.Motchenbacher, J.A.Connelly "Low-Noise Electronic System Design," John Wiley & Sons, Inc., New York (1993)

[4.17] R. Keim, "What is electronic noise and where does it come from?" All About Circuits website (2018)
https://www.allaboutcircuits.com/technical-articles/electrical-noise-what-causes-noise-in-electrical-circuits/

[4.18] R. Cerda, "Sources of Phase Noise and Jitter in Oscillators"
https://www.crystek.com/documents/appnotes/SourcesOfPhaseNoiseAndJitterInOscillators.pdf
Nice explanation

[4.19] J.J.Gagnepain, G. Theobald, J.Uebersfeld "Analysis of 1/f2 and 1/f Frequency Noises in Quartz Resonators" J. De Physique, Colloque C8, supplement 12, 1981. Copy the following link and paste in browser
https://hal.archives-ouvertes.fr/jpa-00221719/document

[4.20] Wikipedia "Flicker Noise"
https://en.wikipedia.org/wiki/Flicker_noise

[4.21] Allan Deviation for crystals
https://www.rakon.com/products/technical-resources/tech-docs/download/file?fid=54.317

[4.22] allan deviation
http://home.engineering.iastate.edu/~shermanp/AERE432/lectures/Rate%20Gyros/Allan%20variance.pdf

[4.23] D.A. Howe, K.J. Lainson, "Simulation study using a new type of sample variance" DTIC report (Dec 1995)
https://apps.dtic.mil/dtic/tr/fulltext/u2/a509016.pdf

[4.24] D.A. Howe, D.W. Allan, and J.A. Barnes, "Causes of Noise Properties in a Signal Source"
https://tf.nist.gov/phase/Properties/twelve.htm
From "PROPERTIES OF OSCILLATOR SIGNALS AND MEASUREMENT METHODS"
See Table of Contents: https://tf.nist.gov/phase/Properties/toc.htm
And also https://tf.nist.gov/phase/Properties/main.htm

[4.25] Example Allan plots
http://www.leapsecond.com/pages/adev-fm/
http://www.leapsecond.com/pages/adev-avg/

[4.26] Random Process
http://web.stanford.edu/class/archive/ee/ee278/ee278.1152/lect06-2.pdf

[4.27] Lecture on Ergodic and Stationary Processes
https://www.youtube.com/watch?v=k6y2kzayV6A&#t=1433

[4.28] Robert Nau, "Random Walk Model" An alternate example in terms of currency
https://people.duke.edu/~rnau/411rand.htm

[4.29] Random walk not stationary
https://stats.stackexchange.com/questions/246357/why-is-a-random-walk-not-a-stationary-process

[4.30] measure variance for three to find one
https://www.dsprelated.com/thread/1950/measuring-allan-variance-of-atomic-clock

[4.31] A. Makdissi, ALAMATH.com "ALAVAR" : Allan Deviation Plots; free and excellent but not integrated with the noise generation software.
https://www.alamath.com/alavar/ Specifically: http://www.alamath.com/progs/Alavar52_setup.exe

[4.32] A. Makdissi, ALAMATH.com "ALANOISE" software; free and excellent but not integrated with the Allan Variance software.
https://www.alamath.com/alavar/ Specifically: http://www.alamath.com/progs/Alanoise3_setup.exe

[4.33] EZL Data Plotting has professional selectable plotting features including nth order regression, FFT, and ADEV(n), various types of noise generation (and more). Be aware the software expects the input to be a time series not frequency and so must use the menu item to convert between frequency and time. The software is fully integrated. Excellent. 30 day free trial prior to $79 single lifetime payment.
http://www.ezlsoftware.com

[4.34] M. Sims "Lady Heather". Free and the software typically promoted by the TimeNuts website.
http://www.ke5fx.com/heather/readme.htm

[4.35] W. Riley "Stable32: Software for Frequency Stability Analysis". Not clear if it is still available.
https://ieee-uffc.org/frequency-control/frequency-control-software/

[4.36] W.D.Stanley, "Investigation of Allan Variance for Determining Noise Spectral Forms With Application to Microwave Radiometry" Nasa Contractor Report 194985 (Nov. 1994)
https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19950009313.pdf
It has some old but useful, short, Quick Basic programs that can be converted to modern languages.

[4.37] Lolo406 "C code for Allan Variance" (2012)
https://www.scribd.com/document/112404612/C-Code-for-Allan-Variance
Might be of some use.

[4.38] Some older NIST plotters might be of some use.
https://www.itl.nist.gov/div898/software/dataplot/refman1/ch2/allan_var.pdf
https://www.itl.nist.gov/div898/software/dataplot/
https://www.itl.nist.gov/div898/software/dataplot/ftp/win_vista/homepage.htm

[4.39] J. Rutman "Characterization of phase and frequency instabilities in precision frequency sources: Fifteen years of progress," Proc. IEEE, 66, 9 p.1048 (1978)
http://www.photonics.umbc.edu/Menyuk/Phase-Noise/rutman_ProcIEEE_090178.pdf

[4.40] Error Function
https://en.wikipedia.org/wiki/Error_function

[4.41] Nimig18 "Inverse Error Function in C" Stackoverflow.com (Dec 2014)
https://stackoverflow.com/questions/27229371/inverse-error-function-in-c

[4.42] S. Winitzki, "A handy approximation for the error function and its inverse" (2008)
https://www.academia.edu/9730974/A_handy_approximation_for_the_error_function_and_its_inverse

[4.43] Source code, listings, downloads, flash drives, preprogrammed microcontroller:
(A) Chapter 10 and Appendix 6 provide the source code for the C/C++ program
(B) Appendices 7-9 provide the PC software
(C) Check EBay.com in connection with the title of this book: software and preprogrammed MEGA328P
(D) check www.AngstromLogic.com for current offerings, or write to Sales@AngstromLogic.com
Keep in mind that Angstrom Logic, LLC is not setup/able to answer questions on the book or software.
(E) Check Amazon in connection with the title of this book.
(F) Note: The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the words). The contents are copyrighted with all rights reserved (also see front copyright page) except as noted. The software/firmware is scanned for viruses before posting. Download only from secure sources and rescan for viruses before using.

Chapter 5: FE-5650A Initial Setup and Tests

The surplus/used/preowned Rubidium Frequency Standard (RFS) FE-5650A Opt. 58 should be tested for basic functions using the front connector before modifying the circuits to increase the bandwidth and drive capability [5.1-5]. We tested four units. All four of them have an AMP connector on the front panel that offers a 'test signal' (typical default of 8.388608MHz), lock indicator, and a 1pps pulse source (roughly 850nSec pulses into 50 Ohms). Other versions of the FEI units replace the AMP connector with DB9 and SMA connectors. The 5650A has an RS232 interface internal to the unit but it's not externalized through the AMP connector. Apparently some versions do bring the serial lines out to a DB9 connector but perhaps not with standard pin assignments. The serial port can be used to set the output frequency or to interrogate the reference frequency and the Fcode (HEX code) both of which are used to set the default signals.

Figure 5.1: The FE-5650A Option 58 as viewed from the connector side. The unit is designed and manufactured by Frequency Electronics Inc. FEI.

Section 5.1: Temperature

The rubidium standard requires an elevated temperature to properly operate. The temperature plots in the FEI documents [5.6] suggest normal operating temperatures in the range 40-50C. The FE-5650A Opt. 58 has a rather compact but challenging design for temperature control, testing and for that matter, normal operation. The framework consists of two relatively thick (0.2"/5.1mm) aluminum plates with 4-40 tapped holes that meet at a right angle as shown in Figure 5.2. A heat sink needs to screw into these tapped holes. Some initial tests were made with this mounting side facing upward without a heat sink. A k-type thermocouple was temporarily mounted to the physics module (metal enclosure in Figure 5.2) using Kapton/Polyimide tape (usable to 200C/400F) to verify the module remained below the 60-65C maximum upper limit. The temperature generally rose to about 54C and remained below 55-57C without a heat sink. On two units, the temperature unexpectedly rose to 60C and produced smoke and stench. The tan-colored tantalum capacitor (22uF, 35V) on the left side of the upper most PCB (C425 on the DDS board) in Figure 5.2 had internally shorted in the two units within a few minutes of applying power; the short is a typical failure mode for tantalum capacitors. The same capacitor in two other units appears to have been previously changed. Fortunately, except for the physics module and the pre-programmed microcontroller, the parts are easy to find and replace. By the way, most hookup wire is rated for 105C which should be sufficient for the present project, but for an extra margin of safety, consider using silicone coated hook up wire which handles temperatures to 200C.

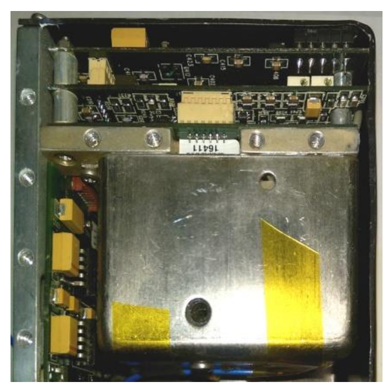


Figure 5.2: Mounting side of the FE-5650A Opt. 58 Rb standard. The black outer shell was removed. Kapton tape can be seen as the yellow strips on the physics package.

Section 5.2: Power Supply

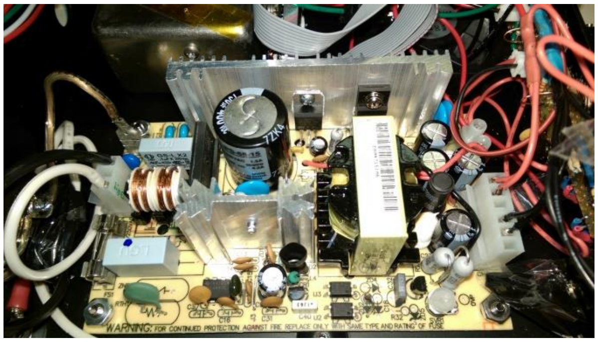


Figure 5.3: The 15V power supply. The regulator at the center of the heat sink is the L7805CV between the large electrolytic capacitor and the transformer. The leads for the associated capacitors pass through unused holes on the board.

The FE5650A needs a well-regulated 15VDC power supply capable of *at least* 2Amps of current for the electronics and heater and a separately regulated 5VDC source capable of at least 200mA for the internal microcontroller and TTL level logic. Separate tests indicated one FE5650A required approximately 1.6 Amps from the 15V supply while the physics module was heating at startup and then, as the module temperature approached 50-53C, the current dropped to approximately 0.60 Amps. The FE-5650A uses an LM2941 regulator [5.2] configured for 14V output - the regulator input must exceed the output by 0.5V. A Schottky diode with a 0.2V drop connects between the 15V input and the regulator and as a result, the unit requires the power supply voltage be larger than 14.7V (=14.0+0.5+0.2). For the 15V supply, it is possible to use an old laptop computer power supply (i.e., the brick) but the cable might need to be shortened to eliminate the voltage drop along the cable (and connector) that might otherwise allow the voltage at the connector to decrease to the 14.7 volt minimum. Be aware the unit can only withstand 18VDC and to help manage the temperature, the input voltage should be kept as close as possible to 15V.

A number of power supplies can be mounted inside an enclosure with the FE-5650A such as the Mean Well PS-65-15 Open-Frame Switching AC-to-DC Power Supply available from Jameco.com or sometimes from Amazon.com or EBay.com for $13 at the time of this writing (Figure 5.3). This Mean Well unit produces 15V at up to 5Amps. Another usable, more compact unit is the Mean Well EPS-65-15 that provides 15V at 5Amps. The Mean Well PS-45-15 can provide 15V and at 3A although we did not test it with the FEI units. With these mentioned power supplies, an additional 5V regulator will need to be added as discussed below; however various manufacturers offer power supplies with the dual output of 15V and 5V. The FE5650A specifications/data sheet states that the unit requires 5V at 100mA while several blogs indicate 200-300mA. The Mean Well PS-65-15 (Figure 5.3) has a heat sink with an extra hole which allows a L7805CV voltage (5V, 1Amp) regulator to be mounted. Thermal grease should be added between the heat sink and the L7805CV. The heat sink does become quite warm from the normal operation of the 15V supply but not enough to problem the L7805CV. We measured a temperature of 48C on the heat sink at the regulator which places the regulator junction temperature close to 75C, well below the 150C maximum. As a comment, the references indicate that more recent FE5650A units have built-in 5v regulators and need only the 15V.

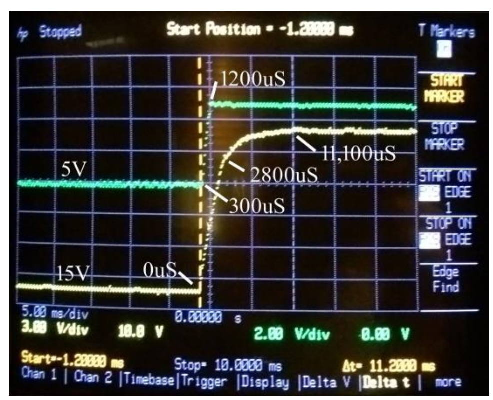


Figure 5.4: Turn-on and settling times for the 15V and 5V power supplies.

Some of the online groups [5.4] state that slowly ramping up the voltage to the FE5650A can/will kill the unit. In particular, refrain from using *variable* 15V power supplies since users have a tendency to slowly ramp up the voltage and also because some are not well regulated. According to the blog, the slow ramp causes the microcontroller to enter a quasioperational state that then causes it to overwrite/corrupt the non-volatile memory. Apparently the affected units stopped working and the internal microcontroller returned garbled characters in response to the status command. This obvious bad behavior should be distinguished from the case found for the surplus/used FE5650A tested here, whereby the units appear to return encrypted responses to the status command but still correctly set the frequency. We measured the response time (Figure 5.4) of our combined 15V/5V supply (Figure 5.5) with the FE5650A as the output load. The 5V regulator output establishes within 1.2 mSec while the 15V one becomes fully stable within 11mSec. It should be pointed out that the Mean Well supply appears to have a built-in delay of 1 or 2 seconds from the time of applying the 110V mains power to the initiation of the 15V output, which also feeds the added 5V regulator.

The power supply schematic appears in Figure 5.5. The Line (L, hot) connects to the on/off switch which then connects to a fuse on the PS-65-15 PCB. The fuse is suitable for the 5amps which includes the 5V output. Keep in mind that it does not protect against improperly wired circuits that draw less than 5Amps such as might happen with the 5V regulator. The green, high-efficiency LED indicates on/off for the unit in addition to indicating that both the 15V and the 5V supplies produce voltage. The plug is wired with 18Ga stranded lamp cord although regular 18Ga stranded silicone hookup wire would be better. The heat sink and case are connected to ground. It’s always a good idea to check the voltages prior to connecting the power supply to the FE5650A.

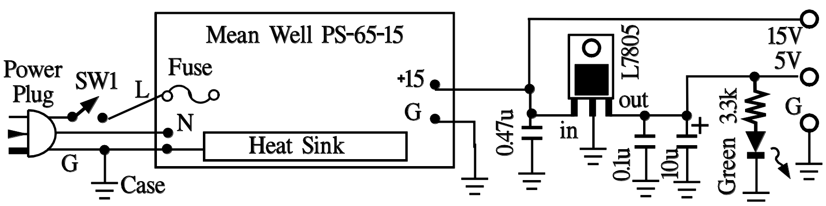


Figure 5.5: The power supply for the FE-5650A. The 0.47uF and 0.1uF are ceramic capacitors 50WVDC. The 10uF is an electrolytic at 35WVDC – note the polarity. The L7805CV is attached to the Heat Sink.

We found it most convenient to mount the various components in the final enclosure prior to testing the FE5650A. At the least, the power supply could be secured and isolated from metal and fingers etc.

Section 5.3: AMP Connector

Now it’s time to wire the AMP connector mounted on the front panel shown in Figure 5.1. Some FE5650A units might be sold with a DB9 connector rather than the AMP connector and they might have an SMA connector for the signal output. Some units have the serial port externalized through the DB9 connector. In the unit modified here, the all-important serial port was only internally available. The unit appeared to be removed from working equipment and used the AMP connector. The AMP connector has the manufacturer product number AMP 104655-3. The mate consists of the connector AMP 5-104893-2 and requires two of the Terminate Covers AMP 104891-2. At the time of this writing, the mating connector and covers can be purchased through Digikey.com using the stock numbers A115267-ND for the connector and A124419-ND for the covers. The connectors require two ribbon cables with 10 wires per cable; the wire spacing should be 0.050 inches (1.27mm). Digikey.com also carries a mating AMP connector suitable for soldering to a printed circuit board (PCB). This PCB version has the manufacture number AMP 5-104652-2 and digikey.com stock number A33547-ND.

The block diagram of Figure 5.6 shows the AMP connector installed on the FE5650A Opt. 58 unit as viewed from outside of the FE-5650A looking toward the connector. The pins are labelled according to their function. Obviously, the view in the figure is the same as the view of the mating connector looking at its back side where the wires attach. Before proceeding, make sure the mating AMP connector properly seats the FE5650A connector. Notice pin #1 corresponds to the largest cutout (i.e., keyway) on the installed connector and the largest ridge on the mate. The next section discusses the connections and associated circuits (Figure 5.7). The signals (lock, test, and 1pps) from these pins can be externalized to the outside world along with the sine and square wave signals from within the unit. For a compact design of the wire harness, consider soldering together the +15V pins at the connector and similarly the ground connections. The ground for 5V along with Bits 1 and 2 should be connected to the ground for 15V. If the purpose of implementing the connector is to test multiple FE5650A units, then it's worth considering the possibility of soldering the components shown in Figure 5.7 right at the connector. MOST IMPORTANT: If the AMP connector is used for the power, then all of the pins for '+15V IN' and all of the pins for GND must be used since for fewer pins, since as stated in the references, the PCB traces can melt due to the high current draw for the 15V. The references further suggest the +15V can be applied "directly to the input protection diode and the ground directly to one of the voltage regulator ground pins."

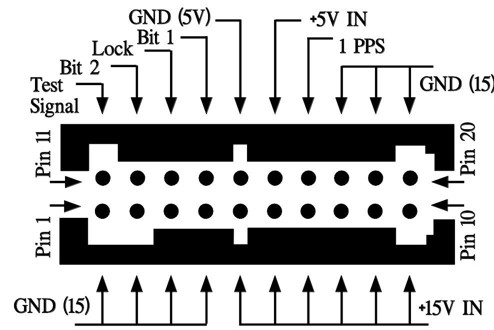


Figure 5.6: Pin functions for the AMP connector installed in the FE-5650A Opt. 58 looking at the connector from outside the FE-5650A. Notice the widest keyway marks pin 1.

Generally, the mating connector is designed for ribbon cables to carry all of the signals and power. The ribbon cables are prepared and attached as follows. Cut the ribbon cable flat so that the cut edge is perpendicular to the edges of the cable. Although not necessary, the side of the ribbon cable with red markings (if any) should be used to indicate pin #1. The proper connector has v-slots in the metal receptors; the v-slots are sharp and cut through the plastic wire insulation and thereby make contact with the wire cores. Line up the cable so that each wire aligns with the receptor on the connector. Somewhat press the cable into the receptors enough to see the sharp points pass into the cable and see that they sit between the wires. Lightly press a cover onto the cable with the cover prongs passing into holes at the edge of the connector. Do the same on the other side for a cable and cover. The plastic prongs on the covers self-align with the connector holes. The light pressure on the cover prevents the cable from slipping. The covers should interlock with each other when they are forced together. If the covers fall off or they are not secure, then they haven't been pressed enough. Use an Ohm meter to ensure each connector pin connects to exactly one wire.

Section 5.4: Pin Functions and Circuits

Next consider the various functions associated with the pins in the AMP connector on the FE5650A Opt. 58 unit. The greatest interest centers on the use of the FE5650A to generate signals with very stable accurate frequencies. The test pin (pin 11) in Figures 5.6 and 5.7 provides a signal with the a frequency set by sending HEX code (Fcode) to the AD9830A DDS in the unit. The default frequency from all of the tested units was 8.388608 MHz which, when divided by 2^{23} by the ripple counters, provided the 1 pulse per second (PPS) with roughly 840 nSec pulse width. The 'test signal' pin requires a pullup resistor of 2.2k to develop a voltage-based signal (since the pin is open collector). The resulting test signal is a sinewave of approximately 1.5Vpp centered at approximate 2.5 volts; however, the test signal at connector pin 11 doesn't have much drive capability. Apparently most unmodified units of this type produce the default frequency of 8.388608MHz and, right out of the box, the FE5650A can be used to check the calibration of a frequency counter, for example (Appendix 2). The RFS can be used to calibrate lab equipment but that equipment often requires 10MHz or even 1kHz for calibration; therefore, the FE5650A frequency needs to be programmable and have sufficiently wide output bandwidth. The modifications to extend the bandwidth will be presented in the Chapter 7. Even though the test signal on pin 11 doesn't have much drive capability, it is still worth bringing the test signal to the front panel of a final enclosure.

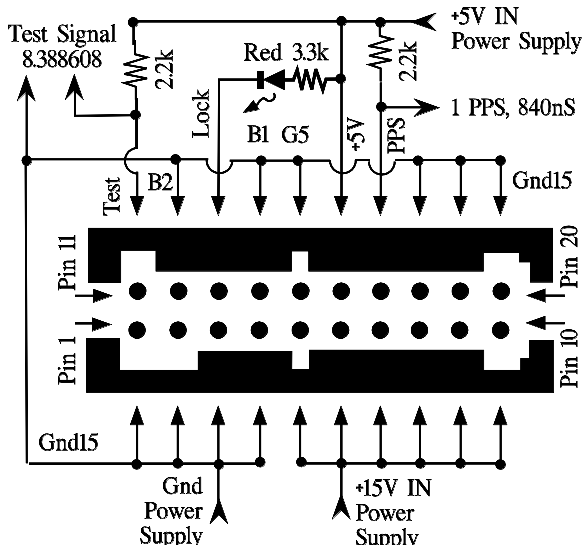


Figure 5.7: Pin connections and circuits for extracting useful signals.

The 'Lock Alarm' signal on pin 13 (open collector) sinks current through the high efficiency LED until the 'Lock' condition occurs when the frequency of the internal VCXO matches and locks to the rubidium-87 hyperfine transition frequency. When this occurs, the voltage on pin 13 will transition to +5V (i.e., the open collector discontinues sinking current and returns to the high impedance state) and the LED will extinguish. At this point, the internal AD9830A DDS receives an accurate reference from the physics module and therefore accurately produces the frequency set by the PIC microcontroller. Lock pin 13 is 'open collector' and so the LED connected to the pin should use a current limiting resistor, which is 3.3k in this case. The purposes for Bits 1 and 2 (pins 14 and 12, respectively) are not detailed online but could be discovered by tracing the lines back to the regulator board in the unit; however, both bit pins should be grounded for proper operation. If employed, the red Lock LED should be brought to the front panel to indicate when the unit is ready to use.

Pin 17 carries the '1 pulse per second' (1 PPS) output and, if desired, can be brought out to a BNC connector on the front panel of the final enclosure. The 1PPS signal on pin 17 is accurate only when the AD9830A DDS produces 8.388608MHz since the DDS drives twenty three cascaded binary dividers which then gives $8388608/2^{23} = 1$ Hz. The pulse width was measured to be 840nSec. It should be pointed out that without the test-signal pullup resistor, the pin presents a weak signal below 10mV with a frequency below 1 MHz.

As already stated and as shown in Figure 5.7, all of the grounds (pin #s 1-4, 15, 18-20) should be connected together and this common ground connects to the power supply ground G (Figure 5.5). Also note, Bits 1 and 2 on pins 14 and 12, respectively, connect to this common ground. If using the AMP connector on the front of the unit then be sure to use all pins designated as ground otherwise the associated PCB traces could melt due to the high current on the 15V supply. Likewise, all of the '+15V IN' pins should be connected together and then to the +15v supply. Similar to the grounds, using fewer pins makes it likely for PCB traces to fail. Pin 16 for '5V IN' connects to the 5V power output shown in Figure 5.5.

Section 5.5: Enclosure

The power supply and FE5650A Opt. 58 were mounted into a metal case although it's really not necessary for the initial tests – just made it a little easier. Figure 5.8 shows the start of a test as the unit begins to warm. The green LED indicates power and the red LED indicates the FE-5650A has not locked the rubidium 87 microwave frequency yet. The temperature meter shows the physics package at 21.3C and the ambient at 18.6C.

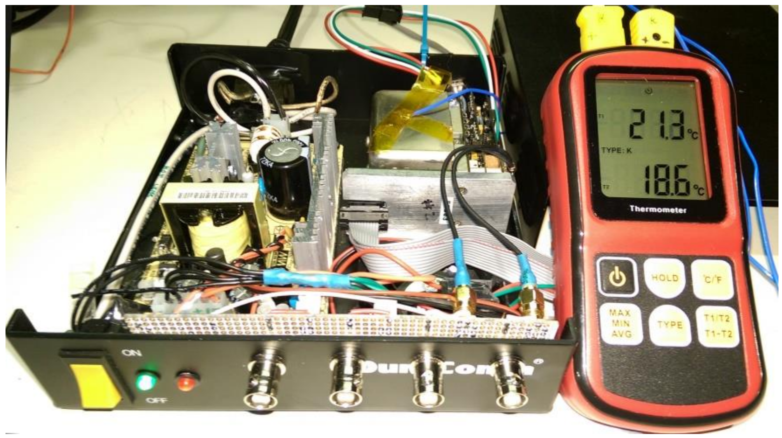


Figure 5.8: The power supply and FE-5650A mounted in a metal enclosure. The green power LED and the red lock LED are illuminated as the physics module (rear right of the enclosure) begins to rise in temperature at 21.3C (ambient 18.6C).

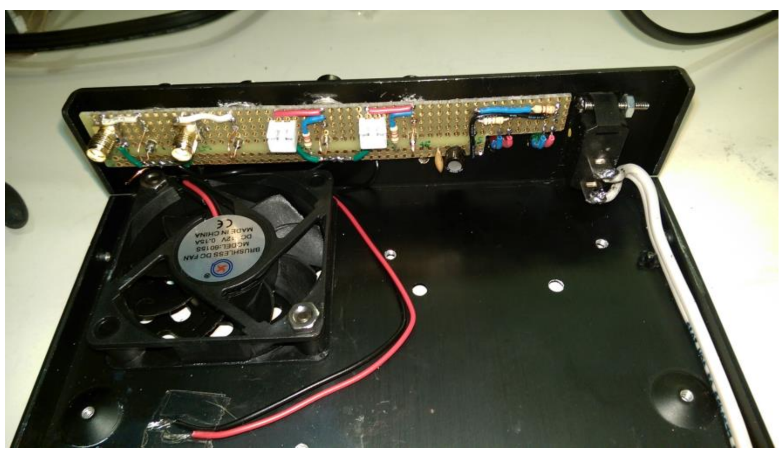


Figure 5.9: The enclosure with the fan and simple breadboard PCB attached to the BNCs on the front panel.

The FE5650A, power supply, fan, and various connectors and components are housed in a surplus all-aluminum enclosure (2.0mm thick) originally intended for a DuraCom power supply (hence the front label that could have been removed). The FE5650A was secured to the enclosure by passing 4-40 screws through the enclosure into the tapped holes. By the way, the holes can be positioned by either photocopying the FE5650A tapped holes or tracing them onto paper with a pencil, and then transferring the positions to the bottom of the case. Prior to attaching the FE5650A, thermal grease was added to the FE5650A aluminum plates between the tapped holes to help transfer the heat to the metal enclosure (Figure 5.2). As a note, the tops of the two aluminum FE5650A plates (with the tapped holes, Figure 5.2) should be flush (i.e., form a plane) so that air gaps won't develop when fastened to the final enclosure/heat sink. The black outer shell of the FE5650A should be removed to further reduce the temperature of the circuit boards and the physics module if the unit is placed within an enclosure. Figure 5.8 shows a temperature meter to monitor room temperature and the physics package temperature. An inexpensive k-type thermocouple is temporarily fastened to the Rb physics module using Kapton/polyimide tape. To ensure electrical isolation of the k-thermocouple, a strip of Kapton tape was placed between the k-thermocouple and the grounded metal enclosure of the Rb physics module.

The power supply is secured as follows. Holes are tapped into the enclosure bottom corresponding to the mounting holes on the power supply. A piece of approximately 1/8 inch thick acrylic the same size as the power supply PCB was placed between the metal enclosure bottom and the PCB to electrically isolate the PCB from the metal case (which is grounded). The melting temperature of the acrylic is 160C/320F which is sufficiently higher than the expected 50C of the heat sink on the power supply. Be careful when attaching the power supplies to the AMP connector – verify and double verify the connections before applying power. If the power supply is thin enough, stand-offs can be used.

The enclosure has a fan mounted to the base (see the left side of Figure 5.9) with air holes in both the base under the fan and in the sides of the top of the enclosure. The fan was not used for the initial tests and the top of the enclosure was not attached. The rats-nest of cables and wires appearing near the front of Figure 5.8 should be shortened/cleaned-up for the final version for esthetics and more importantly, to improve airflow from the fan. The ribbon and power cables were made longer than necessary for the initial tests of multiple FE-5650A units.

The front of the enclosure has four BNC connectors. The test signal attaches to one and the 1PPS to another. The remaining two BNCs externalize TTL square wave and sinewave signals internal to the FE5650A as detailed in an ensuing chapter. The front of the enclosure has an ON/OFF switch along with the green ON/OFF LED next to it and a red Lock LED next to the green one. The lock LED will remain illuminated until the Rubidium standard locks the frequency. Although not used for the initial tests and not seen in Figures 5.8 and 5.9, a 4 pin mini-XLR connector on the back passes RS232 signals to/from external computers or microcontrollers for the purpose of tuning the output frequency. It is possible to use the 4th pin to carry 5 volts to power an external microcontroller and LCD display; in such a case, the pins would carry +5, transmit, receive and ground.

The FE-5650A can be easily disconnected and removed from the final enclosure since it uses cables with connectors. The coax cables that appear in the lower right side of the enclosure (Figure 5.8) were added to the unit to access the internal square and sinusoidal signals as detailed in Chapter 7. The coax cables are not needed for the initial tests. The BNC connectors are attached with nuts to the enclosure front panel.

BNC Female Bulkhead Solder Connector
Amazon Stock Identification Number ASIN: B01L6GTL10

A cut-to-size experimenter's prototype PCB is soldered to the back lug on the BNC connectors as shown in Figure 5.9. The Adafruit PCB Breadboards can be soldered from both sides.

Adafruit Full Breadboard
Amazon Stock Identification Number ASIN: B00SK8KAMM

Female SMA connectors were soldered to the breadboard for the coax cables from the FE5650A.

SMA connector, Female, PCB Mount, Straight
Amazon Stock Identification Number ASIN: B07GXSN7VS

The use of pigtailed SMA connectors makes it easy to remove or replace the FE5650A as desired. The resistors for the test signal, lock alarm, 1pps, and on/off LED were also placed on the breadboard.

As a note, **if** for some reason pin 11 with the test signal does not properly operate, it is still possible that the unit internally generates the correct signal. Chapter 7 will detail the method of accessing the desired sine and square waves directly from the internal sources.

Section 5.6: Initial Test Results

As mentioned, when the FE-5650A Opt. 58 unit was not secured into the enclosure but tested in the position shown in Figure 5.2 sitting on a metal desk, the temperature rose to approximately 54C. The ambient temperature was approximately 19C. The temperature above the ambient room temperature, the relative temperature T_{rel} was then approximately 34C. Determining the above-ambient relative temperature T_{rel} allows one to determine the approximate module temperature T_{mod} for various ambient temperatures T_{amb} as roughly

$$T_{mod} = T_{amb} + T_{rel} \tag{5.1}$$

although good temperature regulation will invalidate this condition since T_{mod} will be fixed regardless of the ambient temperature. For the present case where the FE5650A has the configuration of Figure 5.2 in still air and with the black shield in place, the module temperature will be $T_{mod} = T_{amb} + 34$ in degrees centigrade.

Consider the situation of the FE5650A mounted in the metal enclosure shown in Figure 5.8 – the enclosure top was not in place and the fan was not operated. The temperature of the Rb module, the voltage at the cathode of the LED, and the frequency were all measured at 30 second intervals until 5 minutes and then the measurements were made once per minute. It should be pointed out that the frequency changed between the 30 sec readings; nevertheless, the 30 second readings provide some indication of the frequency swings expected prior to the frequency lock condition.

The results appear in Figure 5.10. The frequency locked at 3mins for this FE-5650A unit. The temperature rose to 33.6C above the ambient temperature of 19.3C. The desired/set output frequency is 8,388,608 Hz. The frequency error is the difference between the measured frequency and the set frequency measure in Hz. It should be noted that the frequency varied more than shown prior to the frequency lock time because measurements were made only once every 30 seconds during that period.

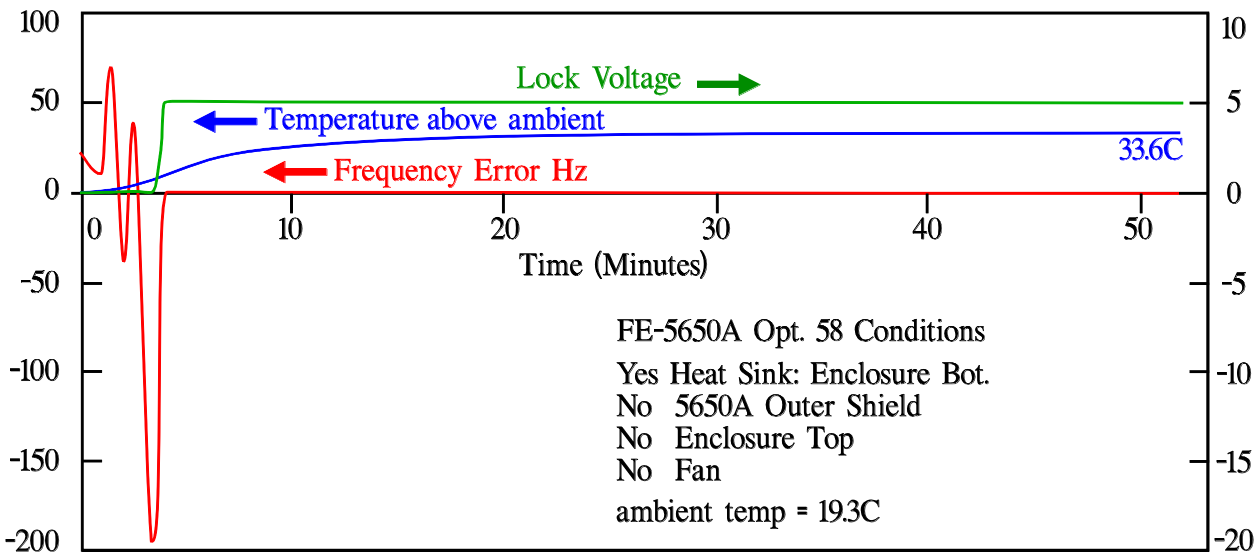


Figure 5.10: The Frequency Error, Temperature above ambient and Lock Voltage. The measurements were recorded every 30 seconds for 5 minutes after which time they were recorded every 60 seconds.

The 1PPS was tested using an older digital oscilloscope HP54111d in single shot mode. The pulse height was 5volts and pulse width was 840nSec into 50 Ohms. A 20MHz analog scope was not sufficient to view the 1pps signal even with a trigger because the waveform would only appear once every second for duration of less than 1 microsecond.

Section 5.7: References

[5.1] DDS board: original board schematic and simulations for mods for FE-5680A but DDS same. Includes using serial port, circuit layout on pcb. Very complete.
Matthias Bopp, D.C.Johnson, Deflef, "A precise reference frequency not only for your ham radio station" Rev 1.0 (2013)
http://www.redrok.com/Oscillator_FE-5680A_precise-reference-frequency-rev-1_0.pdf
Vers 0.4 with some Basic programing notes
https://www.fetaudio.com/wp-content/uploads/2009/10/FE-5680A-modifications.pdf

[5.2] Hacking the FE-5650A. Includes information on the power supply, sending and receiving information from the FE-5650A and other notes. Consider making a donation for providing valuable information.
http://www.ko4bb.com/manuals/70.21.206.217/FE_5650A_Opt_58_hack.pdf

[5.3] Modifying the FE-5650A including power supply requirements, microcontroller connections, DDS board functions
Mark Sims, "FEI Rubidium Oscillators comment,"
http://www.ko4bb.com/doku2015/doku.php?id=precision_timing:rubidium_oscillators
https://www.mail-archive.com/time-nuts@febo.com/msg13486.html

[5.4] Skip Withrow, "One sure way to kill your FE-5680A and FE-5650A," NARKIVE Mailing List Archive, Time-Nuts@febo.com
https://time-nuts.febo.narkive.com/Kjd30sgK/one-sure-way-to-kill-your-fe-5680a-or-fe-5650a

[5.5] Another Rb Standard modification video
https://hackaday.com/2013/08/05/turning-a-rubidium-standard-into-a-proper-tool/

[5.6] Example temperature variation
http://freqelec.com/pdf/rfs_12pg.pdf

Chapter 6: Serial Port and Initial Frequency Tests

The FE5650A has an embedded serial port supported by a PIC microcontroller. The serial port allows an external computer or microcontroller to set the FE5650A output and reference frequencies and to request the status. Some versions of the FE5650A externalize the serial port through a DB9 connector but those featuring the AMP connectors, as tested herein, do not carry the serial data.

This chapter details the FE5650A embedded serial port for both the traditional and TTL RS232 protocols. The TTL RS232 with its 0-5V voltage swings should be used with microcontrollers whereas the classic version with its -10V to +10V voltage swings (albeit low current) can be used with an USB-RS232 adapter for computers. A Level Shift Level Inverter LSLI circuit can be constructed, as detailed, to convert from one protocol to the other. The chapter provides discussion and internet links for USB-RS232 adapters and for terminal/communications software compatible with Windows-based computers. A simple loopback connector for the serial port allows the software to be tested without engaging the FE5650A. The chapter delivers examples for changing the frequency and observing the changes on an inexpensive Sinometer VC2000 frequency counter. For digital meters with frequency counters not capable of resolving the 8-14MHz rate, the chapter includes the schematic for constructing a divide-by-four circuit. Plots are also included of the spectral components and frequency response at the output of the unit. Finally, the chapter examines an intriguing case study of potential data encryption by the FE5650A.

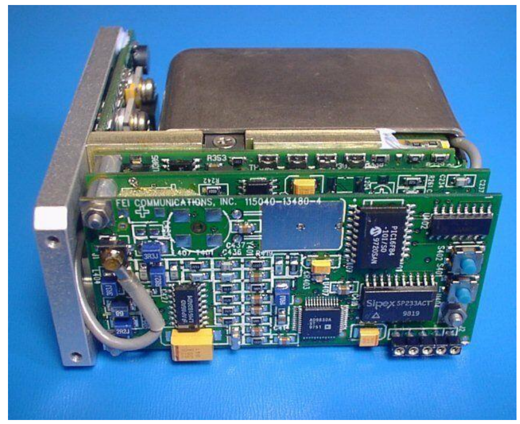


Figure 6.1: A view of the DDS board with J2 for the RS232 on the lower right side. And J1 with thin gray coax at upper left.

Section 6.1: Serial Port Connections

The FEI Rubidium Frequency Standards (RFS), specifically FE5650A and FE5680A (and others), include an embedded RS232 serial port on the Direct Digital Synthesizer (DDS) board shown in Figure 6.1. Some products express the RS232 signals through a DB9 connector on the front panel [6.1-2] although the pin assignments might not match those for the typical serial port. In the case of the FE-5650A with the AMP connector (c.f., black connector in Figure 5.1), the internal RS232 signals can be tapped at an internal dedicated port or at the pins of the PIC microcontroller. The unit provides the standard RS232 port with -10 to +10 volt signals (low current) at connector J2 on the AD9830A DDS board (Figure 6.1). Connector J2 is a five-pin PCB (female) header strip at the lower right hand corner of the DDS board. The three left most pins on J2 (ground, 5650A Transmit, 5650A Receive) connect to an SP233ACT (enhanced line driver/receiver) integrated circuit which inverts the logic levels and uses a charge-pump to develop the negative voltage signals for the traditional RS232 port. The traditional RS232 connections can be used with a USB-RS232 adapter and a computer with communications

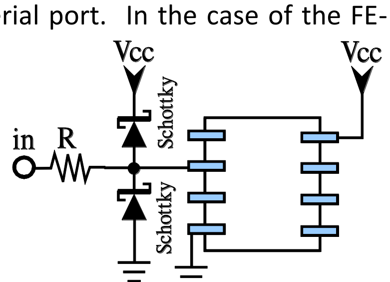


Figure 6.2: Two Schottky diodes (such as BAT41 or 1N5817) clamp the voltage on the pin to a range of roughly -0.2 to Vcc+0.2.

software as will be discussed below.

The traditional RS232 signals probably should not be used with a microcontroller without diode voltage clamps (Figure 6.2) because of the voltage levels involved. Microcontrollers typically use a TTL RS232/USART that operates in the range 0-5V or 0-3.3 and have inverted logic levels compared with the traditional RS232. Some microcontrollers such as the ATMEL XMEGA128A4U can invert the logic levels but require the voltages to remain between ground and VCC (except possibly for very low current signals that won't exceed the capacity of the parasitic diodes on the pins). For the TTL RS232/USART, the proper signals can be accessed at the SP233ACT integrated circuit on those pins that actually connect to the PIC16F84 which is located just above the SP233ACT chip.

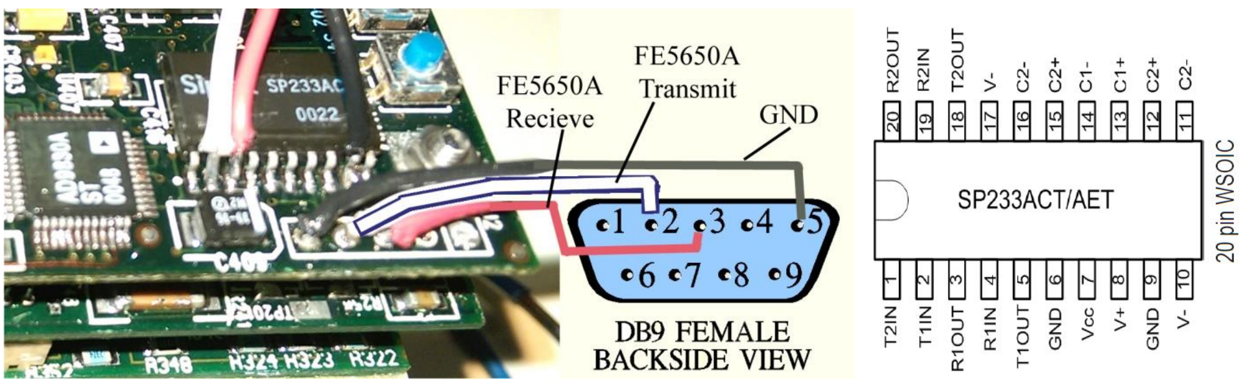


Figure 6.3: LEFT: Connections to the SP233ACT for the TTL-RS232 and to J2 for the standard RS232 signals. Note the small white dot at the lower left of SP233ACT marks pin 1 for the chip. The white wire on the SP233ACT pin 2 is the TTL-RS232 transmit line that can be connected to a 5V microcontroller receive pin. The red wire on the SP233ACT pin 3 is the TTL-RS232 receive line that can be connected to a 5V microcontroller transmit pin. See caution message in text regarding direct connection to the SP233ACT. Connector J2 is on the lower right side of the PCB. The square white box around the right most pin on J2 will be taken as pin J2-1. The black, white, and red wires carry the ground, 5650A transmit, and the 5650A receive signals respectively. MIDDLE: Rear view of the female DB9 with the wires from the FE5650A transmit, FE5650A receive and ground soldered to pins 2, 3, and 5, respectively. RIGHT: Pin out for the SP233ACT.

The initial tests of changing the RFS output frequency use a USB-RS232 adapter and the traditional RS232 signals available at connector J2 (See Section 6.2 for example USB-RS232 adapters). There are several methods suitable to make connections to J2. The first method for some of the initial tests used wires of sufficient diameter such that when inserted into the header holes, they did not fall out. Actually, stranded wire for which the strands were soldered together worked best and did not destroy the socket pins. The second method used male headers or a trimmed IC socket with the male pins pressed into the receptors and wires soldered to the other ends of the male pins. The third method unsoldered the J2 connector in Figure 6.3 and replaced it with wires soldered in the left-most three holes (pins 5,4,3). These soldered J2 wires are used for the standard RS232 signals. Regarding the CAUTION mentioned in the figure caption with respect to the two wires soldered to the SP233ACT chip (plus a ground wire – not shown): Initial tests show that these wires can connect directly to an external microcontroller USART pins without harm even though it is possible for two connect pins to be outputs and thereby risking damage to either of the circuits when in contention. For this reason, it's best to include additional circuitry as discussed in Section 10.3.

Figure 6.3 and Table 6.1 show the required connections. For the standard RS232 signals and a USB-RS232 adapter, use a DB9 FEMALE connector and make connections as shown in the left and middle panels of Figure 6.3. The proper USB-RS232 adapter will have the male counterpart connector.

The table shows pin J2.5 connects to DB9.5, and J2.4 (FE5650A transmit) connects to DB9.2 (PC receive line), and J2.3 (5650A receive) connects to DB9.3 (PC transmit line).

Table 6.1: Table of connections.

Function	J2 – Standard RS232	DB-9 female	IC for TTL RS232
Ground	Pin 5 (left most): J2.5	Pin 5: DB9.5	Pin 9: SP233.9
FE-5650A Transmit	Pin 4 (2nd to left): J2.4	Pin 2: DB9.2	Pin 2: SP233.2
FE-5650A Receive	Pin 3 (middle): J2.3	Pin 3: DB9.3	Pin 3: SP233.3

The last column in the table identifies the MAX SP233ACT pins (9, 2, 3) that provide the FE5650A ground, transmit, and receive functions, respectively. It's best to be able to disconnect the wires at will especially if working with several of these units for testing and reserve soldering wires to the PCB for a final version. Once the above connections are made, the FE-5650A is ready for the RS232 status test.

For those readers planning to interface a microcontroller, the connections can be made to the SP233ACT chip as indicated in Table 6.1 although the circuits described in Section 10.3 should be used. However, it was found possible to use the USB-RS232 adapter with these SP233ACT connections (without the additional circuits) so long as an inverter is used for the transmit and receive signals. The Level-Shift Level-Invert LSLI circuit shown in Figure 6.4 was found to work for the FE-5650A. The circuit can be built into a small plastic box along with the two DB9 connectors and the 78L05 power circuit. The 2N2222 transistors provide the logic inversion and the 78L05 restricts the voltage range from ground to 5V. For 3.3V microcontrollers, either use the voltage clamp in Figure 6.2 at the microcontroller USART receive pin (R=1k) or replace the 5V regulator in Figure 6.4 with a 3.3V regulator such as L78L33 available from Digikey, Mouser, and Amazon among others. Also for the 3.3V, lower value resistors might be used such as 6.8k depending on the transistor gain. Some commercial TTL-RS232 converters might be suitable for your purposes such as Amazon ASIN B00OPTOKI0.

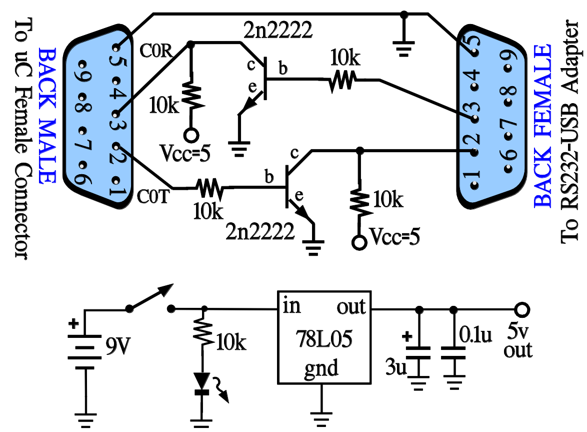


Figure 6.4: The Level-Shift Level-Invert LSLI circuit.

Figure 6.5: The loopback connector is a DB9 Female with a wire soldered between pin 2 and pin 3.

Prior to loading the various possible terminal software (i.e., communications software), it is sometimes helpful to have a loopback connector for the RS232. The loopback connector simply routes the data sent from the terminal back to the terminal and makes it possible to become familiar with the software without using the FE-5650A. The connector needs to be a Female DB9 to attach to the male DB9 on the USB-RS232 adapter. Simply solder pin 2 to pin 3 and attach the modified DB9 connector to the USB-RS232 adapter.

Finally, one additional circuit might be useful to those readers without a proper frequency counter but instead use an inexpensive digital multimeter, such as the Southwire 10040N, capable of measuring frequency frequencies up to about 4MHz. The frequency counter in the Southwire does not have sufficient range and resolution to calibrate the Rb source. To measure frequencies in the range 4 to 16 MHz albeit with lower resolution, the binary divider circuit in Figure 6.6 will reduce the frequencies to the range of 1MHz to 4MHz.

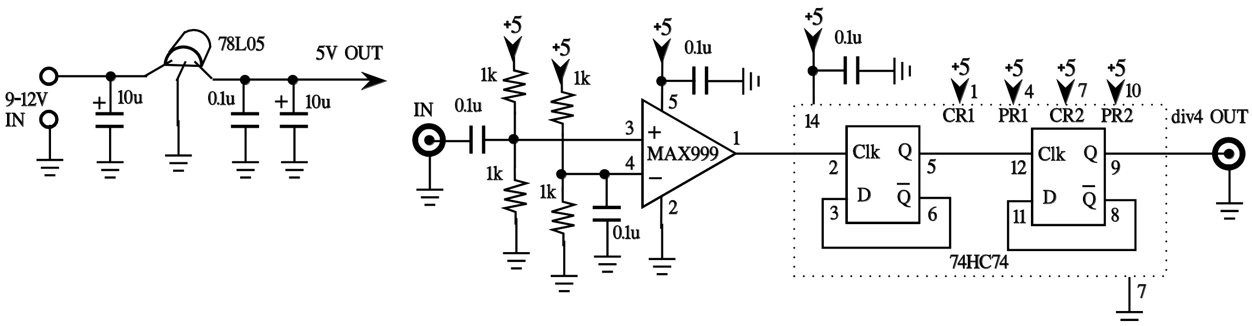


Figure 6.6: A divide-by-four circuit for frequencies above approximately 1MHz, and 400 mVpp and up to 5V.

Section 6.2: Communications Software and USB-RS232 Adapter

Continued initial testing of the FE-5650A using the RS232 connector J2 requires communications software on the computer and an USB-RS232 adapter. These ubiquitous USB-RS232 adapters can be found on Amazon.com and EBay.com for $10 or there about. Older computers had an actual RS232 port especially those computers meant to run Windows XP and older. The USB-RS232 adapters for the Windows 10 computers should have a *male* DB9 connector to interface with the circuits described in previous sections. Some so called ‘USB-Serial’ adapters have the female connectors but are not meant for the RS232 connections – their drivers are not compatible – been there, done that. Here are a couple of USB-RS232 adapters available at Amazon.com.

Sabrent, USB 2.0 to Serial (9-Pin) DB-9, RS-232 Converter Cable, Prolific Chipset
Amazon Stock Identification Number ASIN: B00IDSM6BW

UGREEN, USB 2.0 to RS232, DB9 Serial Cable, Male A Converter Adapter with PL2303 Chipset
Amazon Stock Identification Number ASIN: B00QUZY4UG

The older Windows operating systems came with the HyperTerm.exe which was suitable software for the serial RS232 communications (although not great by today standards). While the older software will still run under the newer operating system, it would be better to use one of the newer, more functional terminal/communications packages described below.

To communicate with the FE-5650A, the computer will need to have terminal/communications software. We use a computer with the Microsoft Windows 10 operating system. There are several communications packages (free) listed below – the internet has many but we have tested those listed. If the program doesn’t automatically list the proper com port (ours was 4 or 5 depending on the USB port used), then open the Device Manage by right clicking the Windows symbol in the lower left corner of the screen, selecting device manager, and noting the com port under the item “Ports (COM & LPT)”. Ours reads "USB-SERIAL CH340 (COM5)" at present. So we would enter COM5 if the program did not automatically detect the correct COM port. Note, online references indicate the FE5650A uses the baud of 9600 (bps) but we find **9950** which is nonstandard – see ensuing sections on the encryption mystery.

1. Appendix 9 lists the source and design code for FE5650A Interface software written using Visual Studio (VB.Net/C#.Net) [6.7]. The software allows the baud rate to be set to any value. The software can be downloaded or found on a flash drive (etc.). It has all the needed components for testing the FE5650A.

2. For those programming the Microchip-Atmel line of microcontrollers, the Atmel Studio has an excellent terminal plug-in that displays the received or transmitted characters in either text or hex code plus it has logging capability. Once installed, it will be found under the View menu in the Atmel Studio. The terminal plugin does accept non-standard baud rates (such as 9950). One negative point is that the CR (i.e., ASCII(13)=ASCII(0x0d)) in the received ASCII causes the display to overwrite previously received ASCII such as for the reference frequency and Fcode. There might be settings to work around the overwrite issue. The display for the hex version of the ASCII does not overwrite but one must use ASCII tables such as Table 6.2 below to translate from hex to text.

3. Termite is a small, easy-to-use program available at:
   https://www.compuphase.com/software_termite.htm
   Notice this is Termite (not Termie). Termite has a plug-in for sending/displaying the HEX version of the text. Enter the parameters in the settings as 9600 baud, 8 data bits, 1 stop bit, no parity, no flow control. The 'settings' has an unconventional 'forward' and it should be 'none'. At least check the radio button for Append CR under Transmitted text in the settings. Use local echo to see what has been typed. To display/send HEX, check the "HEX View" in the Plug-ins list. The program can log the exchange. Termite does accept nonstandard baud rates. Very Nice.

4. Termie is another small, easy-to-use program available on SourceForge.net.
   The site offers both the C source code and the executable. Unless you are planning to modify the source code, download the Binary Zip file and extract the file to a suitable directory.
   Termie-1.0-Source.zip (67 KB) - Complete Visual C# project.
   Termie-1.0-Binary.zip (53 KB) - Program executable in zip archive.
   The older version of Termie that we tested does not have an option to send/display HEX but it can echo back the sent characters and it can add a carriage return or line feed character. The received special characters are displayed in brackets such as <Enq>. *An arbitrary baud rate cannot be set*.

5. The communications software at www.puTTY.org is an SSH and Telnet client. This is nice software that handles RS232/serial in addition to SSH and telnet and others. When PUTTY starts, the Session setting will appear. Click the 'Serial' radio button on the right side and enter the COM port for your adapter under 'Serial line' and enter '9600' under 'Speed'. Next, in the category list, click on 'Serial' at the bottom, and on the right set 9600 baud, 8 data bits, 1 stop bit, no parity and no flow control. Then click 'Open'. It does accept nonstandard baud rates. We noticed that the computer enter key must be pressed a number of times for the puTTY window to respond; possibly the software has a delay for setup.

6. The HyperTerm communication program from Microsoft can still be found and used. A related DLL file might need to be downloaded and if so, the hyperterm and DLL file should be located in the same directory on the computer hard drive. The Hyperterm files can also be found in the XP operating system and copied to the new operating system. There is a newer version of HyperTerm but it does not allow non-standard baud rates such as 9950.

Once the USB-RS232 has installed, attach and power-on the FE5650A, load the communications software. So far, our adapter has been found on COM4 and 5. The parameters for the serial communications should be set to 'NO' handshaking – if that's *not* set to 'NO', the communications terminal will never receive any data. To start, the parameters should be set as the ubiquitous 9600N81:

9600 = Baud   N = No Parity   8 = number of data bits   1 = 1 stop bit

Depending on your unit, it might be necessary to find the right baud rate as discussed further below. We start with 9600. Finally, and almost as important as the 'NO' handshaking, the communications program should be set to send a carriage return, which is ASCII(13) to the FE-5650A after each command (where 13 is the decimal number). Some people suggest that it is necessary to send the line feed, which is ASCII(10), but so far, that has not been our experience.

Table 6.2: Table of ASCII characters along with the decimal, hexadecimal and binary representations.

ASCII TABLE

Dec	Hex	Bin	Char	Dec	Hex	Bin	Char	Dec	Hex	Bin	Char	Dec	Hex	Bin	Char
0	0	00000000	Null	**32**	**20**	**00100000**	**Space**	64	40	01000000	@	96	60	01100000	`
1	1	00000001	Start Heading	33	21	00100001	!	**65**	**41**	**01000001**	**A**	**97**	**61**	**01100001**	**a**
2	2	00000010	Start Text	34	22	00100010	" DQuot	66	42	01000010	B	98	62	01100010	b
3	3	00000011	End Text	35	23	00100011	#	67	43	01000011	C	99	63	01100011	c
4	4	00000100	End Transmission	36	24	00100100	$	68	44	01000100	D	100	64	01100100	d
5	5	00000101	Enquiry	37	25	00100101	%	69	45	01000101	E	101	65	01100101	e
6	6	00000110	Acknowledge	38	26	00100110	&	70	46	01000110	F	102	66	01100110	f
7	7	00000111	Bell	39	27	00100111	' SQuot	71	47	01000111	G	103	67	01100111	g
8	8	00001000	Backspace	40	28	00101000	(	72	48	01001000	H	104	68	01101000	h
9	9	00001001	Horizontal Tab	41	29	00101001	)	73	49	01001001	I	105	69	01101001	i
10	**A**	**00001010**	**Line Feed**	42	2A	00101010	*	74	4A	01001010	J	106	6A	01101010	j
11	B	00001011	Vertical Tab	43	2B	00101011	+	75	4B	01001011	K	107	6B	01101011	k
12	C	00001100	Form Feed	44	2C	00101100	, Com	76	4C	01001100	L	108	6C	01101100	l
13	**D**	**00001101**	**Return**	45	2D	00101101	- dash	77	4D	01001101	M	109	6D	01101101	m
14	E	00001110	Shift Out	46	2E	00101110	. per	78	4E	01001110	N	110	6E	01101110	n
15	F	00001111	Shift In	47	2F	00101111	/	79	4F	01001111	O	111	6F	01101111	o
16	10	00010000	Data Link Esc	**48**	**30**	**00110000**	**0**	80	50	01010000	P	112	70	01110000	p
17	11	00010001	Device Control 1	49	31	00110001	1	81	51	01010001	Q	113	71	01110001	q
18	12	00010010	Device Control 2	50	32	00110010	2	82	52	01010010	R	114	72	01110010	r
19	13	00010011	Device Control 3	51	33	00110011	3	83	53	01010011	S	115	73	01110011	s
20	14	00010100	Device Control 4	52	34	00110100	4	84	54	01010100	T	116	74	01110100	t
21	15	00010101	Neg Acknowledge	53	35	00110101	5	85	55	01010101	U	117	75	01110101	u
22	16	00010110	Synchronous Idle	54	36	00110110	6	86	56	01010110	V	118	76	01110110	v
23	17	00010111	End Transm Block	55	37	00110111	7	87	57	01010111	W	119	77	01110111	w
24	18	00011000	Cancel	56	38	00111000	8	88	58	01011000	X	120	78	01111000	x
25	19	00011001	End Medium	57	39	00111001	9	89	59	01011001	Y	121	79	01111001	y
26	1A	00011010	Substitute	58	3A	00111010	:	90	5A	01011010	Z	122	7A	01111010	z
27	1B	00011011	Escape	59	3B	00111011	;	91	5B	01011011	[	123	7B	01111011	{
28	1C	00011100	File Separator	60	3C	00111100	<	92	5C	01011100	\	124	7C	01111100	|
29	1D	00011101	Group Separator	61	3D	00111101	=	93	5D	01011101	]	125	7D	01111101	}
30	1E	00011110	Record Separator	62	3E	00111110	>	94	5E	01011110	^	126	7E	01111110	~
31	1F	00011111	Unit Separator	63	3F	00111111	?	95	5F	01011111	_	127	7F	01111111	Del

Some of the communications programs allow the user to type the commands in ASCII HEX code in which case, the carriage return can be included as part of the command. For example, sending 'S' and a carriage return in ASCII hex code is

S <cr> => 53 0D

The space between HEX 53 and the HEX 0D can be omitted. For significant work with ASCII code, it would be wise to have a copy of the ASCII table (available on line) or the one provided in Table 6.2.

Section 6.3: FE5650A Commands and Tests

A number of online resources list the possible commands for the FE-5650A [6.3-6]. It's a good idea to refrain from entering a command with unknown consequences. For example, our units appeared to encrypt the information returned from the unit. The question would be whether the innocuous Status command might be encrypted and accidently end up as an R command (which would erase the internal reference setting – not good). However, the S command is unlikely (probability of 1/126) to be encrypted as an R and further, the correct command to set the reference is 'R=' which would require the 'S' to produce the two characters – very unlikely. So we felt comfortable trying the S as well as setting the frequency using 'F='.

Table 6.3: Known commands for the FE-5650A Rb Standard. Avoid using E especially if the unit is not properly working.

Command	Send	Effect	Example of returned data, usage or comment
Status	S	Returns Ref and Fcode	R=50255057.012932Hz F=2ABB504000000000 OK
Set Freq	F=…	Sets new output freq.	Use: F=2ABB5040 (AD9830A uses first 4 bytes).
Set Ref	R=…	Sets reference freq (avoid)	From nonvolatile memory
Enter R, F	E	Sets R, F as default (avoid)	Overwrite nonvolatile mem from previous F= or R=
Serial Num	N	Rumored to return SN	Not observed

All commands should be terminated with a carriage return CR which is ASCII 13dec or 0Dhex. Many terminal programs will place the CR for you. Generally a FE5650A command uses an upper-case letter but the unit appears to accept a lower case letter as well.

The final couple of tests for the FE-5650 start by connecting the unit to the USB-RS232 adapter and starting the terminal software. The 'Test Signal' at the AMP connector pin 11 can be connected to the scope or frequency meter. Keep in mind that the output signal from pin 11 (with the 2.2k resistor in place) will be a sinusoidal signal of approximately 1.5Vpp offset by approximately 2.5V. As previously discussed, the frequency will be accurate once the Lock LED extinguishes.

The first command should be the 'S' for Status followed by the always-required carriage return <cr>. Table 6.3 shows the unit should return something similar to

R=50255057.012932Hz F=2ABB504000000000 OK

but every unit will return slightly different numbers. As discussed below, four of our units returned some characters outside of the first 127 ASCII that appeared to be some type of rotating / changing encryption. If the unit returns readable numbers then write them down and save them. If your unit does not, then it will be necessary to find a key to the code (see below) or use a separate means to obtain a reference frequency. *Actually the stored reference differs from the true reference required to set the*

AD9830A DDS chip that generates the various frequencies and it might be desirable to use the alternate methods for the working unit anyway. Chapter 6 discusses a method to obtain a reference frequency. **NOTE: turning off the unit without sending the 'E' command will cause the unit to return to the previously set R and F values – this is a good precaution.** As another note, the AD9830A only uses 4 bytes (8 hex digits, 32 bits) for Fcode, but it is possible to send 8 bytes (64 bits). The FE-5650A appears to retain the last 4 bytes and return them as part of the Status query but it's not clear as to their purpose.

For now, regardless of the intelligibility of the returned status, the test of the unit's capability of changing the output frequency can be attempted. You will need a frequency counter or oscilloscope capable of distinguishing at least 8 MHz and 12 MHz. It is possible to use an inexpensive multimeter such as the Southwire 10040N with a basic frequency counter but it will be necessary to construct a divide-by-4 circuit (Figure 6.6) to transform the 6-14MHz of the unmodified FE5650A to the range of the multimeter.

1. Set a number near 8.0MHz

   Actually the FE-5650A should produce the frequency of 8.388608MHz by default so no further commands need be sent. An approximate value for Fcode is 28BB503E to set a frequency of 8.0MHz. Here, Fcode refers to the HEX digits sent to the AD9830A DDS in the unit to set an output frequency. To set the frequency, send the following text in the terminal command line and don't forget a <cr>:

   F=28BB503E

2. Set a number near 12.1 MHz

   The Fcode is approximately 3DBB503E. So send the following string of characters and remember that either the terminal program or you must include the <cr>.

   F=3DBB503E

3. Set a number near 10.0 MHz

   Send the Fcode of 32F0AD9C as text:

   F=32F0AD9C

If the frequency changed on the frequency counter, oscilloscope or meter then consider the unit working and proceed to the next chapter to make the final modifications to the unit (also Section 10.3).

Section 6.4: Response and Spectrum

We jump ahead a little to show the results of using the sinewave obtained from connector J1 on the DDS board as detailed in the next chapter. The FE5650A (unit #3) output voltage vs. frequency response can be obtained using an oscilloscope of sufficient bandwidth (20MHz or larger).

The FE5650A sinewave output from connector J1 on the DDS board (covered in the next chapter) couples into a three foot long, 50 Ohm coax cable which then attaches to an oscilloscope. The termination impedance of the scope is either 50Ohms or 1MOhm whereas the output impedance of the FE5650A is approximately 50Ohms. The peak-to-peak output voltage versus output frequency as measured by the oscilloscope appears in Figure 6.7. The bandpass range (1/2 peak) of the 1MOhm coupling covers approximately 7MHz to 14MHz whereas for the 50Ohm coupling, it covers approximately 6MHz to 13MHz. As a point to keep in mind, the waveform becomes distorted for cables longer than 2 or 3 feet without proper termination (50 Ohms). The distortion can be quantified using the RF spectrum analyzer.

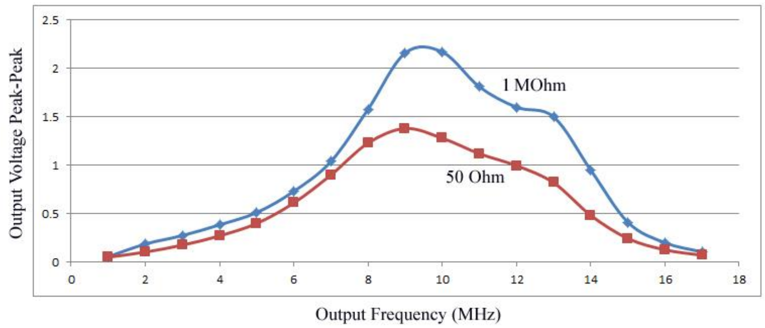


Figure 6.7: The peak-to-peak sinewave output voltage versus the output frequency for the unmodified FE5650A (unit #3 in Table 6.4). The signal passes through a three foot 50Ohm coax cable to an oscilloscope input with either a 50 Ohm or 1MOhm input impedance.

We next determined the RF spectrum of the sinewave output for the unmodified FE5650A (unit #3 in Table 6.4) which provides a fiduciary spectrum to compare with modified units. For example, the next chapter discusses modifications to increase the FE5650A bandwidth which should be expected to change the spurious frequency components of the output spectrum (in particular, the x2 harmonic). For the test of the unmodified unit, we used the oldie-but-goodie HP8568B (Opt. E44) RF Spectrum Analyzer (RFSA, 0-1.5GHz) limited to the scan range of 0 to 20MHz. The unmodified FE5650A was set for 8.388608MHz. We used a three foot 50 Ohm coax cable with the voltage divider shown in Figure 6.8 (in case the FE5650A somehow failed and placed more than zero Vdc or 7Vrms into the RFSA).

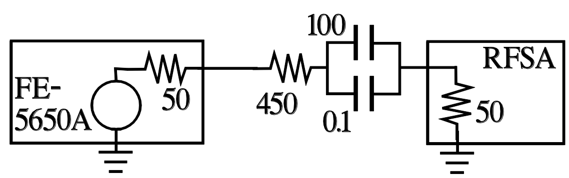


Figure 6.8: Connection between the RFSA and the FE5650A. The caps block DC and the 450 divides the voltage with the 50Ohm input impedance of the RFSA.

The result of the spectral measurement appears in Figure 6.9. The plot shows the power density dBm in the output signal from 0 to 20MHz. The RFSA has 10dB attenuation, 3kHz Res BW, 300Hz Vid BW and shows 10dB per vertical division with the reference voltage at the top of 0dBm. Notice the noise near the floor. The center peak at 8.39MHz is roughly 60dB (10^6) above the harmonic at 16.78MHz. Other spurious components can be seen at roughly 70dB below the main peak.

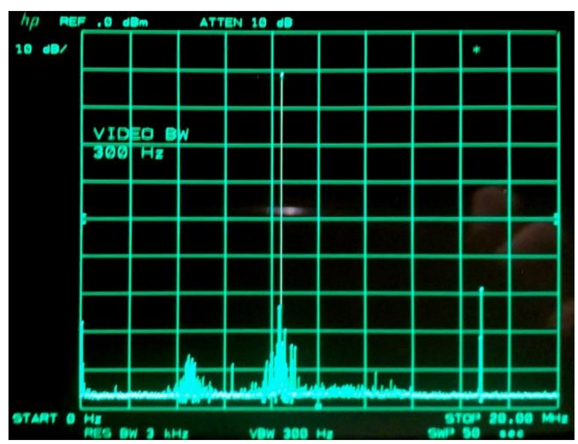


Figure 6.9: Spectrum of the output of an unmodified FE5650A set for 8.388+ MHz (unit #3 in Table 3.4).

Section 6.5: FEI Encryption?

All of the tested FE-5650A units appear to encrypt the response to the Status command thereby obscuring the stored reference frequency and the Fcode to achieve the output frequency. Here Fcode refers to the HEX digits sent to the AD9830A DDS integrated circuit, which combines it with the reference frequency to set the output frequency. The response to the Status command for these units appear to alternate between several different possible encryptions/encodings. Frequency Electronics

Inc. (FEI) sells the units with various options and various prices. One option for the FE-5650A is an externally accessible RS232 port with the capability of setting various output frequencies. Perhaps the company has different pricing for the different options even though the PCBs have all of the functionality built into them, and perhaps encrypting the returned status information helps protect one of the options. Another possibility, as previously mentioned, is that some damage occurred to the nonvolatile memory/flash at some time when the supply voltage was ramped too slowly. However, these units still appear to operate unlike those that were reported as damaged [6.6]. Keep in mind that these units with 'encrypted' Status responses are still usable once having deduced a true reference frequency as will be demonstrated in the Chapter 9. Further and more important, the stored reference frequency of 'unencrypted' units should be verified/calibrated against an accurate frequency standard such as the Global Positioning System Disciplined Oscillator (GPSDO) as detailed in Chapter 9. One of the references [6.3-5] finds the stored reference frequency differs from the true reference frequency by an approximate constant. We can verify this constant which then makes it possible to apply to the stored reference frequency to obtain the true reference frequency without using the GPSDO.

We found the following returned ASCII codes (in HEX) from the 'encrypted' FE-5650A when repeatedly sending the status enquiry to FE-5650A Opt 58. The first few characters of the stored values of R and F should be 50255055… and 32F0AD80…, respectively. Sending the Status command 'S' should then return

R=50255055…F=32F0AD80

Or in ASCII: 52 3D 35 30 32 35 35 30 35 35 … 46 3D 33 32 46 30 41 44 38 30 … (6.1)

Here are the actual responses for sending 'S' ten times. All of the numbers are in HEX for the ASCII codes.

d2 4f 4d 92 aa aa 82 aa b2 2e b5 b3 b3 b6 b6 26 4a 41 c6 9d 26 26 14 0a 91 38 30 46 b0 b0 b0 b0 82 82 6a cf 8b c3

b2 4f 6d 92 aa aa 82 35 36 ae b5 93 93 93 d3 42 e9 20 86 4f 26 36 14 0a 11 e1 b0 c6 b0 b0 82 82 82 82 6a cf a9 f8

52 bd 95 92 aa aa 82 aa b2 f2 35 33 33 b6 b6 b3 4a 81 46 bd b3 26 14 0a 11 e1 b0 14 82 82 82 82 82 82 0d 4f cb 8d

d2 4f 4d 92 aa aa 82 aa b2 2e 35 b3 b3 b6 b6 26 8a 41 c6 bd 26 26 14 0a 91 98 32 c1 b0 b0 b0 82 82 6a 4f 8b c3

d2 4f 4d 92 aa aa 82 aa b2 2e b5 b3 b3 b6 b6 26 4a 41 c6 bd 26 26 14 0a 91 38 30 46 b0 b0 b0 b0 82 82 6a cf 8b c3

52 bd 95 92 aa aa 82 aa b2 f2 35 33 33 b6 b6 b3 ca 81 46 bd b3 26 14 0a 11 e1 b0 14 82 82 82 82 82 82 0d 4f cb 8d

d2 4f 4d 92 aa aa 82 aa b2 2e b5 b3 b3 b6 b6 26 4a 41 c6 bd 26 26 14 0a 91 38 30 46 b0 b0 b0 b0 82 82 6a cf 8b c3

b2 4f 6d 92 aa aa 82 35 36 ae b5 93 93 93 d3 42 e9 20 86 4f 26 36 14 0a 11 e1 b0 c6 b0 b0 82 82 82 82 6a cf a9 f8

d2 4f 4d 92 aa aa 82 aa b2 2e 35 b3 b3 b6 b6 26 8a 41 c6 bd 26 26 14 0a 91 38 30 46 b0 b0 b0 b0 82 82 6a 4f 8b c3

52 bd 95 92 aa aa 82 aa b2 f2 35 33 33 b6 b6 b3 ca 81 46 bd b3 26 14 0a 11 e1 b0 14 82 82 82 82 82 82 0d 4f cb 8d

First notice that the responses appear to select among several 'encryptions' as evident by comparison with the expected response in Expression 6.1. While it might be possible to 'reverse engineer' the responses, not only is it unnecessary by performing a calibration, but it would likely only apply to these older units. As for the possible methods used to encode the response, patterns are evident for some of the characters. For the returned lines, the first character appears as 52 as it should for the 'R' character. For others lines we see the first character of D2 which can be found from the proper value of 'R' as 52 by adding 80 which produces D2; this addition corresponds to changing the leading bit in 52 = 0101 0010 to D2 = 1101 0010. The returned first character selects among 52, D2, B2. Similarly ASCII tables show '=' corresponds to 3D and adding 80 produces BD which appears in some lines. Other patterns such as 'aa' representing '5' might be seen except then where's the starting 50. Sometimes the digits such as 25505 might be returned something like 92 95 95 90 95 where the second digit is the required number. The issue becomes one of the pattern changing from line to line even though each line should produce the same result for the singular status enquiry. Additionally and worse, the lengths of the returned lines vary suggesting dropped characters.

The issue of dropped characters suggests an alternate approach that works – next section. However, even when the alternate approach does not work, then so long as setting a new frequency 'F=' works, there isn't much point in trying to decrypt the response since fortunately, an inexpensive GPSDO can be used to determine an actual reference frequency needed to calculate the required Fcodes to set the desired output frequency. Chapter 9 will describe a method of calibration.

Section 6.6: FEI Key

The dropped characters and the changed bits in the characters provide the key to descrambling the response to the status command. These suggest the possibility that the culprit lies with the baud rate or maybe a serial port parameter such as stop bits or parity. The stop bits and parity were checked and they are not the cause. The real culprit was the baud rate. But it's not exactly correct to say 9600 is the wrong baud rate since the FE5650A can still properly receive commands with extensive hex such as F=32F0AD80; however, it is the wrong baud since the FE5650A cannot transmit even one or two characters without an error.

To find the proper bits per second for our units, we started at 9600 baud and first decreased it by steps of 100, but still no response. Then we increased by steps of 100 and found a baud where some of the characters began to appear. So in steps of 50, we found the range of 9850-10050 which averages to 9950. This nonstandard **9950** baud rate worked for both the transmitter and receiver for the FE5650A whereas the 9600 worked only for the FE5650A receiver. Using the 9950 baud rate, the FE5650A returned the Table 6.4 responses to the status enquiry. These will be used in Chapters 8 and 9 in relation to the calibration. As a reminder, when changing the baud rate, the serial port must be closed and then opened again to the new baud rate.

Table 6.4: Response to the status enquiry

Unit	FEI Number	Response to Status Enquiry
1	61627	R=50255054.934100Hz F=2ABB5050 7E8A5200<cr>OK
2	57803	R=50255056.533663Hz F=2ABB503A ACF26C00<cr>OK
3	55380	R=50255055.305534Hz F=2ABB504B 3210BE00<cr>OK
4	59321	R=50255055.802777Hz F=2ABB2A39 1A23AE00<cr>OK
5	71199	Incompatible Interface

As regards the curious happenstance of the embedded PIC microcontroller having the capability of receiving at two separate baud rates but transmitting a single rate, a couple of possible mechanisms come to mind. For the first, it might be possible for the microcontroller to extract the baud rate using a general purpose IO pin by timing the incoming bits. This would suggest that any attempt to send a frequency-set command at any baud should cause the microcontroller to properly responds and make that change. This baud-independent frequency control was verified for Unit #3 in Table 6.4. Another possibility is to run two USART receivers in parallel but these would be fairly baud rate specific. The other units were later found to behave similarly and properly respond to the 9950 baud. Problem solved.

Section 6.7: References

[6.1] Matthias Bopp, D.C.Johnson, Deflef, "A precise reference frequency not only for your ham radio station" Rev 1.0 (2013)
http://www.redrok.com/Oscillator_FE-5680A_precise-reference-frequency-rev-1_0.pdf

[6.2] M. Greenman, https://www.qsl.net/zl1bpu/PROJ/Ruby.htm and links therein.

[6.3] Mark Sims, "FEI Rubidium Oscillators comment,"
https://www.mail-archive.com/time-nuts@febo.com/msg13486.html
http://www.ko4bb.com/doku2015/doku.php?id=precision_timing:rubidium_oscillators

[6.4] Matthias Bopp, D.C.Johnson, Deflef, "A precise reference frequency not only for your ham radio station" Rev 1.0 (2013)
http://www.redrok.com/Oscillator_FE-5680A_precise-reference-frequency-rev-1_0.pdf
Vers 0.4 with some Basic programing notes
https://www.fetaudio.com/wp-content/uploads/2009/10/FE-5680A-modifications.pdf

[6.5] Hacking the FE-5650A. Consider making a donation for providing valuable information.
http://www.ko4bb.com/getsimple/index.php?id=manuals&dir=02_GPS_Timing/FEI
http://www.ko4bb.com/manuals/70.21.206.217/FE_5650A_Opt_58_hack.pdf

[6.6] Skip Withrow, "One sure way to kill your FE-5680A and FE-5650A," NARKIVE Mailing List Archive, Time-Nuts@febo.com
https://time-nuts.febo.narkive.com/Kjd30sgK/one-sure-way-to-kill-your-fe-5680a-or-fe-5650a

[6.7] Source code, listings, downloads, flash drives, preprogrammed microcontroller:
(A) Chapter 10 and Appendix 6 provide the source code for the C/C++ program
(B) Appendices 7-9 provide the PC software
(C) Check EBay.com in connection with the title of this book: software and preprogrammed MEGA328P
(D) check www.AngstromLogic.com for current offerings, or write to Sales@AngstromLogic.com
Keep in mind that Angstrom Logic, LLC is not setup/able to answer questions on the book or software.
(E) Check Amazon in connection with the title of this book.
(F) Note: The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the words) and the contents are copyrighted with all rights reserved (also see front copyright page). The software/firmware is scanned for viruses before posting. Download only from secure sources and rescan for viruses before using.

Chapter 7: Modifications

The FE-5650A Rubidium Frequency Standard (RFS) provides accurate and stable frequency/clock signals in a compact, self-contained, portable enclosure. The RFS is useful for either calibrating equipment with crystal oscillators or perhaps substituting for the crystal oscillator such as in frequency counters or function generators. Besides accuracy and stability, a general purpose frequency standard needs to be tunable with adequate output amplitude. The present chapter shows the modification to externalize the square wave from the MAX 913 comparator and the sinewave from the AD9830A Direct Digital Synthesizer in addition to the modifications to increase the bandpass frequency range of 6MHz to 14MHz to the range of approximately 100Hz to 15MHz.

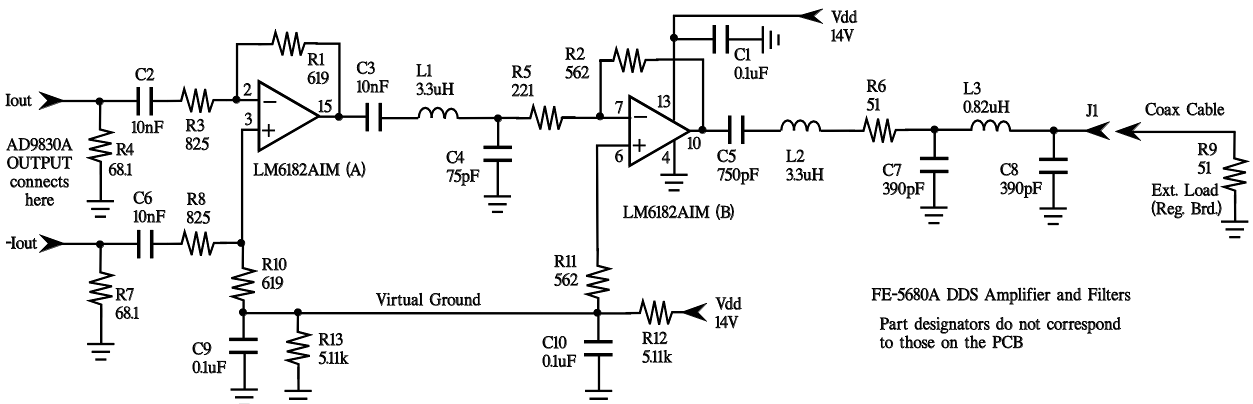


Figure 7.1: Amplifier and filter circuit for the AD-5680A/5650A after Reference [7.1].

Section 7.1: Overview of Modifications

A number of steps can be identified to modify the circuit in Figure 7.1:

1. The first step in modifying the circuit consists of tapping the connector on the DDS board at connector J1 where the buffered/amplified sinewave leaves the board through a thin coax cable. The amplifier output provides significantly better drive capability than the ‘test signal’ on the external AMP connector. The references [7.1-3] suggest removing the 51 Ohm resistor on the regulator board but then caution to replace the 50 Ohm load at the point where an external circuit uses the signal. We leave the 50 Ohm resistor connected in place.

2. The second step consists of placing 20uF multilayer (MLCC) ceramic capacitors across C2, C6, C3 and C5 to increase of bandwidth. These ceramic capacitors presently have the largest capacitance in the same form factor (0805) as the existing capacitors – they can be soldered on top of those already on the PCB. The ceramic capacitors increase the bandwidth to the range 100Hz to 15MHz. The online references suggest using tantalum capacitors in the range 33uF to 100uF. These have significantly larger physical size than the multilayer ceramic capacitors and one must be careful of the polarity. For tantalums, some references recommend connecting the positive terminal of C2 to R4, C6 to R7, C3 to pin 15, C5 to pin 10.

3. The third step consists of paralleling the C9 and C10 capacitors with 20uF SMD ceramic capacitors to improve the filtering of the virtual ground at low frequencies. The low frequency response is slightly improved.

4. A fourth step consists of shorting coil L2 with a piece of wire to improve high frequency signal amplitude. However, doing so significantly emphasizes the signal amplitude near 10MHz.

5. A fifth modification consists of tapping the square wave signal at the comparator and ripple counter on the regulator board. Unfortunately, the square wave is not well formed and requires additional circuitry to square it up (not covered here).

It is possible to almost completely redesign the filters at the output of the AD9830A chip to further decrease the minimum frequency to under 10Hz while maintaining nearly flat frequency response through the 15MHz (see the appendix in [7.1]).

Section 7.2: Coax Connections

The square and sinusoidal waves can be externalized using thin coax cables (RG174). The front aluminum plate (front plate in Figure 7.2, left plate Figure 7.3) will need to be removed and then the regulator board detached from that front plate to install the cable. The new cable will be placed next to the plate and oriented parallel to the existing one for reassembly. The sinusoidal signal from connector J1 can also be externalized by a thin coax cable; this coax does not require the unit to be disassembled since it attaches to the exposed side of the DDS board at connector J1 (Figure 7.3). The free end of the two coax cables can be soldered into other circuits or to male SMA connectors to attach to external equipment or to the female counterparts on the breadboard shown in Figures 5.8 and 5.9 in Chapter 5.

Figure 7.2: The FE-5650A Option 58 as viewed from the connector side. The AMP connector is on the left side and the two screws on the right side hold the front plate in place.

Figure 7.3: A view of the DDS board. The cable on the left attaches to the output signal at connector J1.

A number of references suggest using rigid or semi-rigid coax (50 Ohm) to externalize the signals. The semi-rigid cables flex less than the standard type and thereby reduce the deformation of the coax dielectric and shield, which affect the cable distributed capacitance and inductance, and thereby negatively impact the cable performance for phase, phase noise, and SWR [7.4,1]. However, the semi-rigid coax makes the assembly significantly more unwieldy and difficult to install. Unless the rubidium standard will be used for exacting clock and phase-related applications, consider using the thin RG174.

. As mentioned, two thin coax cables will be soldered into the FE-5650A. We used pigtailed male SMA connectors (i.e., male connectors with a length of coax cable already soldered to it). The pigtailed connectors can be purchased. As an easy approach, simply purchase an RG174 coax cable with male SMA connectors on both ends and cut the cable into two pieces of the proper length. For example, here's one on EBay.com that can be cut.

SMA Male to Male RG174 Coax Pigtail 10ft 3m Extension Cable Adapter

If you plan to use connectors on the coax, be sure to find the SMA type and not the Reverse Polarity SMA (RP-SMA or RSMA). The RP-SMA might not match typical electronic equipment since it reverses the body style of male and female; it was specially developed for WiFi applications.

Unused WiFi or GPS antenna coax cable can also be used. The existing connectors will likely need to be cut off and discarded. If your drawer of old parts doesn't have the thin coax, purchase a small length of RG174, cut it to the proper length and solder a male SMA connector to one end. Here are some Male SMA connectors from amazon

SMA Male, Crimp Straight RF Coax Connector RG188, RG178, RG316, RG174A-U/LMR100A/RFC100A

Amazon Stock Identification Number ASIN: B06XRKCF3C

For some, we used the SMA male connectors for the thicker coax 50Ohm RG8 cables. After soldering the core RF174 wire to the pin, we soldered the braid to the outer conductor, added epoxy into the tube with the pin and then covered the solder joint and part of the RG174 with heat shrink, which also helps with strain relief. The smaller SMA male connectors for the RG174 coax was handled similarly but care was needed to prevent the center core wire from contacting the grounded surrounding metal tube.

For convenience, the first modification consists of adding the coax for the square wave at the MAX913 on the back of the regulator board. Then second, we add the sinewave output from the amplifier and filter circuits. Third, we test the added connections. And then we add the capacitors and test the circuits.

Section 7.3: Externalize the Square Wave

The first step consists of removing the aluminum plate that surrounds the AMP connector and covers the regulator printed circuit board (PCB). Remove the two screws on the front of the aluminum plate (see Figures 7.2 and 7.3). The regulator PCB attaches to this aluminum plate. As a caution, the PCB also has a 'connector tab' made from the same PCB (Figure 7.4) that pushes into a female socket mounted on one of the three remaining PCBs. Carefully pull the aluminum plate away from the other three PCBs while perhaps slightly rocking it to dislodge the tab from the connector. The relevant circuits are located on the side of the PCB facing the aluminum plate. Remove the screws holding the two regulators to the aluminum plate. Be watchful of the two cylindrical rings on the back side of the regulators as they will definitely need to be replaced.

Referring to Figures 7.3 and 7.4, the existing gray cable carries the signal from the DDS board while the black cable is the one hereby added to return the square wave generated at the comparator (Max 913). The Max913 output drives the ripple counters for the 1pps output, but it does not have 50 Ohm output impedance. The chip can continuously source/sink 20mA which corresponds to a load resistance of approximately 250 Ohms for a DC signal. The chip appeared to handle a load resistance of 100 Ohms at 50% duty cycle square wave although sufficiently low frequency would likely cause the maximum ratings to be exceeded.

We carefully soldered a 50 Ohm resistor (small ¼ watt) to the $\bar{Q}$ output on pin 8 (similar to the references) and resolved to maintain the external load resistance above 50 Ohms but primarily above

200 Ohms for a safety margin. The other end of the 50 Ohm resistor was simply soldered to the inner conductor of the added coax cable; a small piece of Kapton tape was place under it for insulation. In retrospect, a small piece of heat-shrink tubing could have been used to insulate the end of the resistor lead (instead of the Kapton).

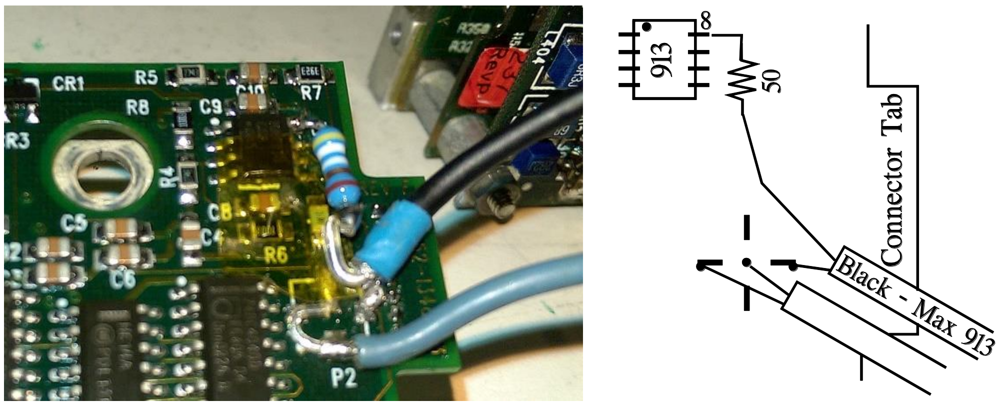


Figure 7.4: The added (black) coax shield attaches to the same connector ground (but opposite side) as the gray coax whereas the inner conductor attaches to a 50 Ohm resistor which in turn connects to the Max 913 pin 8.

The shield wire is soldered to the same connector as the shield for the existing gray cable. The connection pad consists of 4 traces with a center pin (refer to the right side of Figure 7.4). Prior to soldering, use an Ohm meter and check for continuity between ground and the target PCB trace to which the shield will be soldered. Actually check to make sure the selected trace has continuity with the shield of the other coax. There appears to be a thin insulator/protective coating on the PCBs and you might need to melt the solder on the relevant pads prior to testing for continuity. We soldered the shield to the trace nearest to the connector tab. Keep in mind the tab must push into a connector when reassembling. A piece of heat shrink tubing can be used to cover any exposed shielding on the coax. The blue heat shrink in Figure 7.4 looks slightly discolored because we use the tip of the soldering iron to shrink the tubing.

Finally, reassemble the PCB on the aluminum plate being mindful to place the cylindrical rings behind the regulators when inserting the screws through the regulator tabs. Place the just-added thin coax next to the original gray cable and run them both along the milled section of the aluminum plate. Attach the regulator board assembly to the main FE-5650A unit by inserting the PCB connector tab into the mating connector. Make sure the tab is fully engaged – the aluminum plate and PCB will be slightly loose/wobbly until the screws are reinstalled through the front of the plate. CAUTION: make sure the top edge of the plate (with tapped screw holes) is even with the top edge of the plate holding the Rb module (see the edges with the tapped holes in Figure 7.5). All of these edges need to be flush (form a single plane) since the plates will be brought into contact and secured to a METAL case to help remove excess heat. Thermal compound will be used but both of the metal plates need the contact with the underlying case/heat sink.

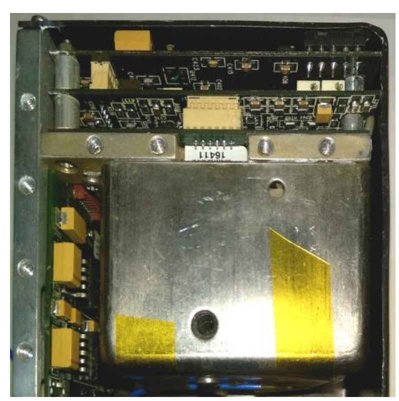


Figure 7.5: Edges/sides with tapped holes.

Section 7.4: Externalize the Sine Wave

Attaching a cable to the output of the AD9830A DDS is similar but easier than the one for the MAX913 comparator (square wave). Now the board can be easily accessed without any disassembly. The board of interest can be seen in Figure 7.3. The DDS signal leaves the board at connector J1 (Figures 7.6 Left). We tap into the signal at connector J1. The middle panel of Figure 7.6 shows J1 with the center pin and the surrounding pins/tabs. The J1 pin on the left connects to C8 and L3 in Figure 7.1 and so we know it is also routed to the center pin of the J1 connector. The center wire in a RG174 coax cable should be soldered to this left side pin as shown in the middle portion of Figure 7.6. We did not need to remove the coax connected to J1 to perform the soldering. Use an Ohm meter to determine which of the surrounding pins of J1 connect to the shield of the gray cable or equivalently to ground (such as the aluminum plates). Keep in mind the pins might have a thin insulting coating and so it might be helpful to melt the solder prior to testing for continuity. We found the lower pin of J1 should be soldered to the shield of the coax cable routed to the outside world. The right hand panel of Figure 7.6 shows the original gray coax, the newly soldered black coax to connector J1, and the black coax coming from the comparator on the regulator board. We wrapped a couple turns of wire around the two black coax cables and anchored that wire under the nut on the board. As a note, it would not be difficult to install a semi-rigid coax here for the better performance but make sure to adequately anchor the coax.

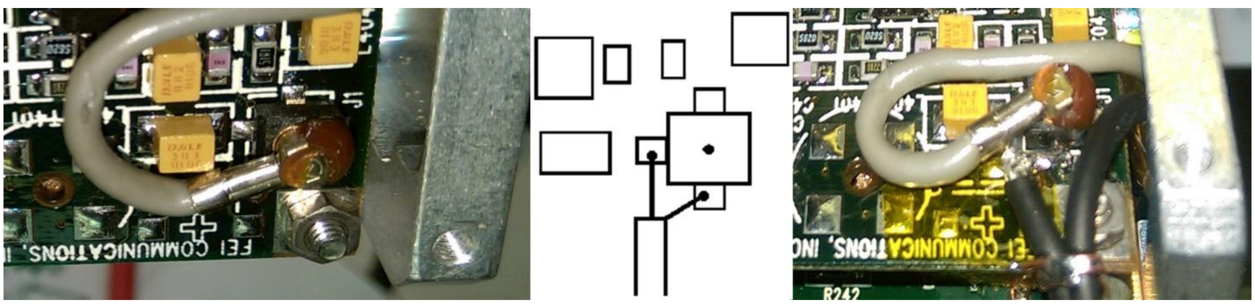


Figure 7.6: Left: The connector J1 with its pins. Middle: Solder the center wire of a thin coax cable to the left pin (signal) and the shield to the lower pin (ground). Right: The original gray wire carries the signal to the Max913 comparator and the right hand black wire brings the square wave from the comparator to the outside world. The left hand black wire brings the AD9830A signal to the outside world from J1.

Section 7.5: Sine and Square Wave Tests

The tests are easy. Attach the two new coax cables to an oscilloscope or a frequency counter and power the unit as described in Chapter 5. The measurement should show the default frequency of 8.388608MHz (or whatever appropriate for your unit). The output from the amplifier and filters should produce a sinewave and should be capable of driving a 50 Ohm load. The square wave from the Max913 will likely appear more similar to a sinewave that a square wave at the 8MHz frequency. The chip output can continuously source/sink 20mA which can drive a total load resistance of approximately 200 Ohms at 5V (i.e., an external 150 Ohms when including the internal 50 Ohm resistor soldered at the Max913). The Max 913 did handle an exterrnal 50 Ohm load (i.e., 100 Ohms when including the internal 50 Ohms) for frequency larger than about 100Hz (50% duty cycle). Too low of frequency would likely cause the maximum ratings to be exceeded.

Section 7.6: Capacitors and a Coil

One of the references [7.1] shows further modifications that produce wide bandwidth with flat response. The simplest modification to the DDS board to increase the bandwidth requires the paralleling of four to six capacitors by larger value capacitors and the optional shorting of a coil. Figures 7.7 shows the location of capacitors C10, C6, C2, C9, C3, C5 to be paralleled and coil L2 to be shorted where the part designators correspond to that in Figure 7.1. The traces help identify the correct devices. [7.1]

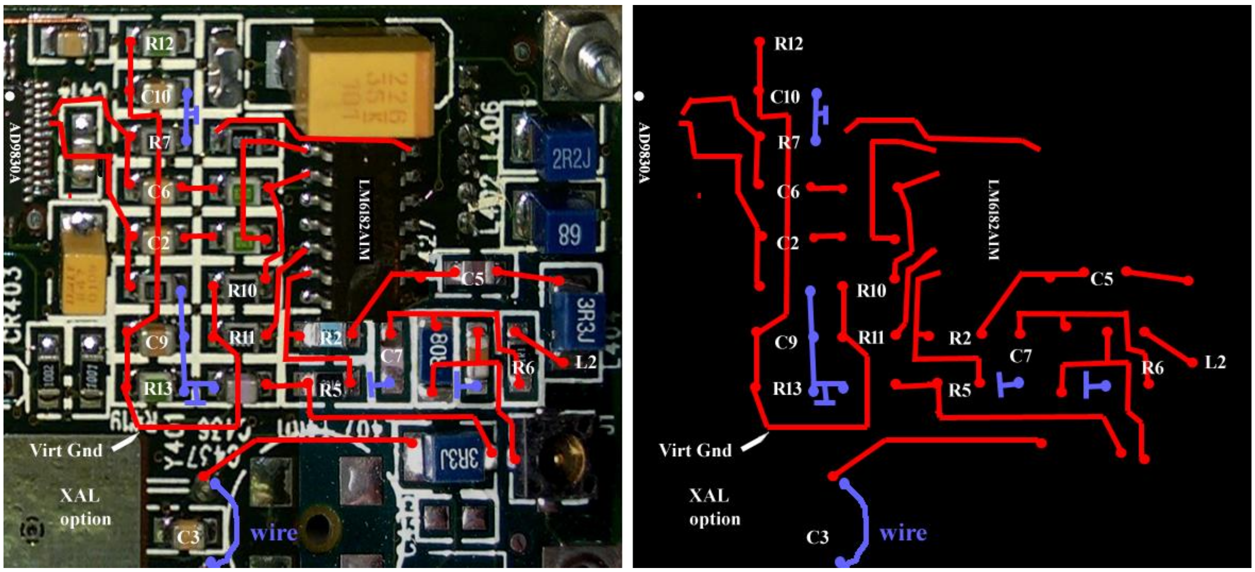


Figure 7.7: Left: Relevant DDS board traces. The part designators correspond to those in Figure 7.1. Right: The designators and traces without the obscuration by the PCB components. The small straight line segments refer to a ground connection. Similar to Ref. [7.2].

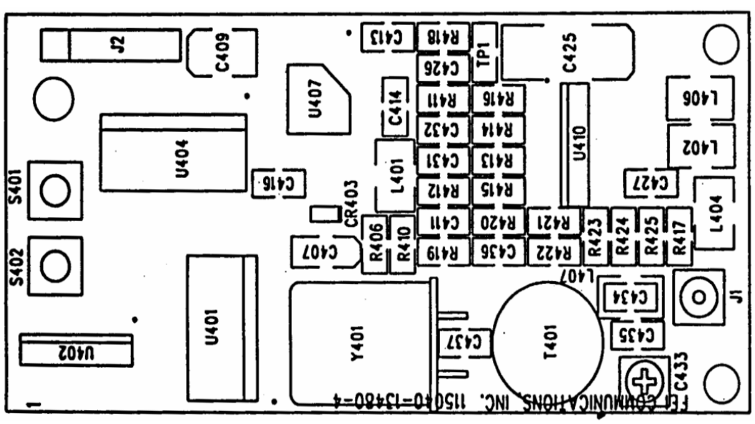


Figure 7.8: The manufacturer's part designators for the DDS board from [7.1]. If the Y401 crystal is present, it might need to be removed and the pads shorted together at the board.

As previously mentioned in Section 7.1, some references use tantalum capacitors for coupling and filtering owing to their low series resistance (ESR) and large values in small packages. Some references use up to 100uF. However the FE-5650A DDS board incorporates the type 0805 Surface Mount Device (SMD) capacitors for coupling the signals and filtering the virtual ground. We found the 0805 Multi-Layer Ceramic (MLC) SMD capacitors of 22uF 25V to be suitable for increasing bandwidth.

Electronic distributors such as Digikey.com and Mouser.com have a variety of these type of capacitors (and appear to sell out fast) such as

Murata 22uF 25V MLC SMD 0805 Mfg # GRM21BR61E226ME44L, Digikey # 490-10749-2-ND
Murata 22uF 25V MLC SMD 0805 Mfg # GRT21BR61E226ME13L, Digikey # 490-12389-1-ND

These 22uF 0805 capacitors can be soldered to the top of the existing capacitors in a parallel topology without changing the overall size of the PCB - the modified PCB still fits within the original black metal cover. The MLCs don't have polarity but they are susceptible to internal stresses that can damage the layers during soldering. Use minimal soldering temperature and duration. Definitely use an Ohm/resistance meter to check for shorts after soldering – the meter should momentarily read low resistance and then show increasing resistance as the capacitor charges.

For those readers who have not previously soldered Surface Mount Devices (SMD), it might be helpful to practice with an inexpensive kit such as the following available at Amazon for under $10.

Gikfun SMT SMD Soldering Practice DIY Kit EK1752 ASIN: B01C5N2XPO
Gikfun DIY SMD SMT Soldering Skill Training Board Ek7028 ASIN: B00VWB8F8K

The modifications do not require any special soldering iron beyond one with a fine tip and temperature control such as the following.

Elenco temperature controlled soldering station Amazon stock # B001SH0TC8

Unlike the present project, those that require soldering many SMDs on a blank PCB should use paste solder and a solder reflow oven (or less pricey, a modified toaster oven). The first of the mentioned Gikfun practice kits makes for good practice in using the oven. Repairs of SMD PCBs also benefit from the use of a hot-air rework station. Ovens *cannot* be used for the modifications discussed here.

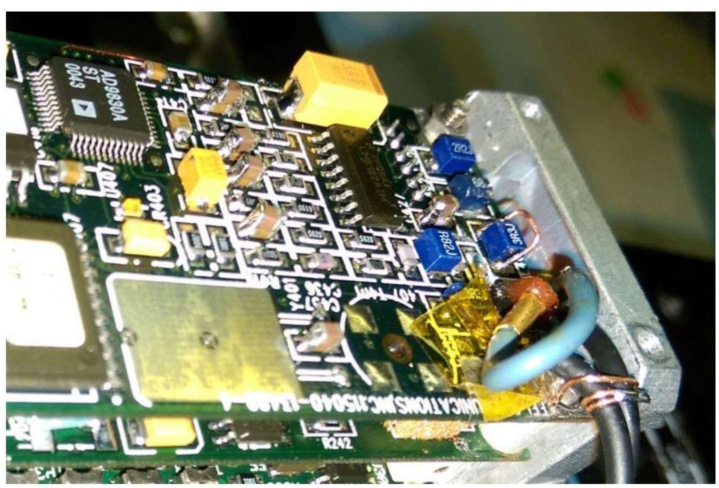


Figure 7.9: A close look shows the capacitors added to the top of existing capacitors – the tall tan bumps. Notice the wire added around coil L2 on the right hand side of the circuit board.

It will be necessary to position, hold and solder the SMD capacitors on top of the existing ones – not as easy as soldering directly to a flat board. Basically, one end of the new capacitor needs to be lightly tacked with solder to one side of the existing capacitor, then the other side can be fully soldered and then the tacked side can be fully soldered. We have used a number of methods but tweezers, fingers or tapes worked fine: (i) Stainless steel tweezers can be used to hold the TINY SMDs during soldering but finger pressure must be maintained on the tweezers to retain grip on the SMD. The tweezers with curved tips are especially helpful. (ii) Tweezers with normally closed tips (a.k.a., pinchers? jaws?) – the tips remain closed until the tweezer sides are squeezed. These tweezers eliminate nervous hands flinging the poor little SMD across the room into never-never land. (iii) A mechanical device can be easily constructed to hold vacuum tweezers that can position and hold the SMD. The vacuum tweezers consists of a hollow tube with a hollow tip both of which connect to a small vacuum source to hold the SMD to the tip. Be sure to have several tips since they will suck up solder and clog. (iv) Kapton tape or even ordinary transparent tape can be used to position and hold the SMD. (v) A human finger works well too but keep it way from the iron!

We usually first add a light coating of solder on the new SMD capacitor prior to mounting to the underlying existing capacitor on the board; for this process, the capacitor can be easily held by the normally-closed tweezers. We add some solder to the existing capacitor ends too. Once the new capacitor is positioned and held in place, lightly tack one end by momentarily applying iron heat. Once it's tacked, the restraints on the SMD can be relaxed and the other side can be fully soldered. Then the tacked side can be fully soldered. Figure 7.9 shows the SMD capacitors soldered on top of the existing capacitors on the PCB.

The capacitors should be soldered in place along with the wire across coil L2. The (blue) coil L2 can be seen on the right hand side of Figure 7.9 just above the gray coax. The soldered jumper wire can be seen to surround L2.

Prior to testing, the initial setup in Chapter 5 should be completed. The easiest method for testing consists of using the RS232 port and a USB-RS232 adapter for a computer.

Section 7.7: Modification Tests

The tests of the modified FE5650A require a method to determine the drive capability of the sine and square waves and also to verify the frequency range has been properly expanded. We leave the frequency calculations to the next short chapter and instead select a few frequencies in the 1MHz to 15MHz range. To make the tests, it is necessary to either use a frequency counter and a volt meter suitable for that frequency range (see the circuit below) or else an oscilloscope with the capability to change input impedance. Many oscilloscopes only have the high impedance termination (1M Ohm) so it will be necessary to obtain a 50 Ohm feedthrough terminator such as the Amazon 50 Ohm BNC Feedthrough Terminator with Amazon Stock Identification Number: B07G566JC7. As an alternative, use a 50 ohm terminator with a BNC T connector, or even a 50 Ohm resistor at the input of the oscilloscope. Most likely, if the oscilloscope shows any kind of output from the modified FE5650A, the unit is properly functioning.

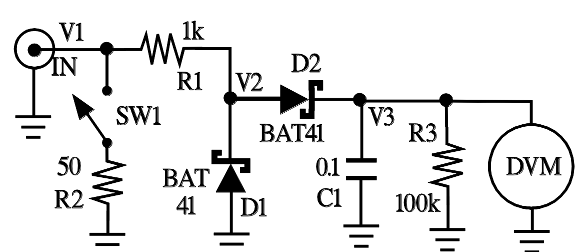


Figure 7.10: AC voltmeter for signals 1-15MHz using a DC Digital Volt Meter (DVM).

If an oscilloscope is not available, then a simple circuit (Figure 7.10) can be constructed to determine the drive capability of the modified FE 5650A. The circuit is basically an AC volt meter suitable for 8MHz. The input BNC of the circuit connects to the FE5650A sinewave output. If desired, the connection to the center terminal of the BNC can be removed and a 0.1uF capacitor inserted in its place in order to remove a DC component that might offset the meter reading. The switch SW1 couples the 50

Ohm termination resistor. In order to develop a DC voltage across the DVM, the circuit uses diode D1 to conduct current to ground for negative voltages V2 without significantly affecting the current for positive voltage V2. Typically, diode D2 would be used with capacitor C1 to detect and hold the peak; however, R3 slowly discharges C1 for the voltage V3 to follow the input voltage. Diode D2 helps eliminate the negative going voltages V2 discharging C1. Capacitor C1 removes the high frequency components and charges to a DC level as controlled by the current drain through R3 and the high input impedance of the DVM. Diodes D1 and D2 should be Schottky diodes for the lowest forward voltage drop but the circuit will work with Germanium or Silicon diodes. Select small signal diodes with low junction capacitance.

The modified FE-5650A can be tested (unit #2 in Table 6.4). As in Chapters 5 and 6, apply the Amp connector and the serial port connector. Power up the unit and the PC and run the terminal software. Use either the oscilloscope or the circuit in Figure 7.10, and use short coax cables to keep the capacitance to a minimum. The FE5650A should start at the default frequency (8.388608MHz for our units). For the oscilloscope, verify that the output drops by about half when the termination is switched from 1M Ohm to 50 Ohm. For the case of the circuit, the peak-to-peak voltage read on the DVM should also drop by about a factor of two when the 50 Ohms is connected.

Attach the sine output to a frequency counter. Next attach the USB-RS232 adapter and run the terminal software. Send the following commands by simply typing the Fcode as text into the terminal and remember to include the carriage return.

Fcode	Frequency
F=32F0AD80	10MHz
F=0518115A	1MHz
F=00826823	100kHz
F=000D0A6A	10kHz
F=00012DD7	1kHz
F=00002162	100Hz

The listed frequency should show on the frequency counter. Keep in mind that the true reference frequency has not been used to list the Fcodes and so the Frequency might not be exact. We discuss the calculation in Chapter 8 and calibrations in Chapter 9.

Section 7.8: Response and Spectrum

We jump a head a little and provide the response and the spectrum of a modified unit (unit #2 in Table 6.4).

Figure 7.11 shows the FE5650A output voltage vs. the output frequency for the following several modifications:

1. Replace only the coupling capacitors C2, C3, C5, C6.
2. Replace the coupling capacitors and the virtual ground filtering capacitors C9 and C10
3. Replace the coupling and filtering capacitors and short coil L2.

Figure 7.11 shows the response results for the modified FE-5650A output (unit #2 in Table 6.4) from the AD9830A through the amplifier and filter circuits. The signal is coupled through a 50 Ohm 3ft long 50Ohm coax cable (roughly 100pF total) into an oscilloscope set for 1MHz input impedance – coax cables longer than 2 to 3 feet cause distortion of the waveform when not properly terminated (50 Ohms). As shown in Figure 6.7 (Section 6.4) for unit #3 (Table 6.4), the unmodified FE-5650A has a range of roughly 6MHz -14 MHz – this response would be a relatively narrow bump at the right side of the plot in Figure 7.11.

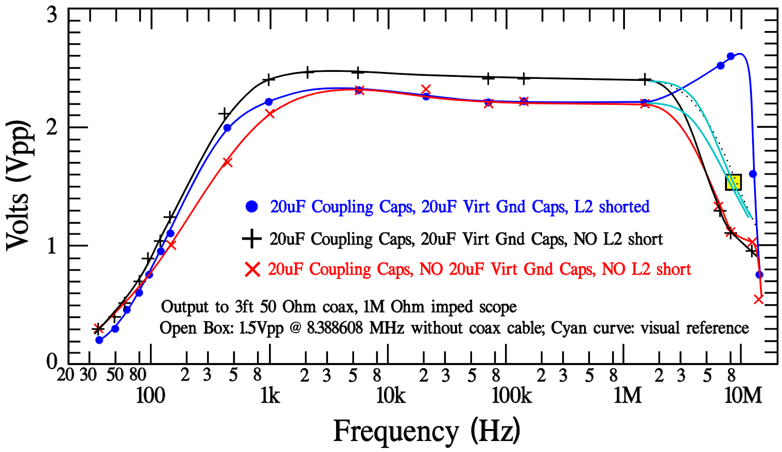


Figure 7.11: A semi-log plot of the frequency response for the FE-5650A using several different configurations: the 'X' corresponds to only modifying the four coupling capacitors C2,3,5,6; the '+' corresponds to modifying all six capacitors but not L2; and the '.' refers to changing all 6 capacitors and shorting L2. The FE-5650A was attached to the 1 MOhm input of an oscilloscope through a 3 foot long coax cable (roughly 32 pF/foot). The square box and curves show the approximate response without the coax. Shorting L2 causes the response to peak.

It can be seen that changing the four capacitors of C2,3,5,6 has the greatest effect on the response. Shorting coil L2 greatly increases the response at 10MHz presumably due to a pole previously lurking in the background. The three curves should be normalized to one for a reliable comparison of the frequency response, but the figure does provide an estimate of the voltages involved. The square box and curves near 8MHz shows the approximate response for the case of no coax cable as measured by a high impedance scope probe.

We next used an HP8568B RF Spectrum Analyzer (RFSA) to determine the output spectral components associated with a FE5650A (unit #2 in Table 6.4) modified by shorting L2 and increasing the six capacitors to 20uF. The measurement used the voltage divider in Figure 6.8 in Chapter 3 and a three foot long 50 Ohm coax cable. The FE5650A output impedance and the RFSA input impedance are 50 Ohms. The RFSA was set for 10dB attenuation, 3kHz Res BW, 300Hz Vid BW and 10dB per vertical division with the reference voltage at the top of 0dBm.

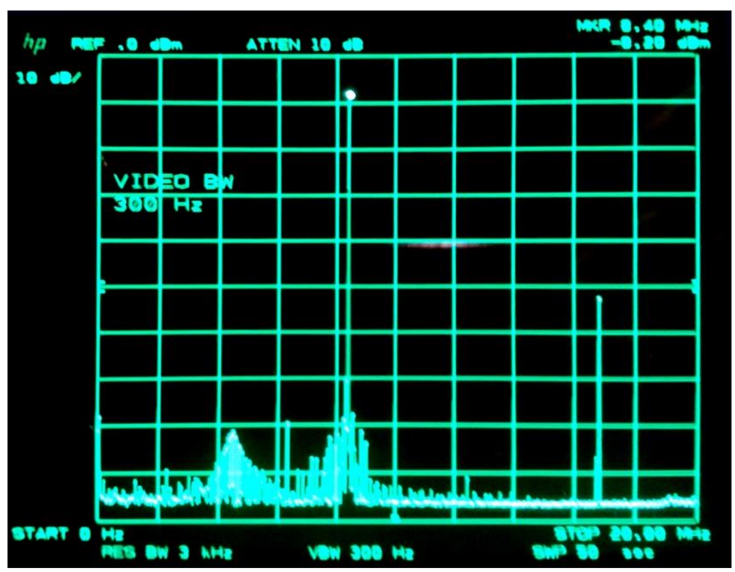


Figure 7.12: Output spectrum of the modified FE5650A set for 8.388+ MHz. Vert 10db/div, Hor 20MHz; RBW 3kHz; VBW 300Hz

Figure 7.12 shows the measured spectral power density for the modified unit. The plot shows the power density dBm in the output signal from 0 to 20MHz. Notice the noise near the floor. The center peak at 8.39MHz is roughly 44dB ($2.5*10^4$) larger than the harmonic at 16.78MHz compared; this ratio can be compared with the 60 dB for Figure 6.9 for the unmodified unit.

Similarly the spurious components approximately 70dB below the the 8.388+ spike appear more pronounced for Figure 7.12 than for Figure 6.9.

Section 7.9: References

[7.1] DDS board: original board schematic and simulations for mods for FE-5680A but DDS same. Includes using serial port, circuit layout on pcb. Very complete.
Matthias Bopp, D.C.Johnson, Deflef, “A precise reference frequency not only for your ham radio station” Rev 1.0 (2013)
http://www.redrok.com/Oscillator_FE-5680A_precise-reference-frequency-rev-1_0.pdf
Vers 0.4 with some Basic programing notes
https://www.fetaudio.com/wp-content/uploads/2009/10/FE-5680A-modifications.pdf

[7.2] Modifying the FE-5650A including power supply requirements, microcontroller connections, DDS board functions. Manufacturer’s part designators for the DDS board, Technical Manual TM0107, Rubidium Frequency Standard, Model FE_5650A Series, Operation and Maintenance Instructions. Consider making a donation for providing valuable information.
Mark Sims, “FEI Rubidium Oscillators comment,”
http://www.ko4bb.com/doku2015/doku.php?id=precision_timing:rubidium_oscillators
https://www.mail-archive.com/time-nuts@febo.com/msg13486.html

[7.3] Skip Withrow, “One sure way to kill your FE-5680A and FE-5650A,” NARKIVE Mailing List Archive, Time-Nuts@febo.com
https://time-nuts.febo.narkive.com/Kjd30sgK/one-sure-way-to-kill-your-fe-5680a-or-fe-5650a

[7.4] Coax sensitive for size changes
Standard Wire & Cable Co., Coaxial Cable: Theory and Applications
http://www.standard-wire.com/coax_cable_theory_and_application.html

Chapter 8: Errors and the Reference Frequency

The output frequency of the FE-5650A can be controlled via serial port by transmitting a 32 bit number in the form of eight HEX digits 0-F written as text. The embedded PIC microcontroller (uC) receives this Fcode (a.k.a., tuning word) and transfers 16 bits at a time to the AD9830 Direct Digital Synthesizer DDS to program the desired output frequency. The PIC microcontroller (uC) uses the reference frequency stored in the PIC nonvolatile memory of the microcontroller to calculate the output frequency based on the incoming Fcode. The stored reference frequency is denoted Rstored. Therefore, for the user to send the correct Fcode to the FE5650A to generate the desired output frequency Fout, the user must know Rstored and any related algorithm (i.e., programmed formula) the PIC uses to calculate the true reference frequency R.

The chapter discusses several different methods of calculating the true reference frequency R. The first relates the stored reference frequency Rstored to the true one as best as possible without discussing the actual calibration related measurements. After some calculational examples, the chapter reviews the required commands to obtain the stored reference frequency. In some cases the reference frequency might not be obtained or in some instances, the true reference frequency must be more accurately calculated, and hence the true reference frequency must be obtained from measurements. The chapter ends by discussing somewhat obvious conclusions regarding the extraction of R from the measurements. However, the next chapter discusses the actual measurement methods for extracting the true reference frequency.

The Chapter performs calculations with base 16 numbers (i.e., HEX) and so Appendix 1 has been included to provide an introduction to both the HEX numbers and to the Microsoft Windows 10 calculator (programmer and scientific modes) – the calculations require a programmer's calculator with lots of digits. The best that we have found so far is the Windows 10 Calculator from Microsoft. There are a few Scientific and Programmer Calculator apps for the smart phone. The one named 'Calculator2' works as well as the Windows calculator but perhaps more convenient on a smart phone. However, none of these calculators operating in the programmer mode (for Hex calculations) handle fractional HEX digits which means the user must be careful when a zero appears in the display after a calculation – the fraction will be truncated or rounded to the nearest non-fraction digit. For the FE5650A settings, there will be a lot of converting back and forth between the base 10 (decimal) and the base 16 (HEX).

Section 8.1: Reminder of the DDS Relations and Associated Uncertainties

As previously discussed in Section 3.2 and Topics 4.1.2-3, the relation among the output frequency Fout, the tuning word Fcode and the (true) reference frequency R has the form

$$\mathrm{F_{out}} = \mathrm{F_{code}}\frac{\mathrm{R}}{2^{32}} \qquad \text{or} \qquad \mathrm{F_{code}} = \mathrm{F_{out}}\frac{2^{32}}{R} \qquad \text{or} \qquad R = 2^{32}\frac{\mathrm{F_{out}}}{\mathrm{F_{code}}} \qquad (8.1a,b,c)$$

where $2^{32} = 4{,}294{,}967{,}296$, and F_{code} is the eight-hex digit number (32 bit, 4 bytes) sent to the AD9830A, and where R symbolizes the *true* reference used by the AD9830 DDS chip with the approximate value of 50.2550255+ MHz

Several types of error have relevance with respect to the rubidium standard. An Allen Deviation Analysis, according to the literature, generally shows the rubidium standard has an inherent accuracy of approximately $2*10^{-11}$ which, at 10MHz, produces $2*10^{-11}$ * 10,000,000 = 0.0002 Hz. For simplicity, we approximate this error as 0.001 Hz. Notice this error is a percentage of the operating frequency which indicates the error (Hz) scales with the frequency.

As discussed in previous secitions, Equations 8.1 provide relations for small changes in the three quantities of the Fcode, the reference frequency R and the output frequency Fout, symbolized by ΔF_{out}, ΔR, and ΔF_{out}, respectively.

$$\Delta F_{out} = \Delta F_{code} \frac{R}{2^{32}} + \Delta R \frac{F_{code}}{2^{32}} \qquad (8.2)$$

Consider first the DDS digitization/quantization error associated with ΔF_{code}. We are not interested for the moment in the uncertainty in the reference frequency R and so assume $\Delta R = 0$ which means

$$\Delta F_{out} = \Delta F_{code} \frac{R}{2^{32}} \qquad (8.3a)$$

The Fcode changes at minimum by a single step and hence we assign the Fcode uncertainty as $\Delta F_{code} = 1$. Assuming the reference frequency is approximately 50255055, we find

$$\text{Quantization Uncertainty: } \Delta F_{out} = 0.0117 \text{ Hz} \qquad (8.3b)$$

The quantization uncertainty is an absolute number and is not a fixed *percentage* of the operating frequency; that is, the error does not scale with frequency.

The determination of the true reference frequency R produces another source of uncertainty expressed through ΔR. We are not interested in the quantization error now and set $\Delta F_{code} = 0$. Equation 8.2 becomes

$$\Delta F_{out} = \Delta R \frac{F_{code}}{2^{32}} \qquad (8.4a)$$

The uncertainty in the output frequency (at 10MHz) with Fcode ~ 32F0AB00 provides

$$\Delta F_{out} = 0.2\, \Delta R \qquad \text{for 10MHz} \qquad (8.4b)$$

So if the reference frequency is known to within $\Delta R = \pm 1$ then the output frequency will be known to within ΔFout = ±0.2 Hz. The error actually scales with the frequency because of the term Fcode in Equation 8.4a. So at 5MHz, one would expect 0.1 ΔR. Apparently the uncertainty in the reference frequency needs to be on the order of 0.05Hz or less to bring the corresponding uncertainty of the output frequency on par with that due to the digitization of 0.01.

Section 8.2: A Weak Relation between the Stored and True Reference Frequencies

In order to calculate the proper Fcode to produce a desired output frequency Fout in Equations 8.1, one must know the value of the true reference frequency R. In principle, a value for R can be measured from within the unit by using a suitable frequency counter. However, the FE5650A microcontroller stores a reference frequency (Rstored) in non-volatile memory. The question naturally arises as to how well Rstored approximates R. For a good approximation or for the case that R can be formulaically related to Rstored across all manufactured FE5650A units, there would be no need to measure R for each unit. Another method of extracting a value for R consists of performing an accurate measurement of Fout for a given Fcode and then using Equation 8.1c. This measurement method would be most accurate and it would be required for the case of the FE5650A encrypting the response to the status request command 'S' – the method will be explored later.

If the status command 'S' returns intelligible characters, then the value of Rstored can be ascertained along with the Fcode that produces a default output frequency Fout which can be measured using an accurate frequency counter. One internet reference [8.2] found the stored value differs from the true value and it varies among units. The reference goes on to suggests forming the ratio of the true reference frequency R to the stored reference frequency Rstored, and denoting it by k_p.

$$R = k_p R_{stored} \tag{8.5a}$$

The reference tested a number of FE5650A units and determined the average value of k_p, denoted by $\langle k_p \rangle$, had the value $\langle k_p \rangle$=1.000,000,002,150. The idea was to obtain a reasonably accurate prediction for the true-reference frequency for a given unit by retrieving the stored reference frequency from that unit using the status inquiry command S, and then multiply by $\langle k_p \rangle$. Then the required Fcode could be determined to produce a desired output frequency Fout using Equation 8.1b. The ensuing chapter shows that we calibrated four FE5650A units and found

$$\langle k_p \rangle = 1.000{,}000{,}003{,}755$$

In fact, we found the values for k_p of

1.000,000,003,675   1.000,000,004,852   1.000,000,005,454   1.000,000,001,039

which differ from both averages by roughly

$$\Delta k_p = \pm 0.000{,}000{,}002$$

To what extent does the uncertainty in k_p make it unreasonable to use the above average value of k_p and the retrieved value Rstored in any given unit to determine output frequencies? Should the units be calibrated at all? The uncertainty in k_p causes error in the predicted true reference frequency R through Equation 8.5a as

$$\Delta R = R_{stored} \Delta k_p \tag{8.5b}$$

Using a value of Rstored of very roughly 50255056, we find the error in R as

$$\Delta R = \pm 0.10 \text{ Hz}$$

which, from Equation 8.4b, gives the error in output frequency of roughly

$$\Delta F_{out} = \pm 0.02 \text{ Hz}$$

The error isn't so much different from the quantization error 0.012 in Equation 8.3b. *If your application can withstand the ±0.02 Hz error level, then no need to calibrate.*

Section 8.3: Example Calculations for Fout and Fcode

The example constant of proportionality k=1.000,000,002,150 between the true and stored value of the reference frequency provides a starting point for programming the FE5650A. As previously discussed, the status inquiry of the FE5650A via the 'S' command returns R=Rstore and F=Fcode data. Knowing Rstored means R can be approximated through Equation 8.2 and in turn Fcode can be calculated in order to set the output frequency through Equation 8.1b. The calculated Fcode can be transmitted from a computer or microcontroller uC to the FE565A using the RS232 serial port (RS232). For now, to discuss the method of calculating the frequency, assume that the FE-5650A physics module produces 50.25505+ MHz as the true reference R. As a side comment, some online sources write F for Fcode and FF for Fout.

Examples using the true reference frequency R=50255056.353937 Hz. See Appendix 1 for HEX math and suitable apps and calculators to handle the digits.

Example 8.1: Show Fcode = 2ABB503E produces Fout = 8.388608 MHz

Equation 8.1a is probably easiest to calculate by converting Fcode to the decimal form. The Fcode digits can be entered into the Microsoft Windows 10 'Programmer' calculator when the HEX option has been selected. The DEC window provides the following conversion

2ABB503E hex ==>> 716 918 846 decimal

Next, switch to the 'scientific' calculator and recall 2^{32}= 4 294 967 296 decimal to calculate

Fout = Fcode * (Ref / 2^{32}) = 716 918 846 * (50255056.353937 / 4 294 967 296)

So Fout = 716 918 846 * 0.16692068 = 8.388608 MHz

Example 8.2: Show Fcode = 32F0AB00 produces Fout = 10.000000 MHz

Similarly, converting Fcode to decimal provides

32F0AB00 Hex ==>> 766 857 278 decimal

Equation 8.1a becomes

Fout = Fcode * (Ref / 2^{32}) = 766 857 278 * 0.16692068 = 10.000000 MHz

Example 8.3: Find the Fcode to produce Fout = 1MHz. Assume the stored reference frequency is
Rstored = 50,255,056.245889 .

Use the 'scientific' mode of the Windows calculator and Equation 8.5 to first convert Rstored to the approximate true value R = 50,255,056.353937. Continue with the scientific mode and use Equation 8.1b to find Fcode as

$$\mathrm{F_{code}} = \mathrm{F_{out}} \frac{2^{32}}{R} = 1{,}000{,}000 \frac{4{,}294{,}967{,}296}{50{,}255{,}056.353937} = 85{,}463{,}386.326$$

Now here's an issue, the Fcode must be converted to Hex which means the decimal places need to round to the closest integer. In this case, use Fcode = 85,463,386. Next, switch the Windows calculator to 'programmer' mode, click on DEC at the left side and enter Fcode. The HEX display will show

Fcode=518115A

which is OK as an answer but the FE5650A requires **8 Hex digits** to be transferred. The command to enter into the FE5650A is the text F=0518115A<cr>. Notice the zero has been added.

It should be mentioned that Version 1 of the PC-FE5650A Interface software for the PC described in Appendix 9 can also perform the Fcode calculation. To do so, double click the 'R=' textbox so the background color changes to white. Enter the true value of R into that textbox. Next, select and delete the text next to the send button at the bottom and enter 1000000 (no commas!). Next increase and then decrease one of the spinner values by one using the up-down arrows so that the software will make the calculation. Notice the actual error in the output frequency will be -0.004 Hz.

Example 8.4: Estimate the various errors associated with Example 8.3 above. Assume the stored reference frequency is

Rstored = 50,255,056.245889

and the possible error in Rstored is

$$\Delta Rstored = \pm 0.000{,}000{,}000{,}3$$

Now we estimate the various errors. First the inherent uncertainty based on the Allan Deviation $2*10^{-11}$ will be

$$2*10^{-11} * 1{,}000{,}000 = 0.002 \text{ Hz}$$

The uncertainty in Fout due to the digitization step can be found from Equation 8.3a

$$\Delta F_{out} = \Delta F_{code} \frac{R}{2^{32}} = 1 * \frac{50{,}255{,}056.353937}{4{,}294{,}967{,}296} = 0.0117 \sim 0.012 \text{ Hz}$$

which agrees with the previously computed value in Equation 8.3b. This error does not scale with frequency. Next, compute the uncertainty in Fout due to the uncertainty in R, namely ΔR, as follows. First recall the uncertainty in the constant of proportionality k_p in Equation 8.5 was assumed to be

$$\Delta k_p = \pm 0.000{,}000{,}003$$

Therefore the uncertainty in the true reference frequency derived from the stored value will be

$$\Delta R = \Delta k_p \, R_{stored} = \pm 0.000{,}000{,}003 * 50{,}255{,}056.245889 = \pm 0.15$$

The uncertainty in the output frequency due to that of the reference can be found by using Equation 8.4b.

$$\Delta F_{out} = \Delta R * 0.20 = \pm 0.03$$

Normally, the digitization error dominates but we found the actual digitization error for the FE5650A Interface software is -0.004. So combining the digitization and the reference uncertainties, one can expect the error in the output frequency to range

$$\Delta F_{out} = -0.004 \pm 0.03 => -0.034 \;\; to \;\; +0.037 \text{ Hz}$$

As detailed in Section 6.5, some FE5650A units appear to return encrypted ASCII code in response to the status enquiry 'S' (although the problem might be the baud setting). If the Status request does not return intelligible information, then it is not possible to know Rstored, and hence R, to complete the calculations exemplified above. Generally, using $R \cong 50.255056$ MHz (assuming $\Delta R = \pm 1$) should be sufficient to determine Fout to within approximately $\Delta F_{out} = \pm 0.2$ from Equation 8.4b). The next chapter will show several methods of determining an appropriate value for the reference frequency R.

Section 8.4: Programming

To program the FE5650A frequency using a Windows-based computer, install one of the terminal communications programs mentioned in Chapter 6 or the FE5650A Interface software described in Appendix 9, and attach a USB-RS232 adapter between the FE5650A and the computer as described in Chapter 6. Recall that the terminal app should be set to 9600N81NO, or 9950N81NO (etc.) which means the serial port should be set to 9600 or 9950 buad, no parity, 8 data bits, 1 stop bit, no handshaking. Recall some FE5650A units scramble the response to the status enquiry at 9600 baud but appear to descramble it at 9950. Apparently, the baud depends on the manufacturer's choices for the unit under test.

To find the stored reference frequency Rstored and the Fcode used to set the current output frequency Fout, send the status 'S' command/enquiry using the terminal software as

S<cr>

where <cr> refers to the carriage return/enter character. Either the terminal app should be setup to automatically send the ASCII code for carriage return/enter <cr> or the ASCII code can be included by hand. For example, if the terminal program is set to send ASCII in HEX but without the carriage return, send the following

53 0D

Table 6.2 in Chapter 6 has the various ASCII codes and shows 53 is the ASCII HEX code for S and 0D is the ASCII HEX code for <cr>. Usually, the terminal programs will have the option to automatically append the <cr> ASCII code.

After sending the Status enquiry, the FE-5650A should respond with

R=5025505… F=2ABB5 …

The 'R=' indicates the stored reference frequency is 5025505… and the 'F=' indicates the Fcode which sets the default frequency. Start with this stored R value and use Equation 8.5 to find an approximate true Ref value. However, if the FE5650A is known to produce an exact default frequency such as Fout=8.38860800MHz, then it would be better to find a true R from Equation 8.1c

$$R = \frac{2^{32}\mathrm{F}_{\mathrm{out}}}{\mathrm{F}_{\mathrm{code}}} \qquad \text{(8.1c repeat)}$$

by substituting the known value for Fout and the returned Fcode.

To set the output frequency Fout, the Fcode must be calculated for the desired output frequency, and then the Fcode must be sent to the FE5650A. Suppose the desired output frequency is Fout=10,000,000.00 and, as in the above examples, the true reference frequency is R=50255056.353937 Hz, then the corresponding tuning code is calculated to be Fcode = 32F0AB00 using Equation 8.1b. Enter the following text into the terminal program

F = 32F0AB00<cr>

where, as usual, <cr> provides a reminder that a carriage return needs to be included. A frequency counter or oscilloscope connected to the output should be able to provide some verification of the 10MHz. Sending the status enquiry should return the just-sent Fcode and the unchanged stored reference frequency Rstored.

For some units, the FE-5650A when queried by the Status command 'S', returns what appears to be encrypted ASCII code as previously discussed in Chapter 6. Assume the baud of 9950 was not found to correct the problem and so neither the stored reference frequency nor the Fcode for the default frequency would be known. As a check, we interfaced one of the units at 9600baud and attempted to set Fout = 10.000000MHz using the examples above with Fcode=32F0AB00 and Ref=50255056.353937 Hz. The following command was sent to the FE-5650A

F=32F0AB00 <cr>

An inexpensive Sinometer VC2000 frequency counter displayed 9.999 993 MHz (after calibrating to the 0.5 Hz level). To correct the deficit of 7 Hz between the desired and actual output frequencies, the results in 8.3a was used to find

$$\Delta F_{out} = \Delta F_{code}\frac{R}{2^{32}} = \Delta F_{code} * \ 0.012$$

Solving for ΔF_{code} we find

$$\Delta F_{code} = 83\ \Delta F_{out}$$

To increase the FE-5650A frequency by 7, the Fcode must be increased as suggested by Equation 8.4b by the amount 83*7 = 581 = (0x)0245, where (0x) means HEX. The proper Fcode to send would then be on the order of Fcode = 32F0AB00 + 0245 = 32F0AD45. Slight changes can be made to find a better Fcode. Once Fcode is known to set Fout=10,000,000, Equation 8.1a can be used to find an approximate value for R.

It should be obvious that an accurate frequency-measurement system will be required to calibrate the true reference frequency R to the 0.001Hz level. Such calibration can be made using a highly accurate frequency counter or perhaps better, a Global Positioning System Disciplined Oscillator GPSDO [8.3] and at least one of an accurate frequency counter, or one capable of accepting the GPSDO as a reference frequency (such as the inexpensive FA-2 [8.4]), or either an ordinary analog oscilloscope or a circuit capable of displaying beats. Calibration issues will be discussed in the next chapter.

Section 8.5: Comments on the Point, Range and Regression Calculations for R

In order to produce an accurate output frequency Fout, the true reference frequency R and Fcode must be determined.

$$F_{out} = R\,\frac{F_{code}}{2^{32}} \qquad \text{(8.1a repeat)}$$

Three of the easiest and perhaps least expensive but not exquisitely accurate methods for determining the true reference frequency R consist of what might be called the Point, Range and Regression methods. Equations 8.1a indicates the uncertainty in the output frequency ΔF_{out} can be related to the uncertainty in the reference frequency ΔR (assuming small changes in Fcode are not of interest so that $\Delta F_{code} = 0$) by

$$\Delta F_{out} = \Delta R\,\frac{F_{code}}{2^{32}} \qquad (8.6)$$

As discussed in relation to Equation 8.4b, for an uncertainty in the reference frequency of $\Delta R = 1$ and for an output frequency of 10MHz approximately produced by Fcode = 32F0AD82, the uncertainty in the output frequency would be $\Delta F_{out} = 0.2$ Hz which is rather large. So the main objective must be to determine the reference frequency with as much accuracy as possible.

(i) Point

The lowest order estimate (or so it would seem) of the true reference frequency R is simply found by substituting a single point into Equations 8.1a.

$$R = 2^{32}\,\frac{F_{out}}{F_{code}} \qquad (8.7a)$$

where, as before, $2^{32} = 4{,}294{,}967{,}296$. Consider the inexpensive Sinometer frequency counter VC2000 which, after calibration, has an error on the order of $\pm 0.5Hz$ (c.f., Appendix 2) and that error can be included in Equation 8.7a to explicitly see the error term:

$$R = 2^{32}\,\frac{F_{out} \pm 0.5}{F_{code}} = 2^{32}\,\frac{F_{out}}{F_{code}} + 2^{32}\,\frac{\pm 0.5}{F_{code}}$$

The expected error ΔR_1 based on the frequency counter error is then (note the subscript 1 for the one point method)

$$\Delta R_1 = \pm 2^{32}\,\frac{0.5}{F_{code}} \qquad (8.7b)$$

which for Fcode=32F0AD80 giving approximately 10MHz yields

$$\text{Expected Error in R:} \qquad \Delta R_1 = \pm 2.5 \text{ Hz} \tag{8.7c}$$

Notice, the same result can be found from Equation 8.6 using $\Delta F_{out} = \pm 0.5$. Now compare the expected error in R with the actual error. The 'actual error' is found by computing R, call it R_1, from Equation 8.7a and subtracting the 'actual value' of R, call it R_{gpsdo}, obtained from careful GPSDO calibration in Chapter 9. The Sinometer VC2000 frequency counter provided Fout = 9,959,078 for Fcode = 32BB503E. So Equation 8.7a provides R_1=50,255,053.8. A best value determined by the GPSDO in Table 9.2 is R_{gpsdo}=50,255,056.777515 (Fcode=32F0AD80, Fout=9,999,999.9994). The actual error in R is then the difference R_1-R_{gpsdo} = -3, which can be compared with the expected error of $\pm 2.5 Hz$ based on the single point measurement (see Table 8.1).

Table 8.1: Results for the single point method.

Fcode	Fout	Est. R Value	Actual R Error	Expected R Error
32BB503E	9,959,078	50,255,053.8	3	± 2.5

An uncertainty of 3Hz for R leads to an uncertainty of about ± 0.6 Hz for Fout based on the discussion with Equation 8.6. One might think that the error should decrease when using a range of data points to deduce R, but this turns out to be incorrect.

(ii) Range

The range method for finding an approximate true Ref value (i.e., calibration) again uses the simple frequency counter VC2000 that displays to the 1 Hz level with 10sec gate time. Assume the code values Fcode1 and Fcode2 produce the two frequencies Fout1 and Fout2, respectively. Substituting these values into Equation 8.1a to produce two relations involving R, subtract the two equations and solve for R to obtain

$$R = 2^{32} \frac{(Fout2 - Fout1)}{(Fcode2 - Fcode1)} \tag{8.8a}$$

Suppose the measurement of Fout by the frequency counter is accurate to within $\pm 0.5 \; Hz$ then this last relation provides an estimated value for R of

$$R = 2^{32} \frac{(Fout2 \; \pm 0.5 \; Hz \; - Fout1 \pm 0.5 \; Hz\,)}{Fcode2 - Fcode1} = 2^{32} \frac{(Fout2 - Fout1)}{Fcode2 - Fcode1} + 2^{32} \frac{\pm 1}{Fcode2 - Fcode1}$$

where the last term provides the expect worse case. The last term leads one to speculate that the wider the range of frequency, specifically the larger is $Fcode2 - Fcode1$, then the smaller will be the error in R because of the division by $Fcode2 - Fcode1$ in the last term. The error ΔR_r for the range (note the 'r' subscript) can be written as

$$\Delta R_r \; = \pm 2^{32} \frac{1}{Fcode2 - Fcode1} \tag{8.8b}$$

As an example, consider our measurements using the VC2000 calibrated to the 0.5Hz level. We find the following results for two points

Table 8.2: Results for the 2 point estimate

Fcode	Fout	Est. R Value	Actual R Error	Expected R Error
28BB503E	7,995,990	50,255,055.2	1.5	±12
3DBB503E	12,118,475			

Perhaps the numbers have conspired to provide a reading that differs by only 1.5 from the more accurate GPSDO number of 50255056.777515. In general, the expected maximum error (from Equation 8.7b) is ±12Hz. Notice the expected uncertainty in the reference frequency is larger than that for the single point method. Why? The answer: the point method is more accurate because it is a special case of the range method. The point (Fcode, Fout) = (0,0) is a valid fixed point for the FE5650A. Including the point (0,0) in the range (i.e., Fcode=0 produces Fout=0) makes the denominator in 8.7b as large as possible and thereby decreases the error.

(iii) Regression

For regression, generally a large number of data points will be measured and plotted on a graph, and then the best straight line will be drawn through the points (Appendix 4). In the present case, the output frequency Fout measured on the lowly VC2000 frequency counter will be plotted against the Fcode. A plot of the points for output frequency Fout vs. Fcode will scatter about a straight line since Equation 8.1a has the form of a straight line (y=mx+b)

$$F_{out} = mF_{code} + b \qquad (8.9a)$$

where the slope and intercept have the values

$$m = R/2^{32} \qquad \text{b=0} \qquad (8.9b)$$

So once the best straight line is drawn through the points (by linear regression) the slope will be known and the true reference frequency R can be determined through Equation 8.9b. The linear regression can be performed by hand and calculator; however, it's easier to use math software such as SMath (available free from smath.com). Appendix 4 includes relevant calculations for regression and also an example SMath sheet.

For the present case, the data set consisted of 22 points equally spaced in the range of approximately 8MHz to 12MHz. The linear regression provided a slope m from which R was then deduced. The error between the estimated R and the GPSDO value was 11 Hz as shown in Table 8.3. The actual error of 11 is quite large and on the same order as found from the two-point range method.

Table 8.3: Results for regression with and without the point (0,0)

Data Point Set	$R = m * 2^{32}$	R Error (Hz)
Does NOT include 0,0	50,255,067.6	11
Does include 0,0	50,255,057.4	0.7

Next, the point Fout=0 for Fcode=0 was included as shown in Table 8.3, and the actual error dropped an order of magnitude to 0.7 which is roughly half that for the point method. It would appear that the FE-

5650A can be calibrated using regression and a 1Hz resolution frequency counter to the extent that the error for R is smaller than 1 Hz. To further improve the accuracy, it should be clear that a highly accurate frequency counter or frequency standard such as the inexpensive FA-2 would be required.

Section 8.6: References

[8.1] Calculator2:
Windows 10: https://www.microsoft.com/en-us/p/calculator/9wzdncrfhwxl?activetab=pivot:overviewtab
iPhone: Available; check Apple App Store or Google Play. Author: Richard Walkers Calculator^2.

[8.2] Mark Sims, "FEI Rubidium Oscillators comment,"
http://www.ko4bb.com/doku2015/doku.php?id=precision_timing:rubidium_oscillators
https://www.mail-archive.com/time-nuts@febo.com/msg13486.html

[8.3] GPS Disciplined Oscillator (GPSDO) : BG7TBL
Ebay item number: 163420544010 and others

[8.4] FA-2 frequency counter
EBAY: Ebay item number EIN: 183971992537 and others
Amazon: EAN: 0781827663950

Chapter 9: Calibration

The FEI Rubidium Frequency Standard (RFS) FE-5650A/5680A should be calibrated in the sense of determining the true reference frequency R. In response to the status enquiry, the units should return a stored reference frequency Rstored which differs from the true value R as discussed in Chapter 8. In addition, the status request returns the Fcode (i.e., tuning word) that sets the default frequency of the unit. If the default frequency is known to several decimal points, then it is possible to find R using Equations 8.1 (Section 8.1) since the corresponding Fcode is known. Once the true reference frequency has been determined, then a tuning word Fcode can be sent through the serial port to set a desired output frequency Fout.

The chapter describes several methods of calibrating the FE-5650A without needing to know the stored reference frequency. The easiest method uses a calibrated frequency counter with 0.001 Hz resolution to deduce a value for the true reference frequency R. A number of frequency counters can be found with the requisite accuracy, but we use the FA-2 with a GPS Discipline Oscillator (GPSDO) as the auxiliary reference signal. Unfortunately, the FA-2 has some bias for the Allen Deviation since it does have some dead time although it appears to be minimal for the 10second gate time. A somewhat related method already been mentioned in the previous chapter, uses an inexpensive Sinometer VC2000 calibrated to the 1Hz level. Several methods provided here use ‘beats’ to compare the RFS output against the accurate GPSDO signal. These several methods include the use of a dual or single trace oscilloscope, or an inexpensive circuit using either an analog meter or LED to display the beats. Of course, it is still possible to obtain a somewhat accurate true reference frequency from the stored reference frequency as discussed in the previous chapter. All of these methods, except the last one using Rstored, do require an accurate frequency standard. As an interesting aside, the VC2000 needs to be calibrated to the 1Hz level which can be accomplished using the (free and simple) National Institute of Standards and Technology (NIST) WWV 10MHz broadcast (Appendix 2) or using the default frequency of the FE5650A (typically 8.388608MHz).

Figure 9.1: Front (left) and back (right) of the FA-2 frequency counter

Section 9.1: BG7TBL FA-2 Frequency Counter and GPSDO for Calibration

The easiest and perhaps least expensive method of accurately calibrating the FE5650A RFS is to use the BG7TBL FA-2 Precision Frequency Counter [9.1] shown in Figure 9.1 along with the BG7TBL GPSDO [9.2]. The 10MHz output from the GPSDO connects to the FA-2 reference input on the backside. The GPSDO provides sufficient accuracy to check the calibration of the RFS and determine an Allan Deviation Plot. At the time of this writing, both BG7TBL units cost in the neighborhood of $100 from

EBay. Even if the FA-2 is not purchased, it is a good idea to obtain a GPSDO for other methods of calibration. The GPSDO receives the highly accurate signals from the Cesium Frequency Standards (CFS) on GPS satellites; these signals then discipline an OCXO in the GPSDO. Generally, GPSDOs that have been receiving the GPS signals for longer periods of time will also have the better accuracy.

There has been some discussion on the accuracy of the BG7TBL GPSDO. According to one reference [9.3], the 10MHz output of the BG7TBL GPSDO (10MHz) has an accuracy of 0.05Hz at 30 minutes after startup, and 0.005Hz after 5hours. Another reference [9.4] indicates a 2016 BG7TBL models had a 'bug' so that what should have been 10,000,000.000,000 Hz was actually 9,999,999.999800 Hz while another reference claims that the bug has been corrected [9.5]. An error of either 0.005Hz or -0.0002Hz will be ok for our purposes. Apparently longer run times will bring the GPSDO closer to the 10MHz.

For the measurements made here, the GPSDO was allowed to stabilize for 5 hours and the 10MHz output was connected to the Reference Input on the backside of the FA-2. The Serial Output also on the backside was connected to the PC USB port. The computer was running the *Termite* terminal software [9.6] (also refer to Section 6.2) and it was configured to save incoming ASCII to the PC hard drive. Termite was set to 9600, N, 8, 1: 9600 baud, no parity, 8 data bits, 1 stop bit. In addition, Termite was configured to record a time stamp with the received data in order to determine the time between the samples. The FA-2 will only transmit ASCII to the PC when it receives a valid signal on its front input suitable for 10MHz. The FA-2 was set for a 10 second gate time [9.7-8] but the actual cycle time was found from the time stamp to be approximately 10.3 seconds and so there was approximately 0.3 seconds of dead time.

The FA-2 sends the measured frequency via the serial port to the PC. The Termite terminal software sends the received ASCII and also should be configured to include a time stamp (i.e., time of the measurement). The software will place a .txt file on the hard drive with entries having the form

HH:MM:SS: * F:xxxxxxxx.xxxxxx… (9.1)

By the way, the sleep mode on the PC should be disabled (Windows 10) by right clicking on an unused portion of the PC screen, selecting Display Settings, clicking Power and Sleep, and then setting the Power and Sleep modes to NEVER. The settings prevent the PC from interrupting the serial port and Termite. Also, don't switch ON the FA-2 until both the serial port and the reference have been connected since the FA-2 internal circuits must automatically sense the external reference and the serial port at startup.

Once all of the data has been collected on the PC hard drive, the FA-Convert conversion utility [9.12] is run on the PC to provide a file with the appropriate frequency and time format [9.10]. The time must be converted to seconds, the ':' after SS must be stripped, the * (if present) and 'F:' must be stripped out, and the frequency at the end converted to a number. The software allows the data to be written to file in several different formats and can be selected according to the plotting routine. Some software will accept a single file with both the time and frequency (or fractional frequency) in the same file whereas others might expect the frequency to be in a file of its own. Some software might simply plot the frequency versus sample number and so the proper horizontal time scale can be found by multiplying each sample number by the Gate Time on the FA-2.

Consider first the plots of fractional frequency and Allan Deviation for the FE-5650A RFS (Unit #1 in Table 6.4 in Section 6.6) and the 10MHz Mtron [9.9] Temperature Controlled Crystal Oscillator (TCXO) in order to compare the frequency behaviors. The RFS was set to produce a frequency Fo as close as possible to 10MHz which corresponds to Fcode=32F0AD9C. The Frequency and time stamp were collected over a period of roughly 1.5 hours for a FE-5650A unit and the TCXO. The left two panels (Figures 9.2 a,c) show the RFS (top) and the TCXO (bottom). Consider the RFS first. Panel (a) shows the Frequency versus Time (seconds) for 100 seconds prior to the frequency lock condition on the RFS; only approximately 500 seconds out of the 1.5 hours appears in panel (a) - see also Figure 5.10 in Section 5.6. The frequency vs time plot (a) shows the RFS is still scanning the frequency to obtain the lock condition with excursions on the order of -250Hz to 100Hz. Panel (b) shows the Allan Deviation Plot starting approximately 8 minutes after startup when the temperature sensor on the RFS shield reached 37C. The initial rise indicates an increasing instability as might be potentially attributed to a random walk. Panel (b) indicates an Allan Deviation of approximately 0.03 ppb near an averaging time of 1000secs.

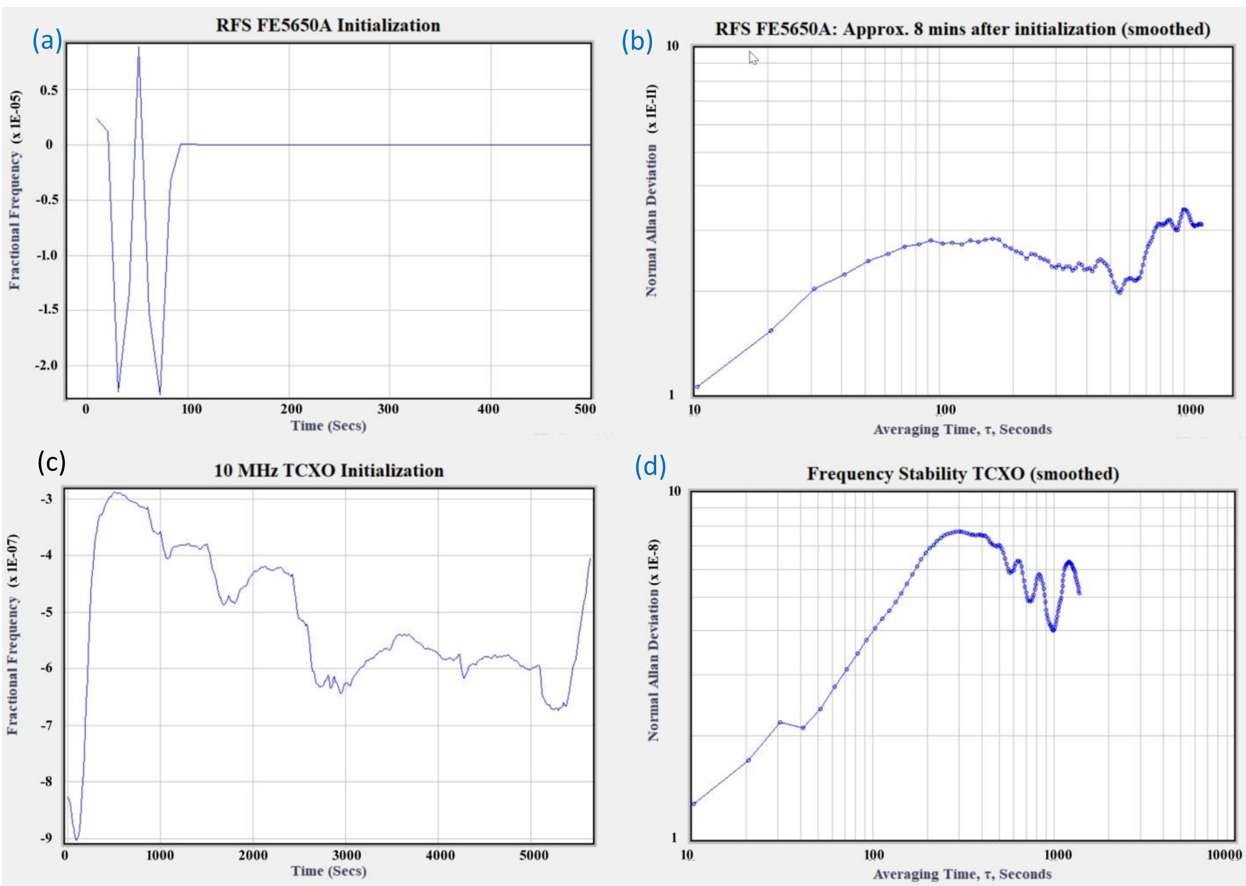


Figure 9.2: Plots for the fractional-frequency time response (left) and Allan Deviation (right) for the FE5650A (top, 10MHz) and Mtron K1601T 10MHz TCXO (right, 10MHz) [9.6]. (a) RFS fractional-frequency versus time starting about 100 seconds prior to lock. After lock, the RFS actually ran approximately 1.5 hours. (b) RFS Allan Deviation vs. averaging time after allowing the unit to run for approximately 8 minutes. (c) MTron TCXO fractional-frequency versus time (seconds). (d) MTron Allan Deviation versus averaging time.

The bottom panels in Figure 9.2, namely (c) and (d), show the Fractional Frequency vs. time and the Allan Deviation vs. averaging time, respectively, for the Mtron Temperature Controlled Crystal Oscillator (TCXO). Both plots appear similar to a random walk especially for low-τ in panel (d). Plot (c) shows, by visual inspection, the departure from 10 MHz is about 10MHz x (6E-07) = 6 Hz and the deviation from the average of approximately -5E-07 is roughly 2E-07 Hz which is about 10MHz x 2E-07 = 2 Hz. For the TCXO, the Allan Deviation (d) shows the worse instability approaches 10E-08 corresponding to 10MHz x 10E-08 = 1 Hz.

The main point of this section concerns the value of the true reference frequency R for the FE-5650A. The reference frequency can be determined by measuring the RFS output frequency Fo produced by the tuning word Fcode sent to the RFS. The relevant Equation appears in Section 3.2, 4.1 and 8.1 and repeated here in the form

$$R = \frac{2^{32}\, F_{out}}{F_{code}} \tag{9.1}$$

The FE5650A RFS was given the tuning word Fcode = 32F0AD9C and it produced the output frequency of Fo=9,999,999.99877 which is the average over the recorded frequency values starting roughly 8 mins after the RFS startup corresponding to Figure 9.2(d). The calculation produces the value for R shown in the 4th column of Table 9.1, specifically R=50255055.128047.

Table 9.1: Value of R, Rstored and the proportionality constant $k_p = R/R_{stored}$

Unit/SN	Fcode	Fo Measured	R Calculated	R Stored	k_p
1 61627	32F0AD9C	9,999,999.99877	**50255055.128047**	50255054.934100	1.0000000038593

As previously discussed in Section 8.2 (Equation 8.5a), some references suggest a proportionality constant k_p for the relation

$$R = k_p R_{stored} \tag{9.2}$$

The PIC microcontroller on the FE-5650A returns the value of R that it uses to calculate Fo, namely Rstored, when it receives the status command. The value of k_p potentially allows one to accurately produce a desired output frequency F_o without going through a calibration since k_p comes with the unit. In the case of Table 9.1, the value of k_p applies a correction to Rstored to obtain R. If the value of k_p could be applied to all FE5650A units, then none of the units would need to be subjected to separate calibrations. But alas this turns out not to be exactly the case (maybe approximately).

Section 9.2: An Oscilloscope and a GPSDO for Calibration

An oscilloscope can be configured to provide relatively accurate calibration of an unknown (but stable) frequency source by comparing it with a second known accurate stable source. The calibrated frequency source used here consists of the GPS Disciplined Oscillator (GPSDO). The uncalibrated source can be any one of the Rubidium Standard, Function Generator (OCXO), Signal Generator, or other tunable stable oscillator. The calibration resolution can be better than 0.01 Hz just by observing the oscilloscope display. The first example discusses the use of an older analog dual-channel oscilloscope with 20MHz bandwidth. The 20 MHz bandwidth clearly shows the 10MHz waveform from the frequency standard while the dual channel configuration makes it easy to observe the uncalibrated sinusoidal wave shifting with respect to the calibrated one. On the downside, this first configuration requires two channels and the 20MHz bandwidth. A second possible configuration uses a single channel scope with 20 MHz bandwidth although discerning the maximum or minimum of the wave can be a bit tricky.

A third possibility uses a simple LED circuit that sums the two signals and then detects the difference frequency and converts it to a DC level across an LED. The differences of 0.1-0.01 Hz can be easily observed. Replacing the LED with an analog voltage meter provides better calibration accuracy (i.e., more valid decimal points). It should be pointed out that the calibration accuracy of an oscillator will be limited by its stability (for one thing). A crystal oscillator frequency will drift as the temperature of the crystal changes which might be as much as 0.5-1Hz for an oven controlled crystal oscillator (OCXO) and even more for a temperature controlled crystal oscillator (TCXO). Of course this means that the OCXO calibration can be no better than 0.5-1Hz at any given instant of time. On the other hand, if the term 'stable' means that the crystal oscillator remains centered about the true frequency then when the frequency is averaged over long time periods, the average will be much better than the 0.5-1Hz.

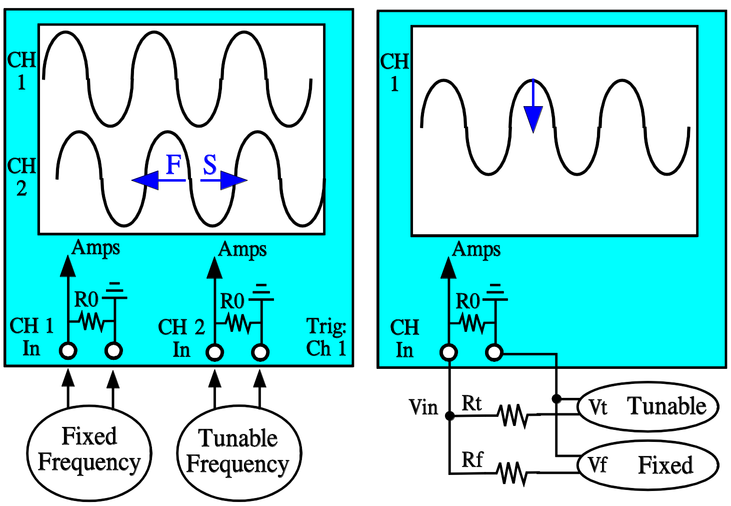


Figure 9.3: Left: A known high precision standard with fixed frequency F_F couples to channel 1 and triggers the oscilloscope. A stable but tunable standard, frequency F_T, couples to channel 2. If the tunable frequency is faster than the fixed one (i.e., $F_T > F_F$) then the Ch2 wave appears to move in the direction of F (fast). If $F_T < F_F$ (i.e., slower S) then the Ch2 wave moves to the right in the direction S (slow). Right: Summing the signals from the Tunable and Fixed sources will cause the waveform to oscillate up and down. When the two frequencies are equal, the displayed wave will remain motionless.

Consider first the case of the dual trace scope as shown by the left hand portion of Figure 9.3. The oscillator with known fixed frequency couples to channel 1 and triggers the oscilloscope. Suppose the Tunable Frequency F_T is 'faster' than the Fixed Frequency F_F (that is, $F_T > F_F$) and at some instant in time appear as in the left portion of Figure 9.4. Because the wave F_F is faster, the peak will occur at sooner times compared with the F_F peaks; consequently, the Ch2 wave appears to move toward the left. Similarly, for $F_T < F_F$, the Ch2 wave will move toward the right. The idea is to adjust the tunable frequency until the wave in channel 2 remains stationary (as close as possible) relative to the wave in channel 1. Often the tunable frequency source will have a dial or scale associated with the frequency. So once the two frequencies agree, one then knows the frequency corresponding to the particular dial/scale reading. For example, if the GPSDO provides the fixed frequency of 10MHz and the Rubidium Source provides the tunable frequency, then

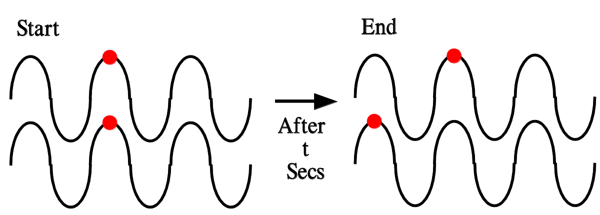


Figure 9.4: The top waves correspond to fix frequency F_F and the lower waves to unknown tunable frequency F_T. When $F_T > F_F$, the lower wave will move left from one peak to the adjacent one after t seconds.

the range of hex code used to set the frequency is the dial/scale and the particular hex code that makes the two frequencies equal must correspond to 10MHz.

When the two frequencies F_T and F_F are close but not exactly the same, it's easy to calculate the difference between them. One only needs to find the time t for the Ch2 signal to pass across one complete cycle of the Ch1 signal as shown in Figure 9.4. The red dot in the figure shows how a lower peak moved to the left to sit below the adjacent peak after t seconds. In such a case, the difference in frequency is 1 cycle divided by the time t

$$|F_T - F_F| = 1/t \tag{9.3}$$

Example 9.1: Suppose the crest of the wave in Channel 2 requires 1min 40secs (i.e., 100sec) to pass from one crest to the next in Channel 1. Then Equation 9.3 shows the frequency mismatch is

1/100 = 0.01 Hz.

If channel 2 moves in the fast direction F to the left, then $F_T = F_F + 0.01$. If channel 2 moves in the slow direction S to the right, then $F_T = F_F - 0.01$.

===

Sometimes the wave of Ch2 might shift so slowly compared to that of Ch1, it would be impractical to wait for the Ch2 wave to transition across a complete single cycle of the Ch1 wave. In such a case, one measures the time t (in seconds) required for the Ch2 wave to shift by Δt seconds (usually nSec or uSec as read on the oscilloscope time scale). The period T of the fixed wave should be known or measured on the oscilloscope. As usual, the relation between the cycle period T and frequency F is

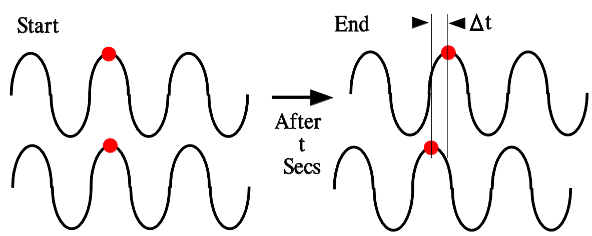


Figure 9.5: The faster wave moves toward the left by a small time Δt as read on the oscilloscope. The red dot shows the motion of the lower crest after t seconds.

$$F = 1 / T \tag{9.4}$$

In terms of radians, the phase shift after Δt seconds is

$$\Delta\varphi = 2\pi \frac{\Delta t}{T_F} \tag{9.5}$$

where $\Delta t / T_F$ is the fraction of one fixed cycle and the 2π converts cycles to radians. The phase difference is obviously related to the small time displacement between the two waves. Now to relate the phase shift to frequency difference, consider the definition of phase (difference) based on the angular frequency ω (radians/second) for each wave (fixed F and tunable T):

$$\varphi_F = \omega_F\, t = 2\pi F_F t \tag{9.6a}$$
$$\varphi_T = \omega_T\, t = 2\pi F_T t \tag{9.6b}$$

where the 'F' subscript refers to the fixed frequency source and the T refers to the tunable one. We have assumed the waves started in phase when t=0. Subtracting the two equations provides the difference in phase between the two waves after the time t.

$$\Delta\varphi = 2\pi\,(F_T - F_F)\,t \tag{9.7}$$

Combining this last equation with Equation 9.5 provides the desired result

$$|F_T - F_F| = \frac{\Delta t}{t\,T_F} \tag{9.8}$$

where we now assume Δt is a small positive number and to reiterate, t is the amount of time for the Δt to be established. The Δt is the number of (phase) seconds difference between the two signals and t is the number of seconds as measured by a stopwatch that was required to develop the phase difference Δt (t is the time required to achieve the Δt). Notice that Equation 9.8 reduces to Equation 9.3 when the delay time is one cycle, that is, $\Delta t = T_F$.

Example 9.2: Suppose we want to compare a Rubidium Frequency source against a GPSDO Frequency standard which is assumed totally accurate. A comparison on the oscilloscope shows the Rubidium frequency F_T to be slightly larger than the GPSDO frequency F_F. Suppose the Rubidium wave moves left by approximately $\Delta t = 10nSec$ after 4 mins 20 secs (i.e., t=260 seconds). Given the GPSDO provides an exact 10MHz signal, an exact period for the GPSDO signal can be calculated. Using the relation between frequency F and period T, namely FT = 1, we have an exact period of T = 1/F = 100 nSecs for the GPSDO. Equation 9.8 then provides

$$|F_T - F_F| = \frac{\Delta t}{t\,T} = \frac{10}{(260)\,(100)} = 0.0004 \text{ Hz}$$

Section 9.3: Calibration using a Single-Trace Oscilloscope

The second configuration for the oscilloscope appears in the right hand portion of Figure 9.6. In this case, the fixed-frequency and tunable-frequency signals are connected to the single channel using a voltage summer consisting of R_f, R_t and input resistance R_0 (capacitors can be connect in series with R_t and R_f to remove DC voltage components). A resistance of 2.2k for R_t and R_f, works fine. The amplitude of the signals from the tunable and fixed sources should be made approximately equal. With reference to the meter circuit below, the input resistors R_t and R_f can be replaced with a potentiometer (2k-5k) to balance the two frequency source signals if they don't have signal amplitude controls.

As indicated in the right hand side of Figure 9.3, when the two frequencies are close but not quite the same, the *displayed* sinusoid collapses to a straight line and then rises again to a 'reconstituted' sinusoid. The superposition of the two sinusoidal *input* signals of slightly different frequencies can be viewed in terms of 'beats' (Appendix 3). The displayed sinusoidal wave (i.e., the summed wave at maximum amplitude) occurs when the crest of one input sinusoid lines up with the crest of the other. The straight line (i.e., zero amplitude) occurs when the crest of one input sinusoid superposes with the trough of the other. Each time the crests of the two input waves overlap (or equivalently, each time the troughs overlap), the scope displays a sinusoid with maximum amplitude. The difference in frequency satisfies Equation 9.3,

$$|F_T - F_F| = \frac{1}{t} \tag{9.9}$$

where, as before, the time t in seconds is the time for one complete cycle consisting of the collapse and reconstitution of the sinusoidal wave. Refer to Appendix 3 for the full (and on the verge of subtle) description of the beat.

Example 9.3: Suppose the GPSDO frequency standard output is superposed with the output of a tunable function generator in the manner shown on the right side of Figure 9.3. Suppose the superposed waveform collapses to a straight line and then reconstitutes so as to form a beat with a cycle time of 10 seconds. The difference in frequency is then found from Equation 9.9 to be

$$|F_T - F_F| = \frac{1}{10} = 0.1$$

So the calibration of the function generator would produce an error less than 1 Hz (actually 0.1 Hz).

Section 9.4: Calibration using a Simple LED or Meter Circuit

The simple circuits (Figure 9.6) are the equivalent of using the single trace scope in that both the led intensity and the meter movement show the beats. The LED circuit (Fig. 9.6a) is low cost but estimating the LED intensity results in low accuracy. The analog meter circuit (Fig. 9.6b) has higher accuracy because of the tick marks on the meter scale; the meter scale makes it easier to see the change in the phase difference between the two signals.

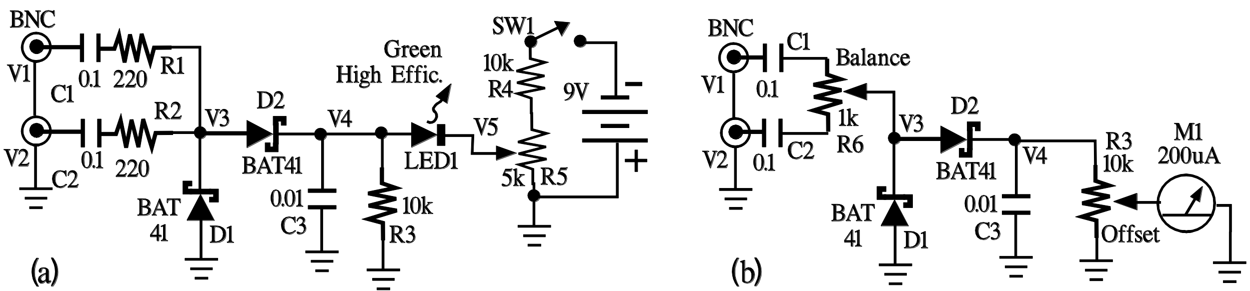


Figure 9.6: Circuits to determine when the frequency of one signal at V1 matches that at V2. (a) The left panel shows an LED circuit where the LED blinking stops when the two frequencies match. (b) The right panel shows a meter that provides better accuracy for comparing the two input frequencies. Notice potentiometer R6 replaces R1 and R2 and can be adjusted to produce roughly equal signals from the inputs at V1 and V2. The same combination of R6-C1-C2 of (b) can replace R1-R2-C1-C2 of (a).

Consider first, the LED circuit in Figure 9.6a. The components were chosen to service the range 1MHz – 15MHz. Input voltages V1 and V2 refer to the signals from the fixed and tunable sources. The LED will slowly blink when the two frequencies become nearly equal; it produces constant intensity at equal frequencies. Resistors R1 and R2 limit the current from the sources while capacitors C1 and C2 remove any input offset voltage. In order to develop a DC voltage across the LED, the circuit uses diode D1 to channel the summed input current to ground for negative voltages V3 without significantly affecting the current for positive voltage V3. Typically, diode D2 would be used with capacitor C3 to detect and hold the peak but R3 discharges of C3 – that is, C3 must discharge for the voltage V4 to follow the beat voltage. Diode D2 helps eliminate the negative going voltages V3 that would otherwise discharge C3. Capacitor C3 removes the high frequency components and charges to a DC level as controlled by current drain through R3 and LED1. As a note, diodes D1 and D2 should be Schottky diodes for the lowest forward voltage drop (improves sensitivity) but the circuit will work with Germanium or even Silicon diodes. The bias network on the right hand side of the circuit allows the LED to be given a range of negative bias on the cathode of the LED in case voltage V4 is not sufficient to bring the LED above threshold. Actually the exponential I-V relation of the LED adds significant sensitivity to the circuit. The circuit uses the high efficiency green LED which minimizes the current to a milliamp or less.

To use the circuit depicted in Figure 9.6a, connect the fixed frequency source to V1 and the tunable one to V2 (or vice versa) and turn them 'on'. Adjust the potentiometer R5 until the LED illuminates. As a first step, adjust the sources to produce similar voltages as follows: Alternately disconnect the fixed and tunable sources from the circuit and check that the LED has approximately the same brightness in either case. If one differs, adjust the input voltage levels or adjust the input resistors R1 and R2 (note, the circuit in 9.6b simplifies the procedure by using the potentiometer R6). With both sources connected, bring the tunable frequency as close as possible to the fixed one. The LED will begin to blink when the frequencies are within +- 30Hz of each other. Adjust the tunable frequency until the LED blink rate slows. Adjust the LED bias using R5 so that the LED remains illuminated throughout the entire beat cycle while still being able to see intensity variation for as much of the full cycle as possible. Adjust the tunable source frequency until the LED intensity quits changing and remains constant as much as possible. At this point, the two frequencies match as best as possible. Changes in the LED intensity can be difficult to notice and the circuit benefits by employing an analog meter instead of an LED. The scale on the meter makes it easy to track the phase of the beat signal. A digital meter would work of course so long as it rapidly updates the reading; however, viewing numbers is less intuitive than a moving needle.

Next, consider the meter-based circuit shown by Figure 9.6b. The input resistors R1-R2 have been replaced with potentiometer R6 which allows the amplitude of the two input signals to be matched. The LED and bias circuit associated with R5 have been removed. Now the 10k potentiometer R3 controls the maximum deflection of the meter needle. Three meters were tested and the best for the circuit was a 200uA analog meter from Amazon.

200uA meter, model 85C1, Amazon

The potentiometers can be adjusted to deflect the meter needle almost full scale for the beat. A 20mV DC full-scale was also test but found to be unwieldy for the currents involved. A digital multimeter was tested on various scales; for the 200 uA scale, the wiper was fairly close to the ground level but it was too difficult to correlate quickly changing numbers with the beat. The 200uA analog meter showed the greatest sensitivity, accuracy and best response time. With some experimentation, the circuit could be improved by adding a transistor or opamp to improve the voltage swing to a volt meter. Keep in mind that R3 (and any parallel paths) allows the capacitor to discharge so that V4 will be a scaled version of the beat voltage. Making C3 or R4 too large would keep V4 larger than the beat voltage and limit the swing of the meter. To help better control the discharge resistance across C3, it would be best to use a high input impedance amplifier or FET or a transistor configured for high input impedance. We use both the 200uA analog meter and the dual-channel analog scope for calibrations.

For the circuits in Figure 9.6, the frequency error between the two sources can be estimated in a manner similar that for the oscilloscope cases. The maximum LED intensity and meter reading occurs when the two input signals constructively overlap each other (crest to crest, trough to trough). The LED intensity will cycle according to Equation 9.9

$$|F_T - F_F| = 1/t \tag{9.10}$$

where t is the time in seconds required for the LED to cycle the intensity (refer to Appendix 3). To determine the actual RFS output frequency, one must observe the Fcode that makes the meter needle nearly stationary – call that Fcode0. Then for Fcodes smaller than Fcode0, the signal to be calibrated is slower so that Fout is found by subtracting the value in Equation 9.10 from the known accurate value.

Section 9.5: Example Calibration using the Dual Channel Oscilloscope

This section shows the calibration results for four of our FE5650A units using the dual-channel oscilloscope and Global Positioning System Disciplined Oscillator (GPSDO) with an accurate output frequency of 10MHz. After allowing the GPSDO to acquire satellites and lock the frequency, the output was connected to channel 1 of an analog dual trace 20MHz oscilloscope. The example calculations make use of unit #2 in Tables 9.2 below and 6.4 in Chapter 6. The FE5650A was attached to channel 2. The scope was triggered on channel 1.

For this trial, the Fcode was adjusted until the signal displayed on channel 2 became as stationary as possible. The value Fcode=32F0AD80 produced a sinusoidal signal moving exceedingly slow toward the right. Table 9.2 shows the moving sinusoid required t=390 seconds to move to the right (the S=slow direction) through a time of Δt=20 nSec. As a note, for all but this case, the fourth column in Table 9.2 shows the time for the uncalibrated Ch2 wave to move through a complete cycle of the calibrated Ch1 wave. Equation 9.7 uses the period T_F of the fixed calibrated frequency given by $T_F=1/F_F$ = 100 nSec, which is obviously the time between adjacent troughs or crests in the left panel of Figure 9.3. Using Equation 9.8, the frequency difference is

$$|F_T - F_F| = \frac{\Delta t}{t\, T_F} = \frac{(20nSec)}{(390Sec)(100nSec)} = 0.000561Hz$$

where as before, the time t is the time required for the Ch2 sinusoid to move through the time Δt. Since the tunable wave (i.e., the uncalibrated one) is slightly lower frequency, the 0.000561 will need to be subtracted from the calibrated fixed frequency of 10MHz to find the actual Fout of the uncalibrated one. **For simplicity, write δF to mean $\boldsymbol{\delta F = F_T - F_F}$ so δF will be negative when the uncalibrated tunable signal has lower frequency than the calibrated one** (i.e., the Ch2 wave moves toward the right in the Slow direction S). As a result, we find that for Fcode = 32F0AD80 (854,633,856 Dec), the FE5650A frequency can be calculated as

$$Fout = F_F - \delta F = 10{,}000{,}000.000 - 0.000561 = 9{,}999{,}999.999439\ Hz$$

Unit 2 shows the next Fcode of 32F0AD81 causes the wave to reverse and move left which indicates Fout is slightly larger than the GPSDO frequency of 10MHz:

$$Fout = F_F + \delta F = 10{,}000{,}000.000 + 0.000561 = 10{,}000{,}000.00833\ Hz$$

The reference frequency for each row, denoted as R_D (Column 6 of Table 9.2), can be calculated by Equation 9.1. The subscript D=direction for R_D refers to the S or F row for a given unit.

$$R = 2^{32} \frac{F_{out}}{F_{code}} \qquad (9.11)$$

For example,

$$R_{D=S} = 2^{32}\frac{F_{out}}{F_{code}} = 2^{32}\frac{9{,}999{,}999.999439}{32F0AD80} = 2^{32}\frac{9{,}999{,}999.999439}{854{,}633{,}856} = 50{,}255{,}056.777894$$

as shown in the unit #2 row for the Direction Dir=S under the heading of 'R=5025505_'. The label for the column of the form R_D=5025505_ should be understood to mean that the numbers underneath should be substituted for the underline character '_' such as **R_D=5025505_** 5.120258 means R_D=50255055.120258. The strange labels were adopted to save space so the table would fit the page.

Table 9.2: Table of units calibrated at 10MHz. Dir: S and F refer to $F_T < F_F$ and $F_T > F_F$, respectively. Fcodes: The Fcodes for S and F produce the closest Fout to the 10MHz. '1 cycle t' is the time in seconds for the uncalibrated signal to move through 1 cycle (period T_F) of the calibrated signal (10MHz). δF = 1/t. R is the true reference frequency. R_D is the reference frequency for the given Dir S or F. R is the average of the two values of R_D (for S and F) and should be understood as the actual true reference frequency. k_p is the ratio of the true to the stored reference frequency and the column average appears at the bottom; the brackets [ ] refer to the case of when Unit #4 is not included in the average. The last column is the difference between the predicted value of R and the actual value of R: the value of $R_{pred} = \langle k_p \rangle R_{stored}$ where R_{stored} is the stored reference frequency in Table 9.3; the numbers in brackets [ ] refer to the case when Unit #4 is not included in the average.

Unit/SN	Dir	Fcode	1 Cycle t	δF	R_D = 5025505_	R = 5025505_	k_p = 1.000 000 00_	δR = R_{pred} - R
1	S	32F0AD9C	360	-0.00278	5.120258	5.118773	3675	0.00378 / [0.0495]
61627	F	32F0AD9D	120	+0.00833	5.117288			
2	S	32F0AD80	0.02u/390	-0.000561	6.777894	6.777515	4852	-0.0554 / [-0.00966]
57803	F	32F0AD81	91	+0.0110	6.777136			
3	S	32F0AD94	214	-0.00467	5.581184	5.579624	5454	-0.0856 / [-0.0399]
55380	F	32F0AD95	156	+0.00641	5.578065			
4	S	32F0AD8F	113	-0.00885	5.854192	5.855006	1039	0.1362 / [----]
59321	F	32F0AD90	315	+0.003175	5.855821			
						Average:	3755/ [4660]	±0.1 / [±0.05]

Table 9.3 revisited: Response to the status enquiry. Table repeated from Section 6.6 for convenience.

Unit	FEI Number	Response to Status Enquiry
1	61627	R=50255054.934100Hz F=2ABB5050 7E8A5200<cr>OK
2	57803	R=50255056.533663Hz F=2ABB503A ACF26C00<cr>OK
3	55380	R=50255055.305534Hz F=2ABB504B 3210BE00<cr>OK
4	59321	R=50255055.802777Hz F=2ABB2A39 1A23AE00<cr>OK
5	71199	Incompatible Interface

The other entries for the actual true reference frequency R_D are similarly calculated except the calculation of δF is simply δF=1/t where t is the time listed in Column 4 of Table 9.2. The number of seconds t is the time for the uncalibrated signal to move through a complete cycle of the calibrated one is listed under the '1 Cycle S'. Continuing with unit #2, the Fcode of 32F0AD81 causes the uncalibrated sinusoid to have higher frequency than the calibrated one and it moves toward the left in the fast direction (denoted F under the Dir column). Using Equation 9.3, we find

$$\delta F = +\frac{1}{t} = +0.0110$$

Notice the '+' indicates that Fout for the FE5650 is larger than that for the GPSDO of 10MHz. The output frequency is then

$$Fout = F_F + \delta F = 10{,}000{,}000.000 + 0.0110 = 10{,}000{,}000.0110\,Hz$$

The corresponding true reference frequency from Equation 9.1 is R=50255056.777136 (Unit 2, Row S in Table 9.2). The true reference frequency R (Column 7) is computed as a two value average of the S and F rows for unit 2 as

$$R = \frac{R_S + R_F}{2} = 50255056.777515$$

All of the entries in the seventh column were calculated in this manner.

On the other hand, compare the results from Section 9.1 for Unit #1 true reference frequency of

R=50255055.128047 (Unit #1, FA-2 and GPSDO)

with those given in the sixth column of Table 9.2 for Unit #1. Notice the longer cycle time 360 gives the better estimate $R_{D=S} = 50255055.120258$ for the FA2-GPSDO value in Section 9.1. If this were to hold true for all of the S and F measurements, one would expect a better estimate for the true value of R would be to either use the R_D with the longer cycle time or use a weighted average so that the longer time t most heavily contributes

$$R = \frac{t_S R_S + t_F R_F}{t_S + t_F}$$

Such an average tends to indicate the smaller cycle times produce the greatest error as might be expected. The measurements of the cycle times can be checked. For a given unit such as unit #1 in Table 9.2, the values of δF (made positive) should add up to the digitization/quantization error of approximately 0.0117 Hz (refer to Equation 8.3b in Section 8.1 in the previous chapter). The sum provides a rough check on the measured times made for Table 9.2.

Unit #1: +0.00278 + 0.00833 = 0.0111 Hz
Unit #2: +0.000561 + 0.0110 = 0.0116 Hz
Unit #3: +0.00467 + 0.00641 = 0.0111 Hz
Unit #4: +0.00885 + 0.00317 = 0.0120 Hz

All of these are accurate to the third decimal place. Also note the comparison of R for unit #1 between Section 9.1 and Row 1 of Table 9.2 shows the Table 9.2 value has error in the 3rd decimal. Consequently, Equation 8.4b in Chapter 8, specifically $\Delta F_{out} = 0.2\, \Delta R$, indicates the RFS output frequency Fo should be accurate to the third digit.

The ratio k_p between the average true reference frequency R and the stored reference frequency from Table 9.2 can be calculated for each of the units (refer to Section 8.2). For example, Unit #2 provides

$$k_p = \frac{true\ R}{stored\ R} = \frac{50255056.777515}{50255056.533663} = 1.000\ 000\ 004\ 852$$

where 'true R' is taken as the value in the R column and the 'stored R' comes from Table 6.4. The average value of k_p, often denoted by $\langle k_p \rangle$, appears at the bottom of Column 8. The brackets [ ] refer to the case when Unit #4 is not included in the average. The last column shows the difference between the predicted value of R and the true value of R found from the calibration. Again, the brackets [ ] indicate that the values are calculated without including Unit #4 in the average of k_p. Linear regression (Appendix 4) was used to calculate a predicted value (akin to $\langle k_p \rangle R_{stored}$) for R with the results

$$R_{pred} = m\, R_{stored} + b \tag{9.12a}$$

$$m = 1.000944380 \qquad b = \text{-47459.6983174} \tag{9.12b}$$

Calculating the difference $\delta R = R_{pred} - R$ for the regression produces the same results as for the constant of proportionality method and so the last column in Table 9.2 also applies to the regression method. It would appear that an estimate of R can be found by simply multiplying the average constant of proportionality $\langle k_p \rangle$ given in the table by the value of R_{stored} obtained from the units nonvolatile memory. Notice that only Unit 4 misses the predicted value (Column 9) by the most. If that unit is left out, the average k_p would be 1.000,000,004,660. Leaving Unit #4 out of the average reduces the variation between the predicted and actual values of R from roughly ±0.1 to ±0.05. Therefore using Equation 8.4b in Chapter 8, specifically $\Delta F_{out} = 0.2\, \Delta R$ (even though it's for 10MHz), the uncertainty in Fout will be better than ±0.02 to ±0.01 Hz.

Section 9.6: Default Frequency

When power is applied to the units, the RFS units begin to oscillate at the default frequency. If one knows the frequency Fout, then based on the stored Fcode, one could find the true reference frequency from Equation 9.1 without the calibration procedure. Nice. The present section simply determines the default output frequency based on the true reference shown in Table 9.2 and the stored Fcode shown in Table 9.3. The calculations use Equation 9.1 and the results appear in Table 9.4.

Table 9.4: FE5650A default output frequency calculated from the true reference frequency R in Table 9.2 and the stored Fcode in Table 9.3.

Unit	FEI Number	True Ref. Freq.	Stored Fcode	Default Fout
1	61627	50255055.118773	2ABB5050	8,388,608.001640
2	57803	50255056.777515	2ABB503A	8,388,608.021098
3	55380	50255055.579624	2ABB504B	8,388,608.020061
4	59321	50255055.855006	2ABB2A39	8,388,494.028891

So the stored Fcode from all but the last unit could be used to determine a reasonable true reference frequency. Further, all but the last unit could be used right out of the box to calibrate electronic equipment or crystal oscillators to the 0.02 Hz level or better.

Section 9.7: Is Calibration Necessary?

Two widely separated groups of FE5650A have been calibrated for the true reference frequency R with the results compared with the stored reference frequency R_{stored} and summarized through a constant of proportionality k_p

$$R = k_p R_{stored} \tag{9.13}$$

One internet reference [9.11] found an average value of k_p = 1.000,000,002,150 whereas Table 9.2 shows an average value of k_p=1.000 000 003 755 for all four units and k_p=1.000 000 004 660 when Unit #4 is omitted from the average. The last column shows the variation of the predicted from the actual R, denoted δR, varies by roughly ±0.1 Hz for all four units and ±0.05 for Units #1-#3. As a consequence, based on Equation 8.4b in Chapter 8, the error in the output frequency, using predicted R values instead of the measured ones, can range from ±0.02 to ±0.01.

Calibrate? Well calculating the Fout values obtained using the predict values of the reference frequency can be in error by ±0.02 to ±0.01 which is on the same order as the quantization error at 10MHz. If this is acceptable, then no need to calibrate. The error will not especially change in time since the Rb frequency is stable to roughly the 0.001 Hz.

Section 9.8: The C-Field Potentiometer as the Final Adjustment

Apparently the C-Field potentiometer can be adjusted to correct for the effects of the frequency quantization caused by the AD9830A Direct Digital Synthesizer IC. To be clear, we have not tried this adjustment. But the idea consists of setting the Fcode to produce a frequency as close as possible to the desired output frequency. As discussed in Chapter 8, if the desired frequency is 10,000,000.000 Hz then one can expect a quantization error of +/- 0.002 Hz. So by setting up the system as in Section 9.1 or other sufficiently accurate method using the GPSDO, and Fcode=32F0AD80, the C-Field potentiometer could be adjusted until Fo=10,000,000.000 Hz … in theory. In our case, we are satisfied with the 0.002-0.01 Hz uncertainty.

Mark Sims [9.11] describes the situation according to his observation of several different units. As previously discussed the stored and true reference frequencies differ. He states that the stored reference frequency corresponds to the "minimum setting of the C-field potentiometer" and the tuning word Fcode is set one step below the desired output frequency Fout. He then states that the factory increases the C-field potentiometer to "increase the actual reference frequency to the desired value".

Section 9.9: FA-2 File Conversion Software and the Graphical User Interface

As mentioned in Section 9.1, the combination of the FA-2 frequency counter and the Termite software produces a text file with entries of the form

hh:mm:ss: * F:0010000023.123456789 (9.13)

The "FA-2 File Converter" software (Appendix 8) [9.12] separates the time and frequency parts and then opens new text files with the time in seconds and the frequency as the raw frequency (given by 'F:' in 9.13 above) or as the fractional frequency or as the error frequency. The text data can be written in one single or in multiple files as will be discussed. Probably it would be much more convenient to add a converter to the FA-2 unit or maybe to software that directly plots the Allan Deviation. Apparently the software Lady Heather has been modified to accept data from the FA-2 Frequency Counter. In any event, the 'FA-2 File Converter' can be used to create files that can be read by a variety of other software including Excel, the EZL plotting routines and the ADEV software discussed in a previous chapter. The software serves its purpose but the reader familiar with programming can improve it as needed.

The first version of the GUI for the 'FA-2 File Converter' appears in Figure 9.7 although other versions might become available that includes graphs, RS232 and ADEV options. The basic layout of the user interface (UI) consists of the various sections:

1. The ‘File Structure’ group determines the number of output text files and the order of the data as either Time,Freq or Freq,Time when all the data occupies a single file.
2. The ‘Frequency Format’ sets the format for the frequency in the output file as Fractional Frequency, Error Frequency or Raw (refer to Chapter 4 for the definitions).
3. The textbox ‘Nominal Frequency’ needs to be set to the expected frequency output from the device connected to the FA-2 front input (10MHz for the RFS). The textbox ‘Gate Set’ needs to be set to the Gate Time of the FA-2. The textboxes for the average frequency and Cycle time show the averaged frequency and the Cycle Time (i.e., gate time plus dead time), respectively, deduced from the data file. No effort was made to tailor the number of decimal places.
4. The button named “SELECT FA2 TXT” selects a text file originating from the FA-2 and Termite software. The top, right side textbox shows the selected filename and the bottom, right side textbox shows the raw content of the file. Note the ‘raw’ frequency appears after the ‘F:’ in the lower textbox for the figure panel (a).
5. The ‘RUN’ button modifies the Time and Frequency content as shown in the lower right textbox in the figure panel (b). The time offset of 2min 21sec (i.e., starting time in panel a) is subtracted from each time entry and so the time sequence would start at zero. Given the first frequency entry occurs after the gate time of 10 seconds, the gate time is added to each member which places the first time at 10 seconds. The frequency shown is the fractional frequency of that in (a) according to

   (Freq Raw – Nominal) / Nominal

   The error frequency would have the form (Freq Raw – Nominal).
6. The ‘Save File’ button saves the modified data shown in the lower textbox of panel (b) according to the selected radio button in the ‘File Structure’ group box.

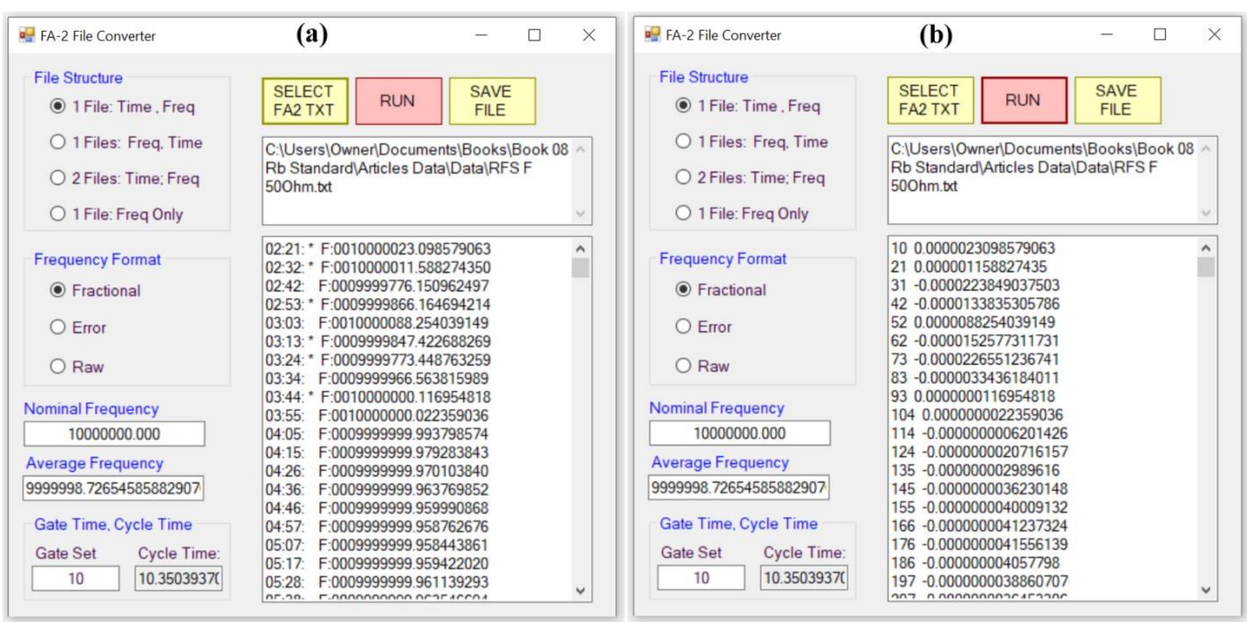


Figure 9.7: (a) The user interface after selecting the file ‘RFS F 50Ohm.txt’ shown in the upper textbox. The raw content appears in the lower textbox with time on the left and frequency on the right. (b) The user interface after clicking the RUN button. The offset of 2min 21sec has been subtracted from the time and then the 10sec gate time added.

Section 9.10: References

[9.1] Example BG7TBL FA-2: See Amazon number: 0634759491460. Also available on Ebay.
https://www.amazon.com/1Hz-6GHz-Frequency-Counter-11digit-s-10MHz/dp/B07VLN39NN/ref=sr_1_1?keywords=0634759491460&qid=1582673265&sr=8-1
[9.2] BG7TBL GPSDO
See for example EBay item number: 264136963968
[9.3] Accuracy of the BG7TBL GPSDO and some technical information
https://radioaficion.com/news/bg7tbl-gpsdo/
[9.4] Note on error regarding the10mhz (9,999,999.999800Hz)
https://www.eevblog.com/forum/testgear/bg7tbl-gpsdo-master-reference/
[9.5] Other various notes on the BG7TBL bug
http://www.ke5fx.com/gpscomp.htm
[9.6] Termite terminal software: https://www.compuphase.com/software_termite.htm
[9.7] FA-2 operator's manual: Press Reset and either of the two above buttons to set input impedance and low pass filter.
http://www.vklogger.com/viewtopic.php?t=14001
http://bg7tbl.taobao.com
[9.8] FA-2 discussions
https://www.eevblog.com/forum/metrology/bg7tbl-fa1-frequency-analyzer/
https://www.mail-archive.com/time-nuts@lists.febo.com/msg04652.html
[9.9] TCXO available from PBSN6040 on EBAY
https://www.ebay.com/itm/10-0000-MHz-TCXO-1ppm-Frequency-Standard/371321186440?hash=item567477a888:g:Q4kAAOxy2FZSP2vW
[9.10] Interesting calibration ideas
https://hackaday.com/2015/05/27/measuring-accuracy-of-rubidium-standard/
[9.11] Mark Sims, "FEI Rubidium Oscillators" 2013
http://www.ko4bb.com/doku2015/doku.php?id=precision_timing:rubidium_oscillators

[9.12] Source code, listings, downloads, flash drives, preprogrammed microcontroller:
(A) Chapter 10 and Appendix 6 provide the source code for the C/C++ program
(B) Appendices 7-9 provide the PC software
(C) Check EBay.com in connection with the title of this book: software and preprogrammed MEGA328P
(D) check www.AngstromLogic.com for current offerings, or write to Sales@AngstromLogic.com
Keep in mind that Angstrom Logic, LLC is not setup/able to answer questions on the book or software.
(E) Check Amazon in connection with the title of this book.
(F) Note: The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the words) and the contents are copyrighted with all rights reserved (also see front copyright page). The software/firmware is scanned for viruses before posting. Download only from secure sources and rescan for viruses before using.

Chapter 10: A Controller for the FE5650A Rubidium Frequency Standard

A simple microcontroller (uC) circuit can be used to control the FE5650A Rubidium Standard; the uC primarily sets the desired output frequency and transfers HEX Fcode to the FE5650A. The controller can interrogate the unit as well as display the result on a small text-based LCD. The chapter shows the circuits (refer to Appendix 6 for the code listings), and provides links to download the programs and possibly purchase preprogrammed chips [10.25]. If needed, the internet has many tutorials for microcontrollers and the construction techniques (c.f., References [10.2-8]). Only the first four sections of Chapter 10 need be read unless the controller programs will be modified in some manner. The last few sections of the chapter briefly describe the software primarily written with C. Several good C programming reference can be found in References [10.9-12].

Section 10.1: Overview

A simple microcontroller (uC) such as the Microchip-Atmel ATMEGA328P can easily be built into a controller to interface with the FEI FE5650A. Such a controller should have a means of communicating with the FEI unit to provide the capability of entering new frequencies, viewing various parameters, and measuring module temperatures. Although an LCD with a touch sensitive screen could be used for entering and viewing numbers, these LCDs tend to be large in order to show a keypad outline and can require more memory than some of the smaller microcontroller chips can provide. The controller described here uses a small HD44780 -based text LCD (two lines, 16 characters/line) and an old-fashioned 16-key keypad. The ATMEGA328P [10.13] has a single USART for communications, an ADC for the temperature measurements and a 16bit timer to trigger the ADC or for general purpose timing. As for programming the ATMEGA328P, we use the free-of-cost Microchip-Atmel Studio 7 [10.14] and the Microchip-Atmel ICE programmer [10.15]. The C/C++ program for the Microchip-Atmel ATMEGA328P can be purchased/downloaded [10.1] or copied from the source code in Appendix 6. Two easy alternatives would be to download the Atmel Solution or purchase a preprogrammed ATMEGA328P [10.1,25].

As discussed in previous chapters, the FE5650A circuitry includes a Direct Digital Synthesizer (DDS) AD9830A which stores the output frequency as HEX code denoted as F (and termed a ‘tuning word’) although we use the name Fcode or sometimes Fc. The DDS chip produces the desired output frequency using the Fcode and a reference frequency R originating in a Voltage Controlled Crystal Oscillator VCXO controlled by the Rubidium physics package. The embedded PIC microcontroller uC stores a value of the reference frequency Rstored which differs from the true value Rtrue. As previously discussed, the output frequency can be calculated as follows

$$F_{out} = R\,F_c/2^{32} \tag{10.1}$$

where R is the true reference frequency and Fc is the HEX code. To generate the sinewave with the proper frequency, the embedded microcontroller on the DDS board transfers a string of HEX characters from an incoming RS232 stream to the AD9830A. Therefore, an external controller only needs to send the correct string of characters to the module using RS232 as was seen in Chapter 6 in connection with the terminal software.

The present chapter develops a bare-minimum ATMEGA328P microcontroller circuit and a program to control the FE5650A [10.25]. All of the parameters such as Rstored, Rtrue, Fout, and Fcode are stored as strings since both the LCD and FE5650A consume strings as opposed to binary. Although routines can be developed to perform calculations using strings, it is more conventional and probably easier to use standard binary representations of the numbers. For the calculations in Equation 10.1, the Atmel AVR version of the GNU GCC tool chain used for compiling the C++ code does not allow sufficient size for the numbers involved for Equation 10.1. As a matter of fact, the 'float' and 'double' are the same. So it was necessary to select among (i) multiplying a number by 1000 or 1000000 so as to shift decimals into the integer portion of the number and then use UINT64_t representations, (ii) performing arithmetic operations on strings or (iii) finding/developing float routines capable of handling 64 bit numbers. Fortunately we found some float64 routines although they required some slight corrections but they work well [10.16]. The interested reader can find more information regarding the float format in Reference [10.17].

The program is divided into several parts. The main routine primarily handles various menus required to access or set the various parameters such as Rstored and Rtrue. The code for control of the USART (TTL RS232), Timer, and ADC along with the float 64 and some other routines are sectioned into libraries as will be discussed. All of the code was written in a few weeks and tested to function with the FE5650A; however, that doesn't mean there aren't any bugs – the big ones have been removed. The source code has been included in Appendix 6 to be modified as desired.

Previous chapters describe the FE5650A enclosure which includes a 15V and 5V power supply, fan and connectors (see Figure 10.1 for a block diagram). For the controller, several possible topologies were considered for the construction including (i) an Integrated Enclosure (top panel Figure 10.1) for both the FE5650A circuits and those for the Microchip-Atmel MEGA328P uC, LCD and keypad, (ii) separate FE5650A and Remote Enclosures (bottom panel Figure 10.1), and (iii) separate enclosures but designing the remote to use either the FE5650A power or its own batteries. A remote enclosure can be used with multiple FE5650A units.

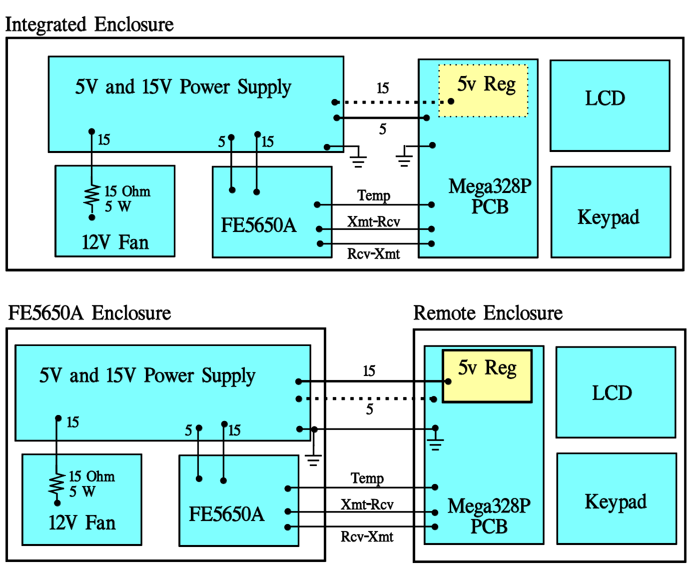


Figure 10.1: Multiple possible configurations for the FE5650A and the controller circuitry.

The top panel shows the 5v/15v power supply, Fan and FE5650A module along with the controller consisting of an ATMEGA328P microcontroller, LCD and Keypad. The 5V regulator can be eliminated by using the 5V from the 15V power supply.

The bottom panel shows the same functionality divided into two separate enclosures. The Remote Enclosure has the 328P microcontroller, LCD and Keypad. A 5v regulator on the PCB would require only the 15V line from the large power supply in the FE5650A enclosure but obviously not the 5V one. The two enclosures can be interconnected using a miniXLR connector with 5 pins.

The topologies essentially differ in the power connections. The construction of the single integrated enclosure makes it possible to use the 5V power already present. The ATMEGA328P and LCD require approximately 40mA total. It is also possible to include a 5V regulator with the 328P circuits and use the 15V supply already available for the FE5650A (see dotted line and box in the top panel of Figure 10.1). The construction of separate enclosures requires the Remote Enclosure to either use the power from the FE5650A Enclosure or to supply its own (or both). The bottom panel shows the Remote Enclosure receives 15V from the FE5650A and then drops it down to 5V using an LM7805 regulator. The dotted line shows it might be possible to eliminate the regulator and 15V, and simply use the 5V from the FE5650A Enclosure. The interconnecting cable has 5 conductors for the USART, temperature sensor and power and ground.

We decided to build separate enclosures and power the Remote Enclosure using (i) 15V from the FE5650A enclosure and (ii) a 9V battery internal to the remote. These were arranged so that the remote uses the FE5650A power when the cable is connected between them otherwise it can use the 9V battery. Such an arrangement allows the remote to be powered and programmed independently of the FE5650A connections. As mentioned, the completed circuit consisting of the Microchip-Atmel MEGA328P uC, LCD, keypad and regulator consumes only about 40mA and so the following batteries could be considered for temporary power. For example, a lithium 9V battery should last approximately 16 hours until its voltage drops to about 7V (the lower limit for the LM7805).

Lithium 9V http://data.energizer.com/pdfs/l522.pdf	>16 hours
Alkaline 9V http://data.energizer.com/pdfs/522.pdf	~7 hours
Alkaline six AA batteries http://data.energizer.com/pdfs/e91.pdf	~45 hours
Alkaline six AAA batteries http://data.energizer.com/pdfs/e92.pdf	<10 hours

Section 10.2: The uC Circuits

The microcontroller circuit appears in Figure 10.2 – a larger version can be found in Appendix 5. As mentioned in the figure description, the controller for the FE5650A Rubidium Standard essentially consists of a Microchip-Atmel microcontroller ATMEGA328P (28pin DIP), a 4 x 4 keypad, an HD44780-based (or ST7302) 16x2 text display, an LM35DZ temperature sensor, and a LM7805 5v regulator. Here's a partial list of parts.

ATMEGA328P, 28pin DIP Digikey ATMEGA328P-PU-ND
LCD 16x2 HD44780 Adafruit AD181, 100495 Amazon ASIN: B00NAY1B2Y
Crystal 16MHz, 10ppm, 18pF load, HC-49/US TXC: 9B-16.000MEEJ-B, Digikey: 887-1244-ND
Inductor 10uH Digikey M8181-ND Bourns: 8230-44-RC. Inductor kits also at Amazon.
LM35DZ Temperature Sensor Digikey: 296-35151-1-ND or Amazon B06WRTXNV3
LED 3mm diam., kit of various colors Amazon B01MU07JXX
Mini-XLR connectors: 5 pin, female TA5F Ebay.com 200913829314
Mini-XLR Panel Connector, male, 5 pin, EBay.com 201257949152
Keypad 16keys 4x4 Membrane Amazon B07B4DR5SH

The circuit uses an Adafruit **LCD** (AD181) based on the HD44780 chip. Alternatively, the circuit could use a text LCD based on the ST7032 chip available from www.buydisplay.com
Display Serial 16x2 COG LCD Module, Pin Connection, White on Blue
These are compatible with the ubiquitous HD44780-based text displays but work from 3-5V in case you choose a 3.3 V microcontroller. They are manufactured by East RisingTechnology Co. as the ERC1602-4 series (under $3). However, the pinout will not match the LCD used in Figure 10.2. In any case, be sure

to choose an LCD with at least 2 lines of text and 16 characters per line (or more). For the current Adafruit display, the figure shows the circuits use only the upper 4 data lines D4-D7 of the LCD; the RW line is tied to ground since data will only be written to the LCD. The 10k trimmer potentiometer connected to pin 3 is used to adjust the contrast between the LCD background and the displayed characters. The trimmer also provides the capability of optimizing the display contrast when the viewing-angle or temperature changes. Notice that the pin 6 Enable line E on the LCD can disable the LCD Data and Control lines so that the uC can use the lines for other purposes until it needs to send characters to the LCD.

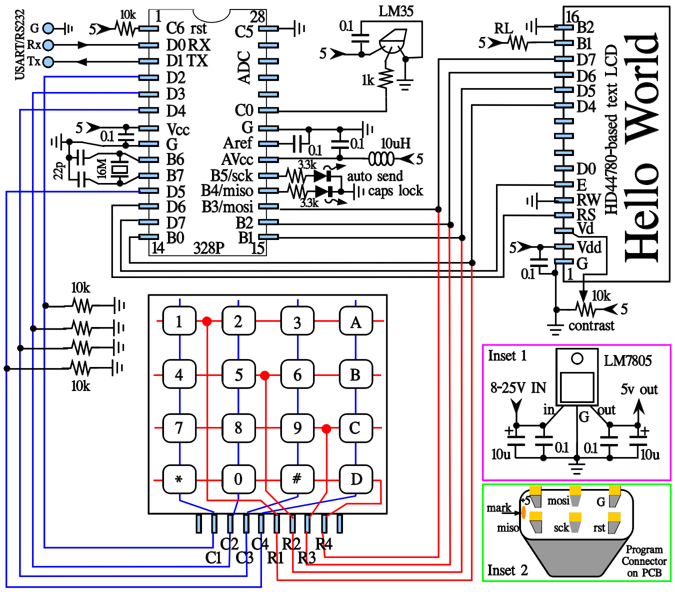


Figure 10.2: The controller for the FE5650A Rubidium Standard consists of a Microchip Atmel micro-controller ATMEGA328P (28pin DIP), a 4 x 4 keypad, an HD44780-based 16x2 text display (Adafruit AD181), a LM35DZ temperature sensor, and a LM7805 5v regulator along with miscellaneous components. The 328P uses a 16MHz crystal for its clock and 10 uH coil with a 0.1uF capacitor to filter the ADC power. The ADC uses the internal 1.1V bandgap reference with the 0.1uF capacitor from Vref to ground. Pin C0 is the ADC input from the temperature sensor. The LCD uses only the upper 4 data lines D4-D7 and the RW line is tied to ground since data will only be written to the LCD. Notice the keypad, LCD and the programmer share some 328P pins.
Inset 1 shows the LM7805 voltage regulator circuit. *Inset 2* shows the programmer connector soldered to the PCB. The connector pins should be directly wired to the 328P pins with corresponding labels.

The **keypad** is one of the common 16 key units similar to the one from Amazon B07B4DR5SH: [16 Keys Matrix Keypad 4 X 4 Membrane Keyboard Module Array Switch for Arduino UNO]

The figure shows that Row R1 has keys 1, 2, 3, and A while Row R2 has 4, 5, 6, B and so on. Similarly, Column C1 has keys 1, 2, 7, and *, while column C2 has keys 2, 5, 8, and 0, and so on. The idea (an oldie but a goodie) is to apply a voltage to one of the rows, and if a voltage appears at one of the columns then the microcontroller can determine the pressed key. For example, if the microcontroller momentarily applies a voltage to row R2 and a voltage appears on column C1, then a key is pressed and that key resides at the intersection of R2 and C1 which is '4'. The microcontroller sequentially sends a few-mSec-wide pulse to each row until a voltage is sensed on a column. The microcontroller incorporates delays since pressing a key can cause key-bounce whereby the key pads repeatedly make/break contact on the mSec timescale even though the key was only pressed once. Without proper conditioning by software (i.e., delays), the bounce would be interpreted as multiple key presses. The microcontroller scans the voltage pulse across the rows. The column lines are tied to ground through the 10k resistors on the lower left side of the figure. When the switch is open (i.e., not pressed), the microcontroller will register the ground logic level at pins D2, D3, D4 and D5. The keypad rows share the same microcontroller pins as the data lines for the LCD. The 328P disables the LCD by using the LCD Enable E line when it scans the keyboard. Even when the LCD is enabled and a key is pressed, the current through the 10k pull down resistor has negligible effect on the voltages at the LCD. Finally, two of the keys provide a Caps Lock and an Auto Send function which must retain their state between key presses. The LEDs connected through 3.3k resistors to pins 18 and 19 (i.e., B4 and B5) indicate Caps Lock and an Auto Send mode.

The LM35DZ temperature **sensor** connects to one of the 328P ADC inputs (C0 in this case). The voltage on the LM35DZ output (middle lead) increases by 10mV for every degree centigrade increase in case temperature. In particular, the relation between temperature and output voltage follows the relation

$$C = mV / 10 \tag{10.2}$$

where mV is the output from the LM35DZ in millivolts and C is the temperature in degrees centigrade. The data sheet shows approximately 0.5C accuracy. The LM35DZ should be mounted to the physics package surrounding the rubidium cell in the FE5650A. If the microcontroller is in an enclosure external to that of the FE5650A such as shown in the bottom panel of Figure 10.1, then a length of cable will be required between the LM35DZ and the microcontroller enclosure. While the LM35DZ can drive some capacitive loads, the construction should include a 1k-3.3k resistor on the output to help eliminate any instability in the LM35DZ output (refer to Section 10.4). Notice the figure shows the correct LM35DZ orientation for the various connections.

The 328P has a 10-bit **ADC** which measures the voltage provided by the temperature sensor. The 10-bits divides a reference voltage Vref into 2^{10} = 1024 'bins' or 'steps' or often called 'bits'. We use the internal bandgap reference of Vref=1100mV which means each bin will be approximately 1mV. The ADC should be checked for calibration but for the present application, an offset of 5-10bits only translates to 5-10mV which in turn translates to 0.5-1C at the temperature sensor. The relation between the bits, Vref and mV can be found in the data sheet for the 328P

$$A_{bits} = \frac{V_{IN} 1024}{V_{ref}} \tag{10.3}$$

where A_{bits} is the ADC results in bits, V_{IN} is the mV at the ADC input (pin C0 in this case) and Vref=1100mV for the internal bandgap reference. Equations 10.2 and 10.3 can be combined to provide

$$C = \frac{mV}{10} = \frac{A_{bits} Vref}{10240} = 0.1074\, A_{bits} \tag{10.3}$$

where C is the temperature in degrees centigrade. The ADC is powered from the AVcc pin 20 which should be filtered through a 10uH coil and 0.1uF capacitor to ground.

The 328P itself uses a 16MHz **crystal** and 22 pF capacitors for its clock connected at pins 9 and 10 (i.e., B6 and B7); the leads for these pins should be kept short.

Inset 1 for Figure 10.2 shows an LM7805 regulator that supplies 5v for the 328P, LCD and LM35DZ. Some 5v regulators do not require all of the capacitors but we tend to use them to help filter noise. The regulator requires an input voltage larger than about 7 VDC and so the obvious choice routes the 15V from the power supply in the FE5650A Enclosure through a cable connecting the two enclosures. It might be possible to use the 5V from the FE5650A Enclosure through a cable between the two enclosures provided the voltage drops no more than a couple tenths volt. Regardless of the configuration, the DC level should be checked for stability.

We chose to construct the controller in a Remote Enclosure that uses the 15V power from the FE5650A enclosure or an internal 9V alkaline battery. Figure 10.3 shows the circuit that incorporates the

LM7805 from Inset 1 of Figure 10.2. The fuse was not included in the final circuit but would definitely be a good idea to limit the damage caused by accidental short circuit. It should be possible to reduce the fuse amperage. The two diodes form a type OR gate whereby the regulator derives power from the 15V source regardless of whether or not the 9V battery is connected. If the 15V power is not connected then the 9V battery can supply the power. The diode associated with the 9V battery also prevents damage to the 328P and LCD if the battery is momentarily connected backwards. Notice that two different diodes are specified [10.18-19].

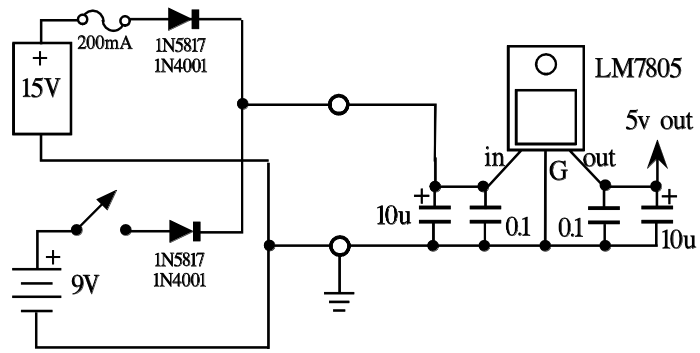


Figure 10.3: The 5V regulator for the Atmel MEGA 328P related circuits derives power from either a 9V battery or the 15V power supply associated with the FE5650A. The fuse can be left out of the circuit if desired. The diodes allow the 15V to power the circuits regardless of whether the 9V battery connects to the regulator.

The 1N4001 diodes have a forward drop of approximately 0.7V and the LM7805 must have at least 7V on the input pin 'in' to provide proper regulation on the output pin 'out'. Consequently, the 9V battery can discharge to 7.7V and still properly power the regulator. The 1N4001 diode allows negligibly small reverse bias currents (less than 1uA) and can withstand reverse bias voltages of up to 50V. All of the voltages for the controller are less than 15V. In addition, the alkaline 9V battery can safely handle 2uA of reverse bias without damage [10.20]; the 1N4001 has reverse bias leakage well below the 2uA [10.19]. We used the 1N4001 diodes for our final design.

The Schottky diodes 1N5817 [10.19] can be used instead of the 1N4001 as their reverse leakage current is 1.7uA and 2.3uA at the reverse voltage of 5 and 10V, respectively, as measured in our lab. The maximum reverse bias that can be applied is 20V. The 1N5819 might be a better choice given their lower leakage. Be aware that higher temperatures produce more leakage. The forward voltage drop is only about 0.25 volts which means the battery can discharge to 7.25V and still properly power the regulator (i.e., the battery "lasts longer" than for the 1N4001 diodes).

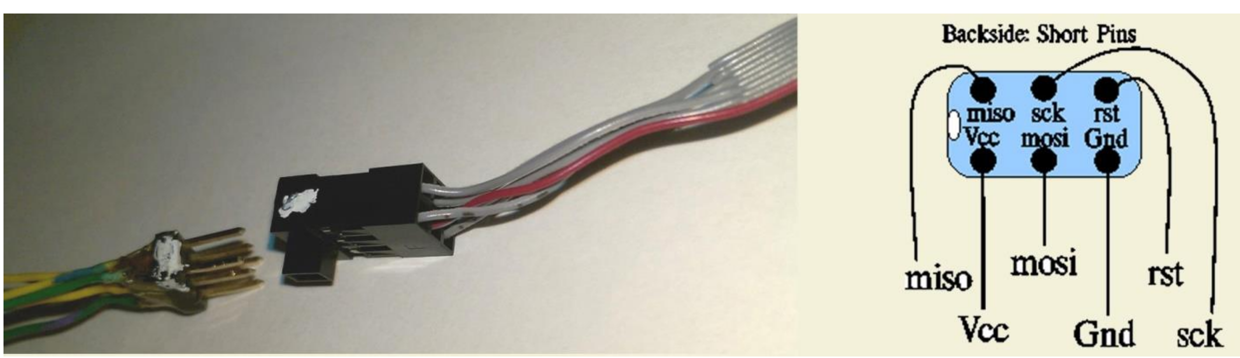


Figure 10.4: Left: The double row male header positioned next to the double row female receptacle for the Microchip-Atmel programmer. Right: View of the bottom side of the double row male connector (i.e., short pin side). The labels indicate the required connections to power and the microcontroller (Vcc = 5V and Gnd = ground = 0).

The ICE (or other) ATMEL programmer connects to the circuit through a 6-pin section of double row male header such as these:

Male Header Double Row 2.54mm (0.1”) Amazon.com B00G9Q7KIE

Refer to Reference 10.2 for more information on fabricating the connector. Several options can be imagined for the header. As a first one, the short pins on the header can be soldered into the PCB so that the 328P can be programmed in-situ. This can be convenient when still in the process of developing the circuit and software. Plus it makes it easy to make changes to the 328P programming. Insert 2 of Figure 10.2 and the right panel of Figure 10.4 show two different views of the same 6pin double row header connector. When soldering the connector to the PCB, the short side of the pins should be inserted into the PCB and soldered. The connector pin labeled as Vcc should connect to 5V, and the one labeled as G should attach to ground. The other labels on the connector pins indicate to which uC pins the connector pins should attach. For example, the one labeled as ‘miso’ should be wired to 328P pin #18 (i.e., B4/miso).

As another option that reduces soldering and wiring and saves PCB real-estate, the programming connector can be left out of the final design. Instead, for programming, the 328P can be placed in a Zero Insertion Force ZIF socket, programmed and then transferred to the final controller PCB.

Zero Insertion Force ZIF socket, 28pin Amazon.com B072KYHNV9

The ZIF can easily be mounted to a solderless white experimenter’s breadboard in order to program the 328P external to the final PCB.

Experimenter Breadboard, solderless, prototype Amazon.com B01EV6LJ7G

The experimenter’s board should include several other components. First, a 5V LM7805 similar to the one in Insert 1 should be included to supply power. A 9V battery can provide the input power for limited durations but it is better to use 9V or 12V adapters. Second, a 16MHz crystal and associated 22pF capacitors can also be included on the breadboard. The 328P does have an internal clock consisting of an RC oscillator (but this is not accurate enough for USART communications). Without the 16MHz crystal, once the programming changes the clock fuse (see ensuing sections), the RC oscillator will no longer operate and the 328P will not properly operate with other possible circuits [10.6]. The left panel of Figure 10.4 shows wires soldered to the short pins of the double-row header and covered with epoxy to increase durability and electrical insulation. The other end of the wires can be attached to the breadboard or ZIF pins for the programming. Finally, include the 10k resistor from 328P pin 1 to +5V.

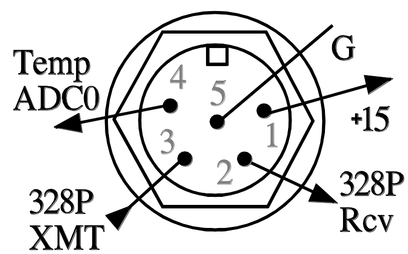


Figure 10.5: Backside view of the 5-pin mini-XLR connector in the Remote Enclosure.

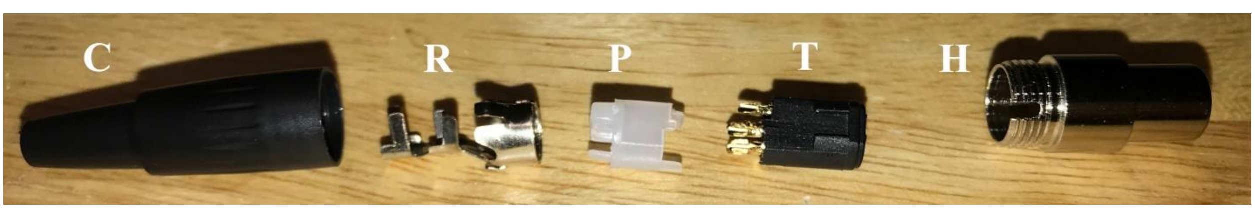


Figure 10.6: Parts of the 5 pin mini-XLR plug. A cable passes through the cap C, metal retainer R, plastic insulator P, and solders to the plastic terminal T. After pressing P into T and then R onto P, the left side tabs of the retainer can be bent over to clamp the cable. The housing H can then be placed over the R-P-T assembly and the cap C can be screwed onto the housing H.

The remote and FE5650A enclosures were interconnected with what looked to be an old keyboard cable (approx. 2 meters long, not the coiled type), which had four insulated wires and a bare ground wire. On either end, a 5 pin mini-XLR was added. The plug components appear in Figure 10.6. Remember to place the cap C, metal retainer R, and plastic insulator P on the cable prior to soldering the terminals T. Once soldered, the insulator P can be mated with the terminals T and then the retainer R pressed onto the insulator. Clamp the rear tabs of the retainer onto the cable. Push the RPT assembly into the housing H and then screw the cap C onto the housing.

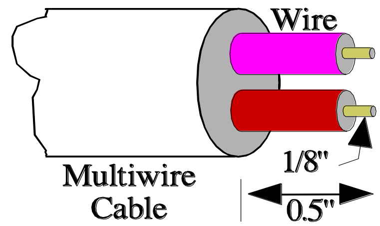


Figure 10.7: Example preparation of a multi-wire cable for the mini-XLR plug.

Figure 10.8: Example of the Remote Enclosure.

We found the cable should be prepared as illustrated in Figure 10.7. The outer cladding should be removed ½ inch from the end and then each wire insulation stripped by 1/8" from the end. If the ground wire is bare, then add some heat shrink to help insulate it. We also found that our cable did not fit into the cap because of a rigid plastic piece that forms part of the cap and cannot be removed. Reeming out the plastic with a drill bit (using a drill) sufficiently enlarged the hole. The cable cladding also needed some slight tapering and part of the strain relief on the cap needed to be cut off.

An example for the Remote Enclosure can be seen in Figure 10.8. The keypad shows the numeric and directional labels. The LCD shows the main menu at powerup. Notice the LED next to Caps Lock and another next to Auto Send. The switch and cable connector have been placed on the upper side next to the LCD. The LCD is much brighter than shown in the figure.

Section 10.3: Modifications and Additions

The FE5650A circuits need final modifications and additions regarding the serial communications, temperature sensor, and the mini-XLR connectors.

Topic 10.3.1: TTL-RS232 Modifications

The TTL-RS232 protocol for the ATMEGA328P USART uses Vcc=5V as the idle/standby voltage for the Rx and Tx lines. The bits for the ASCII code are sent as 5V for logic 1 and 0V for logic 0 [10.21]. Contrarily, the traditional RS232 sets idle as a negative voltage (-12V, but usually low current). The bits for the ASCII code are sent as a positive voltage +12V for logic 0 and -12V for logic 1. Notice the logic levels are inverted in addition to the differing voltage range. Recall from Chapter 6, the FE5650A converts between the TTL and the traditional RS232 protocols using the SP233 IC. Further as discussed in Chapter 6, should you decide to retain the traditional RS232 protocol for use with the ATMEGA328P, your unit will need to incorporate a circuit similar to the LSLI circuit (Level Shift, Level Invert) or an IC that performs a similar function. The traditional voltage range of -10 to +10V can/will damage the Microchip-Atmel 328P uC in addition to presenting inverted logic levels.

Another point worth mentioning is that the PIC16F84 on the FE5650A DDS board does not have a USART; instead the FEI designers used pins 8 and 6, respectively, to transmit and receive ASCII characters. Apparently pin 6 has an associated hardware interrupt for the receive operation. The PIC implements the USART functions in code; this coding probably explains the range of baud that can be received but not transmitted (see the encryption problem in Chapter 6). The receive Rx function can have a range of baud by simply reading a bit multiple times fast enough so as to detect a received 1 or 0. The transmit Tx does not have such freedom since the simplest PIC routine derives one fixed baud rate from the single clock rate.

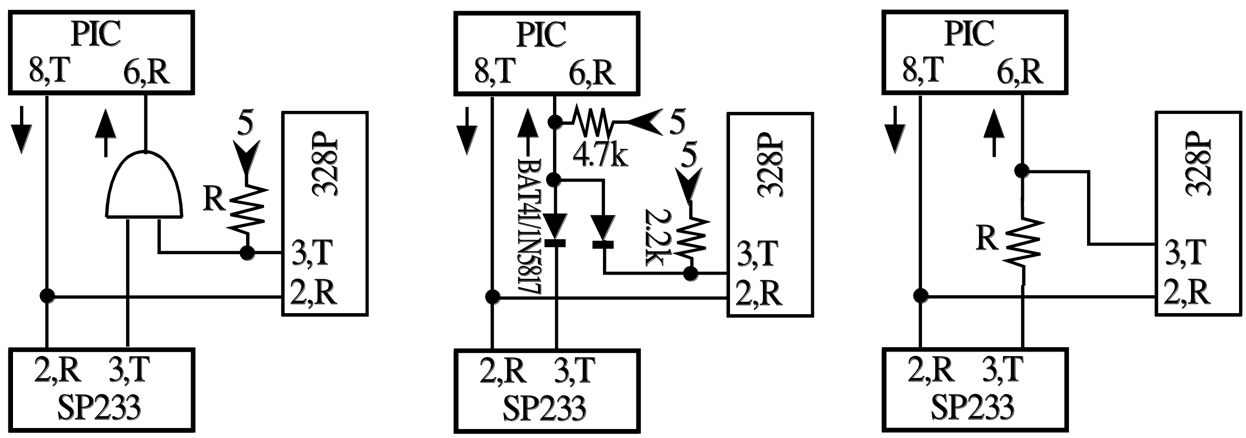


Figure 10.9: Possible methods for splicing the MEGA328P USART into the FE5650A circuit which consists of the embedded microcontroller PIC16F84 and the RS232 converter SP233. The notation 8,T and 6,R refers to transmit pin 8 and receive pin 6, respectively, on the PIC, and 2,R and 3,T refers to the receive pin 2 and transmit pin 3 on the SP233. Left: An AND gate routes the signal from either the SP233 or the Atmel 328P. The resistor R places logic 1 at the AND gate input when the 328P is not connected. Middle: Another version for the AND function. Schottky diodes are preferred although silicon 1N914 or germanium 1N34A diodes would work too. Right: The simplest case where the resistor sets higher input impedance (compared with the direct connection). The Left and Middle circuits have not been tested. We used the Right circuit with R=3k but R=2.2k should be ok. In all three cases, the 3rd pin of the SP233 needs to be disconnected from the FEI circuit board (or else #6 on the PIC) in order to insert the extra components.

Chapter 6 suggested directly connecting the SP233ACT pins 2 and 3 to the Atmel chip. While this worked in all of our test cases, it was definitely not good practice. In particular, the PIC16F84 receiver (pin 6) connects to the SP233 transmitter (pin 3). The direct-connect method would then connect the Atmel328P transmitter (pin 3) to the transmitter of the SP233 (pin 3). Such a direct connection causes several problems. First, the actual logic state can be uncertain when for example, the Atmel transmits zero volts and the SP233 transmits +5V. Second, during the contention condition, the current flow might be sufficient to damage a port. And third, the Atmel 328P transmitter pin can partially power the PIC16F84 and the digital circuits in the FE5650A even though the dedicated FE5650A power supplies are OFF. As it turns out for the separate enclosures, this partial power problem was not remedied (and didn't need to be remedied) since the interface cable powered the remote and so the remote battery would never be 'electrically' connected to FE5650A.

The FE5650A circuit board holding the PIC16F84 and SP233 (i.e., the AD9830A board) needs to be modified. It would be useful to allow either the TTL or traditional RS232 to be used when needed. The best procedure would incorporate a TTL AND gate (CMOS would probably be ok too) as shown on the left side of Figure 10.9 so that either the SP233 or the Atmel328P could send ASCII bits. Notice the modification consists of an AND gate (as opposed to OR) since the idle voltage is +5V and if either the 328P or the SP233 produce 0V then the logic gate will transmit that level to the PIC16F84. The resistor R ensures that the input to the AND gate will be +5V even when the 328P is not present. The middle panel shows a circuit using diodes rather than the full logic gate. Neither of these circuits has been tested as there's a simpler solution.

For the present purpose, the simplest solution appears in the right hand portion of Figure 10.9. A small-size 2.2k resistor is sufficient to prevent the contention issue. We a 3k resistor with quite small physical size that we had on-hand (1/4W, 1%, metal film) but a 1/8 W 2.2k resistor should have small enough dimensions.

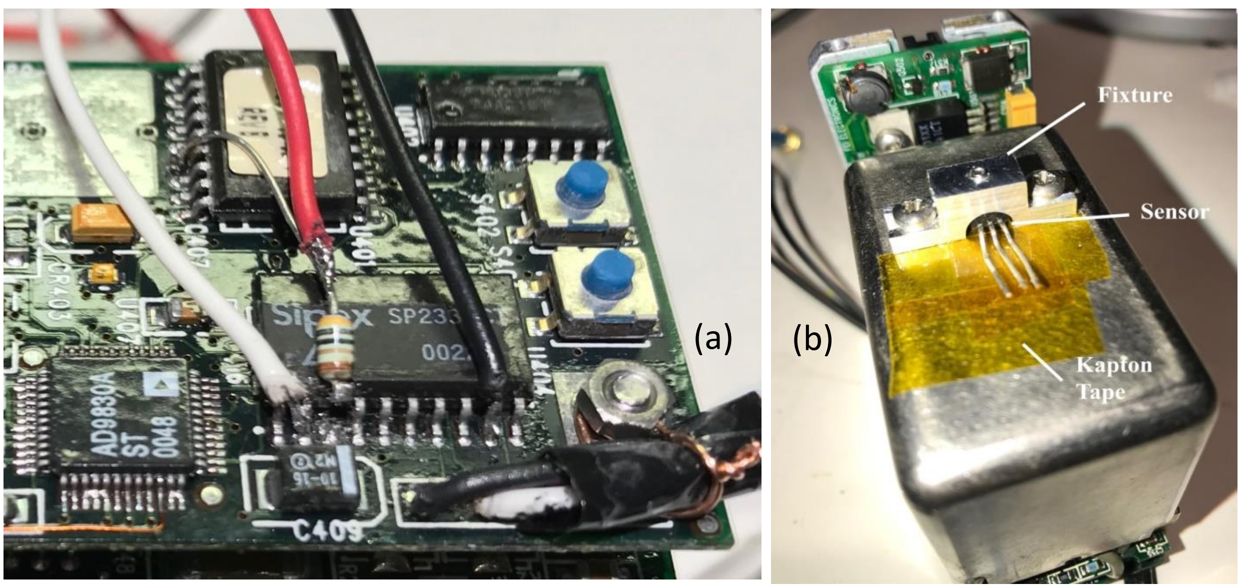


Figure 10.10: (a) Lift SP233 pin 3, solder a resistor to that lifted pin and solder the other end of the resistor to PIC16F84 pin 6. The red wire soldered to the resistor receives data for the PIC microcontroller. The white and black wires connect to SP233 pins 2 and 9, respectively, same as in Chapter 6. (b) The fixture, which is cut from aluminum, holds the temperature sensor in thermal contact with the FE5650A physics package. Two existing screws secure the fixture. A set screw at the top holds the LM35DZ sensor to the underlying metal platform.

Figure 10.10 shows pin 3 of the SP233 has been unsoldered from the PCB and lifted to the top side – be careful not to fatigue the metal pin to where it snaps off the IC. One end of a small-sized 3k resistor is soldered to the lifted pin and the other end to pin 6 on the PIC16F84. Note that PIC pin 6 was not unsoldered. The red wire receives TTL-RS232 data from the Atmel 328P for the PIC16F84. The white wire transmits data from the PIC16F84 to the 328P and the black wire provides the ground reference.

Topic 10.3.2: Temperature Sensor

An LM35DZ temperature sensor [10.22] can be placed in thermal contact with the physics module of the FE5650A and it can be powered from the 5V supply in the enclosure. For mounting, a small piece of aluminum is fashioned into a fixture to hold the temperature sensor in contact with the package as shown in Figure 10.11 and also Figure 10.10(b). The fixture makes use of the two 4-40 screws on the top of the cover of the physics package. Notice the tapped hole on top of the fixture with a set screw to secure the sensor. Strips of Kapton Tape electrically insulate the sensor leads from the package cover (the Kapton withstands temperatures to 200C). Figure 10.11 shows an additional small circuit board mounted across the two rear holes behind the power supply PCB. A connector on the board secures the power supply and sensor output wires. Although not shown, a 0.1uF capacitor attached from LM35 Vcc to ground.

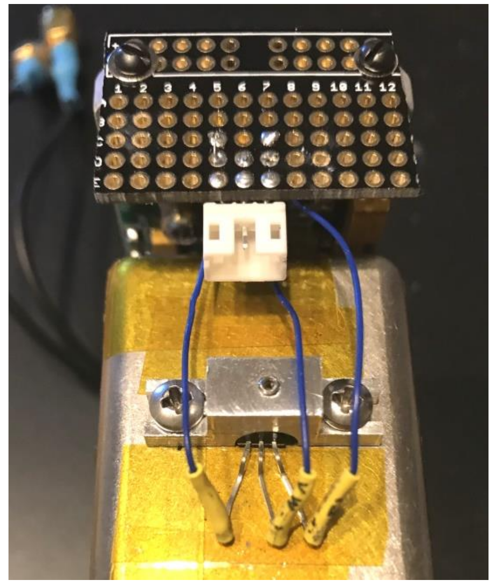


Figure 10.11: The fixture for the LM35DZ and piece of circuit board to hold the connector.

The LM35DZ was initially tested in the circuit shown in Figure 10.2 without any resistor on the sensor output and using wires with lengths of a couple inches. The LM35DZ agreed very well with a K-type thermocouple. However, once the 6 foot long cable was attached between the two enclosures (and without the series resistor), the LM35DZ output mismatched the k-type probe by 3 or 4 C. Attaching an oscilloscope showed the output to have a waveform similar to a lopsided triangle wave with peak-to-peak amplitude of about 400mV, period of 21uS, and rise time of approximately 2.5uS. Figure 10.2 shows a 1k resistor in series with the output of the sensor but we tried a small (about 3-4mm long) 3k resistor soldered to the output lead of the sensor prior to the cable. The resistor eliminated the 'triangle wave'. The sensor output with the 3k resistor agreed with the k-type thermocouple to within 1C. It might seem odd that a 3k resistor works if one anticipates low input impedance for the MEGA328P ADC. However, the ATmega328P data sheet and reference [10.13] indicate the 328P input has been optimized to handle sources with output impedance up to 10k [10.22].

Once the enclosures were assembled, we found the temperature on the physics package depended on position (as tested using a k-type thermocouple) and differed by a couple of degrees.

Topic 10.3.3: Mini-XLR Connectors

Although not a modification to the FE5650A per se, the enclosure needed a 5-pin mini-XLR connector for the multi-wire interface cable. The enclosure has both a 4pin (for the standard TTL interface as per Chapter 6) and the 5-pin for the TTL-RS232, power and the temperature sensor output (see Figure 10.12). Here we show the pinout for our case – yours might differ. Actually, a 3 pin connector would be preferred over the 4 pin. One needs to make sure that the +15V can never access the 4 pin connections – careful.

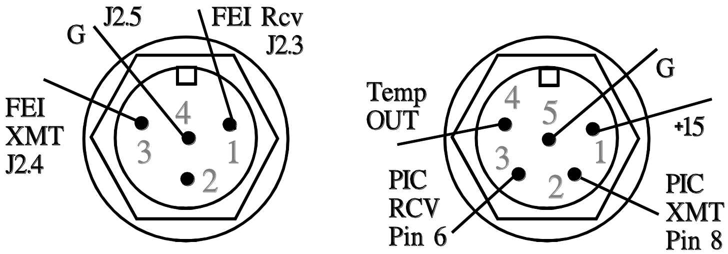


Figure 10.12: Mini-XLR connectors in the 'FE5650A Enclosure'. The 4-pin connector carries the standard RS232 signals. The 5-pin connector carries the TTL-RS232 signals along with power and the output from the temperature sensor.

The enclosure includes both connectors so that either the standard RS232 or the TTL RS232 can be used, but not both at the same time. The standard RS232 can be used with a USB-RS232 adapter for connection to the PC while the TTL-RS232 connects to a uC. Chapter 6 describes the connections necessary for the standard RS232.

Section 10.4: Menus and Basic Procedures for the FE5650A Interface Program

The ATMEGA328P microcontroller (uC) executes a program residing in nonvolatile flash memory. Often such programs are termed 'firmware'. Sometimes, because the flash memory can be so easily reprogrammed, people refer to the code as 'software' even though that term generally applies to a collection of programs and data residing in the volatile random access memory (RAM). The present section describes the use of the program for controlling the FE5650A in conjunction with the previously discussed circuitry and modifications. The 328P should be programmed as described in the ensuing sections and appendices, or a programmed chip should be obtained. Subsequent sections and appendices provide the source code and discuss the programming. The topics below first outline the various menus in the program and then provide basic procedures. Additional code can be added to the source listings and programmed into the ATMEGA 328P uC for user defined options.

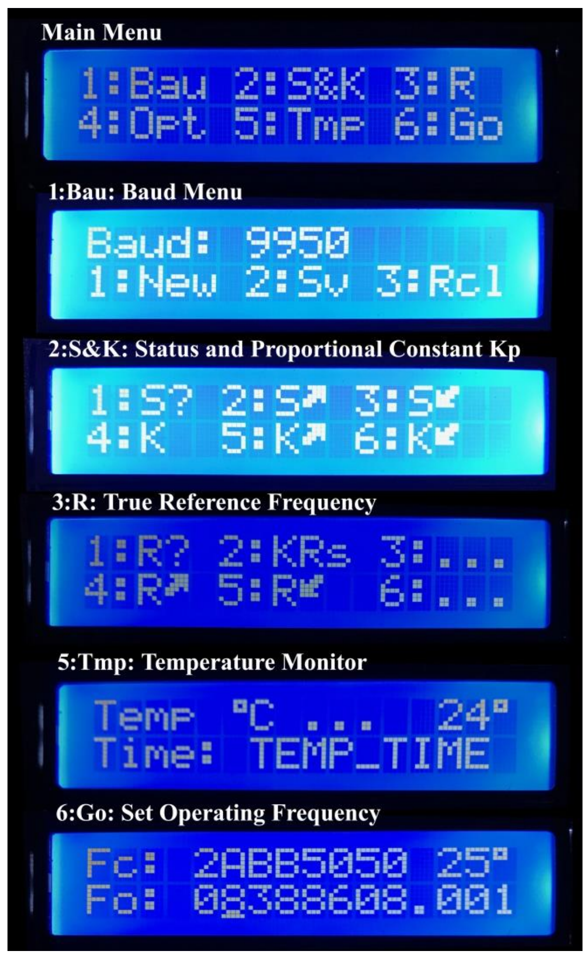


Figure 10.13: Menus in the FE5650A program

The available menus appear in Figure 10.13 and the keypad in Figure 10.14. At startup, the 'main menu' (Figure 10.13) shows on the controller LCD. The user engages the controller by the pressing keypad #6 button to select the 'Go' menu which provides an interface to change the output frequency (see menu #6 in the figure). The toggle key switches between the LCD top line (Fcode) and the bottom line (Fout). By setting 'caps lock', the left and right arrow keys can be used to select a digit to the right or left as indicated by the 'underline' cursor (menu #6 in the figure shows the 8 in Fo has been selected). Once a digit has been selected, the keypad 'up' and 'down' arrow keys change the digit by one step. The FE5650A output frequency Fout can be updated by pressing the OK/Enter/Cr key after entering a new frequency or the ATMEGA328P circuit can automatically update it after each digit change by setting the 'Auto Send' key. Note that the Go Menu shows the physics package temperature in Celsius at the end of the top line.

Figure 10.14: The keypad with labelled keys. The top LED indicates 'Caps Lock' and the bottom LED indicates 'Auto Send'.

A couple of notes are in order. The program is an alpha version in the sense that the menus are not as complete or as easy to use as they could be made but the major bugs have been exterminated so that the program operates with the RFS unit. We did notice one problem that will be commented upon more further down. Sometimes in the Go Mmenu #6, the updating of the temperature interferes with the keypad operation. The user might want to increase the delay time in the program for the temperature to display or more slowly press the keys. We make this comment even though the software might be modified from that described here. As another comment, the baud rate has been set to 9950 although it can be changed. The user can exit the menus by pressing the menu key. Future versions would incorporate microcontrollers uC with more memory and LCDs with more possible lines of text (or a graphics LCD with touch sensitive screen).

With the power applied to all circuits, the ATMEGA328P microcontroller uC recalls previously saved data from eeprom nonvolatile memory including the baud rate, True Reference Frequency Rtrue (i.e., R), the Stored Reference Frequency Rstored, the Stored Hex Code Fstored, the initially used Hex Code Fcode, the corresponding initial Output Frequency Fout, and a constant Kp to convert from the Stored Reference Frequency to an estimated true reference frequency as Restim = Kp Rstored (refer to Chapter 9). Each FE5650A has a stored reference frequency and Fcode.

At startup, after initializing the various modules of the uC, the program shows the Main Menu that appears at the top of the menu list in Figure 10.13. The Main Menu shows the following options.

Number	Prompt	Expanded	Description
1	Bau	Baud	Set the baud rate
2	S&K	Status and Kp	Status enquiry and set Kp
3	R	Reference Frequency	Reference Frequency menu
4	Opt	Options	Does nothing at present
5	Tmp	Temperature	Continuously monitors temperature
6	Go	Frequency	Adjust output frequency

The user should first activate the S&K menu and select the Status Enquiry 'S?' option in order to test the communications. If the controller microcontroller (uC) does not receive a response of any kind, the LCD will show 'no comm' so the user should check cables and power. It can also happen that the baud is so far wrong that the FE5650A doesn't return any ASCII code at all. On the other hand, if the FE5650A recognizes the status inquiry, it will return some ASCII but if the baud rate isn't within an acceptable range for the uC, the LCD will show 'Xmt Err/Encrypt'. In such a case, the user should change the baud similar to that discussed in Chapter 6 using the Baud Menu #1 on the controller rather than the PC with the terminal software.

Consider the operations of the various menus presented on the Main Menu at startup:

The **Baud Menu** can be activated by selecting ' 1:Bau '. The Baud Menu used to set and save a baud rate. The uC USART baud rate will be updated when the user exits the Baud Menu.
1. The 'new' command allows the user to enter a new baud rate which updates the current value of the 'baud' variable.
2. The ' 2:Sv ' option saves the new baud rate to 328P eeprom so that the uC will use the new value next time it starts.
3. If for some reason, the eeprom value needs to be retrieved from 328P eeprom, use the '3:Rcl' option.

The **Status Menu** can be activated by selecting '2:S&K'. The primary purpose is to access the FE5650A parameters Rstored and Fstored and enable changes to the constant Kp. The Rstored and Kp values allow the user to estimate the reference frequency Restimate = Kp * Rstored rather than going through a calibration process.
1. The option '1:S?' queries the FE5650A for the stored reference frequency, which appears on the LCD top line, and the stored power-up Fcode, which appears on the LCD bottom line. It is a good idea to write these in a notebook in case they need to be reentered someday. This option allows the user to quickly connect to multiple FE5650A units, obtain their stored reference frequencies and with other menus (KpRstored), quickly set an output frequency accurate to within about 0.01Hz.
2. The option ' 2:S↗ ' saves the stored reference frequency Rstored and the stored hex Fcode in the 328P uC eeprom memory. As a note, the global variables for the current session can only be updated by using option 3 below.
3. Option ' 3:S↙ ' recalls the values of Rstored and Fcodestored from the 328P uC eeprom and saves to the global variables Rstored and Fstored for use with calculation of 'Restim = Kp Rstored' during the current session if desired. Because Rstored and Fstored have been saved to 328P eeprom, the next session (power down, power up) with the 328P, the global variable will be initialized from eeprom to saved values of Rstored and Fstored. Use this option to update the current global variables for Rstored and Fstored.
4. Option ' 4:K ' allows the user to enter a new constant Kp for the relation Restimate = Kp Rstored. The estimated true R is stored in the global variable Rtrue and is used for all output Frequency Fout calculations. The new constant Kp is saved to the global variable Kp immediately after it's entered.
5. Option ' 5:K↗ ' saves the new constant Kp to 328P eeprom nonvolatile memory. The next time the 328P is powered, the global variable Kp will be initialized with this value from memory.
6. Option ' 6:K↙ ' recalls a constant Kp from 328P eeprom nonvolatile memory and places it in the current global variable Kp.

The **True Reference Frequency Menu** (3:R) allows the user to enter a true reference based on calibration or to use the estimate Restimate=KpRstored for calculations of the output frequency. The value of Rtrue is used to calculate the output frequency Fout according to

$$F_{out} = R_{true}F_c/2^{32}$$

as described in previous chapters.

1. The option ‘ R? ‘ returns the currently used value of Rtrue and allows the user to enter another value. The program immediately updates the global variable Rtrue used for calculations.
2. The option ‘ KRs’ uses the currently available values of Kp and Rstored (global variables) to estimate the true value of the reference frequency Rtrue ~ Kp Rstored. Notice any calibrated value Rtrue will be overwritten.
3. Option ‘3:’ does nothing … available for development.
4. Option ‘ R↗ ’ saves the current value of Rtrue to ATMEGA328P eeprom nonvolatile memory. If the 328P is powered down and then powered up at a later time, this value of Rtrue will be used.
5. Option ‘ R↙ ‘ retrieves the reference frequency stored in 328P eeprom and places it in the global variable Rtrue for calculations
6. Option ‘6:’ does nothing … available for development.

The ‘**5:Temp**’ Menu continuously monitors the temperature measured in degrees centigrade. The value updates every four seconds. Keep in mind that the FE5650A should be operated below 60C.

The ‘**6:Go**’ menu provides the primary functionality for the program to change the output frequency. By setting ‘caps lock’, the left and right arrow keys select a digit to change, and the up and down arrow keys change the digit by one step. The toggle key switches between the LCD top line (Fcode) and the bottom line (Fout). The FE5650A can be updated by either of two methods: (1) once the Fout has been changed to the desired final value, press OK/Enter/Cr key; (2) press ‘Auto Send’ so that each time one of the digits changes, the 328P will send the modified Fcode to the FE5650A. The Auto Send mode facilitates using the unit for calibrating equipment since the change in output frequency will immediately be viewable on external equipment. The bottom panel of Figure 10.13 shows an example for the ‘Go Menu’ display. The cursor location is marked by the underline. The toggle key will move the cursor between LCD lines. Any change in either number will immediately update the other even if the Auto Send has not been activated but without the Auto Send, the new numbers will not be sent to the FE5650A. Finally, notice the temperature at the right hand side of the LCD (degrees centigrade).

Section 10.5: Programming the ATMEGA328P

For the controller to operate, it will be necessary to have a programmed ATMEGA328P microcontroller uC. It would obviously be easiest to purchase a preprogrammed uC [10.1] and simply insert it into the circuit; however, that requires the circuit to use the same keypad and LCD as specified in Section 10.2. Without such a preprogrammed uC, it is necessary to transfer code to the uC and to program fuse bits (i.e., fuses). Both procedures require a programmer and Atmel Studio. Recently, Atmel Studio, starting with 6.2, has made it possible to load a single production file (a.k.a., the ELF file) to program the FLASH (program memory), EEPROM and FUSES [10.25]. However, we will not consider the ELF file beyond using it to program flash if desired.

Assuming a preprogrammed ATMEGA328P is not available, the fuses must be separately configured from the code programming. Once the fuses have been set, several possible methods can be used to transfer the code. Perhaps the easiest procedure would be to transfer the program HEX file. Next would be to load a Microchip-Atmel Studio 7 Solution into the Atmel Studio 7 (AS7) and then transfer the program to the 328P. Access to the source code makes it possible to modify the program as needed. The next possibility would be to either cut-and-paste or else copy-by-hand the source code into the Atmel Studio. We have included the full source code in Appendix 6 in case changes need to be made or perhaps the easier options are not available.

The present section first sets up the Atmel Studio 7 and programmer, it next describes a method of setting the fuse bits in the ATMEGA328P using the Atmel Studio 7 and the Atmel ICE programmer (or equivalent). Next the section discusses the use of the AS7 Solution to transfer code in the form of HEX code, which resides in a file ending in '.hex'. Similarly the ELF file can be used to program the flash. Next will be the procedures involved with using a ready-made Solution or copying the source code and User Libraries.

Topic 10.5.1: Programmer and Atmel Studio

If you do not have the Atmel Studio 7 Solution for the FE5650A Interface program, it will be necessary to set up a Solution in order to program the FLASH program memory and the fuses – assuming you don't have a pre-programmed ATMEGA328P microcontroller (uC). The fuses set certain options within the uC such as whether or not the circuit uses a crystal or a divide-by-8 prescaler.

Make sure you have downloaded Atmel Studio (the current version is 7 at the time of this writing) [10.14]. Make sure the download includes C++ and C along with support for other potential areas of interest such as Arduino. Further, obtain a suitable programmer such as the ATMEL-ICE [10.15]. Use the 6-pin socket and use the AVR side of the programmer. We have an older Atmel programmer ATAVRISP2 which works, but we have not tested the 'compatible' versions found at Amazon and Ebay that sell for about $35. The ones on Amazon claim they operate with Atmel Studio 6 and so, for the present purposes, most likely work with Atmel Studio 7. The first time the programmer is attached to the computer USB port, Atmel Studio will likely upgrade the firmware in the programmer. For those readers wanting more details, refer to References[10.2-8].

Topic 10.5.2: Setup AS7 Solution

The Atmel Studio must be configured for the entering/transferring code to the ATMEGA328P uC and for setting the fuse bits of the uC. Of course, none of this would be necessary for a preprogrammed uC. If you plan to transfer a completed solution then the file has already been set up for the proper names; however, it would be a good idea to check for the proper microcontroller and to also program the fuse bits if you are using a new, unprogrammed uC.

Place the ATMEGA328P into its socket. The circuit should have at least the programmer pins/socket, a 16MHz Crystal and two 22pF capacitors, and 5V power supply similar to that shown in Figure 10.2 – the LCD, keypad, and temperature sensor do not need to be present but it also doesn't cause a problem. Follow the steps:

1. Attach the Programmer to the 328P circuit and to the USB port of the computer.
2. Power the 328P circuit.
3. Start the Atmel Studio software on the computer.
4. Select 'Files' at the top menu and then 'New' and 'Project': Files>New>Project
5. Select: 'GCC C++ Executable Project' (do NOT select OK yet).
6. Place a check mark in the box for 'Create a Directory for Solution'

7. Enter the Name: 'FE5650A Ctrl'. Notice the Solution Name also changes.
8. Click Browse and navigate to a directory for your new solution. We used the following
   Documents/AtmelStudio/7/Solutions
9. Click OK to exit the dialog.
10. The dialog named 'Device Selection' will open. If it doesn't, don't worry, you will have a second chance – see the next steps 11 and 12 below. Select ATMEGA328P and click OK.
11. Locate the Menu Bar near the top of the Atmel Studio and select the following sequence
    Project>FE5650A Ctrl Properties
    where 'FE5650A Ctrl Properties' appears as the last item in the drop down menu box. The FE5650 Ctrl window appears.
12. Click 'Device' from the left hand menu and verify that the ATMEGA328P has been selected. If not, select it now.
13. Click 'Tool' from the left hand menu. Select your programmer in the drop down box such as the 'Atmel ICE'. Then select the 'Interface' ISP for the ICE. The ISP clock must be set to less than ¼ of the clock rate used. Until the fuse bits have been changed, use the 125kHz. After the clock fuse bit has been change away from the default RC clock to something faster, the clock rate can be increased if desired – but 125kHz is really fast enough.
14. Close the FE5650A Ctrl* window
15. Verify that the 'Solution Explorer' window/pane is visible on the right hand side of the Atmel Studio. If not, then from the top menu strip select: View > Solution Explorer
16. When finished, 'Save All' either from the File menu or by clicking the 'double diskette' icon in the Atmel Studio menu/tool strip.

Topic 10.5.3: Programming Fuse Bits

Once having completed Topic 10.5.2 on initializing the 'FE5650A Ctrl' Solution, the fuse bits can be programmed. These only need to be programmed for a new ATMEGA328P or one that has previously been used but for a different program. Make sure the ATMEGA328P has been placed in its socket and a 16MHz crystal with the 22pF capacitors are in place and 5V power. Attach the ICE programmer to the PC and to the uC circuit. Make sure the ATMEGA328P has power. Start the PC and start the Microchip-Atmel Studio and start the 'FE5650A Ctrl' program from the previous topic. Perform the following steps:

1. At the Atmel Studio menu strip at the top, select 'Tools > Device Programming'. The 'Device Programming' dialog opens. The following should appear in the boxes near the top of the dialog:
   Atmel-ICE ATmega328P ISP
   If not, then select them.
2. Click the 'Apply' button
3. Click the buttons to read the Signature and the Target Voltage. If an error occurs, try flipping around the 6pin connector from the ICE programmer where it plugs into the programming pins. The Target Voltage should read 5V within about 0.1 volts. The ISP clock should be 125kHz but if not, change it to the 125kHz and click 'Set'.
4. Click 'Fuses' in the left hand menu. Make sure the check boxes are set as follows:
   a. HIGH.RSTDSBL no check
   b. HIGH.DWEN no check
   c. HIGH.SPIEN check
   d. HIGH.WDTON no check
   e. HIGH.EESAVE check

f. HIGH.BOOTSZ use the default, boot flash…2048…
g. HIGH.BOOTRST no check
h. LOW.CKDIV8 no check
i. LOW.CKOUT no check
j. LOW.SUT_CKSEL Ext. Full swing crystal … 1k ck/14 … 4.1

The last line #j selects the clock. Once changed, the RC clock will no longer clock the chip and so, if the 16MHz crystal has not been attached, the chip will be 'bricked' which only means it no longer responds until the 16MHz crystal properly operates. If the uC becomes 'bricked', it is necessary to provide an auxiliary clock signal to one of the crystal pins until the fuses have been properly set – just be sure the auxiliary clock signal does not produce voltages less than zero or more than 5V. The reference [10.6] provides an example auxiliary clock if needed – no need to build the entire circuit though.

5. Click the buttons for Program, Verify, and Read. The dialog should show some 'OK's on the lower left.
6. Close the dialog
7. When finished, 'Save All' either from the File menu or by clicking the 'double diskette' icon in the Atmel Studio menu/tool strip.

Keep in mind that the fuse settings refer to a single ATMEGA328P and these settings are not stored in the Atmel Studio solution – each new uC needs to have the fuses set.

Topic 10.5.4: Option 1: Program the ATMEGA328P from the HEX/ELF file

Once the Atmel Studio and the fuses have been setup, the program code embodied within the HEX or ELF file can be transferred to the ATMEGA328P uC. Complete Topics 10.5.1 through 10.5.3 and then complete the following steps

1. Make sure Atmel Studio is running and the (empty) 'FE5650A Ctrl' solution is loaded. Connect the programmer to the ATMEGA328P and its circuit and connect the programmer to the PC. Power the 328P circuits.
2. From the Atmel Studio menu strip near the top, select 'Tools > Device Programming'
3. Click the 'Apply' button for the Tool, Device and Interface, and click the voltage/signature read buttons.
4. Select the 'Memories' option from the left hand menu.
5. The textbox under the word 'Flash' has a small button with ellipsis '…' to its right hand side. Click the button, navigate to the location of the HEX file (named 'FE5650A Ctrl.hex'), select it, and click the 'open' button. Note that the ELF file will work just as well here.
6. Click the 'Program' button. After a minute or two, the program should have completed. If the uC is attached to the full circuit with the LCD, the first menu should become visible. As a note, if the fuses have not been programmed in the uC, the menu might eventually show on the LCD but it will be running very slow.

Topic 10.5.5: Option 2: Manually Enter the Code into the ATMEGA328P

The 'manual' option requires the developer to either *very carefully* type lines of code one-by-one or have 'copy-able' text to paste into the previously prepared Atmel Studio Solution named 'FE5650A Ctrl'. Either way, the steps will be the same. The code for main.cpp and the user libraries can be found in Appendix 6. If you have the Atmel Solution 'FE5650A Ctrl' then check that the following steps have been completed; however, there isn't any need to copy code from the appendix.

First, it will be necessary to setup the empty library files and inform Atmel Studio as to their location. Much of the code has been placed in the library files not so much for easy reuse in other programs as is the usual reason, but because during the program development, the code could be moved so as to declutter the work area – out of sight, out of mind. There are seven libraries:

ADC328	Setup, functions for the ATmega328P Analog-to-Digital Converter
KeyPad4x4	Routines for the 16-key keypad
LCD16x2_ST7032	Setup and routines: HD44780-based LCD (two lines, 16 chars/line)
my_F64	64 bit floating point routines
StrNum	String and number routines
TC16	Setup and routines for the 16bit 328P timer counter
USART	Setup and routines for the 328P USART based TTL-RS232

As a note, the LCD16x2 library was named LCD16x2_ST7032 in anticipation of switching to the ST7032-based LCD; however, we never made the change but left the name as the original.

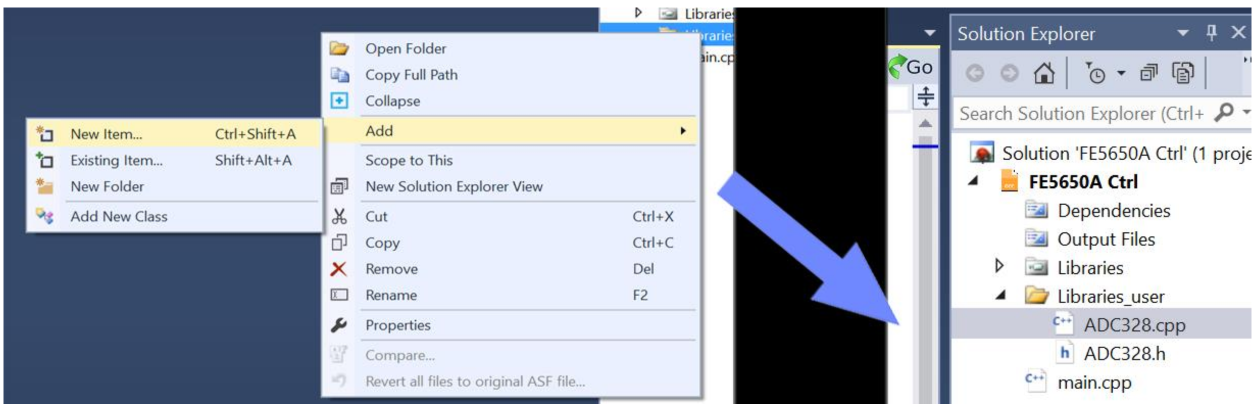


Figure 10.15: Right click the 'Libraries_user' folder, and then 'Add > New Item' and once having selected the .h or .cpp option, the empty file will be shown below the parent directory.

Add Empty Library Files: In order to setup the libraries, perform the following steps (see also Figure 10.15). Files ending with '.h' extensions are header files while those with the same name in the same directory but with the '.cpp' extensions are the implementation files.

1. Open the Atmel Studio Solution named 'FE5650A Ctrl'
2. Verify the 'Solution Explorer' appears on the right hand side. If not select the following sequence from the Atmel Studio menu strip 'View > Solution Explorer'.
3. Verify that 'main.cpp' appears under 'FE5650A Ctrl' … this will be used later.
4. Right click the line reading 'FE5650A Ctrl'
5. Select 'Add > New Folder' and name the folder 'Libraries_user'
6. Add empty .h files and .cpp files for these libraries. Start with ADC328:
    a. Right click 'Libraries_user' and from the menu select the sequence 'Add > New Item'
    b. Select 'include file' and at the bottom name it ADC328 (note the .h extension will be automatically added).

c. Click ‘Add’ and make sure the file ADC328.h has been added under the ‘Libraries_user’ folder.
d. Again, right click ‘Libraries_user’ and the sequence ‘Add > New Item’.
e. Select ‘CPP File’ and at the bottom, name it ADC328. If you do not include .cpp (lower case), Atmel Studio will automatically include the ‘.cpp’ extension to the name.
f. Click ‘Add’ and make sure the file ADC328.cpp has been added under the ‘Libraries_user’ folder.

7. Add empty .h and .cpp files for the other 5 libraries. Here’s another example for the LCD library pair.
   a. Right click ‘Libraries_user’ and from the menu select the sequence ‘Add > New Item’
   b. Select ‘include file’ and at the bottom name it LCD16x2_ST7032 (note the .h extension will be automatically added).
   c. Click ‘Add’ and make sure the file LCD16x2_ST7032.h has been added under the ‘Libraries_user’ folder.
   d. Again, right click ‘Libraries_user’ and the sequence ‘Add > New Item’.
   e. Select ‘CPP File’ and at the bottom name it LCD16x2_ST7032. If you do not include .cpp (lower case), Atmel Studio will automatically include the ‘.cpp’ extension to the name.
   f. Click ‘Add’ and make sure the file LCD16x2_ST7032.cpp has been added under the ‘Libraries_user’ folder.
8. Enter the seven empty file pairs as per above and verify these seven library file pairs appear under the folder ‘Libraries_user’ in Solution Explorer.

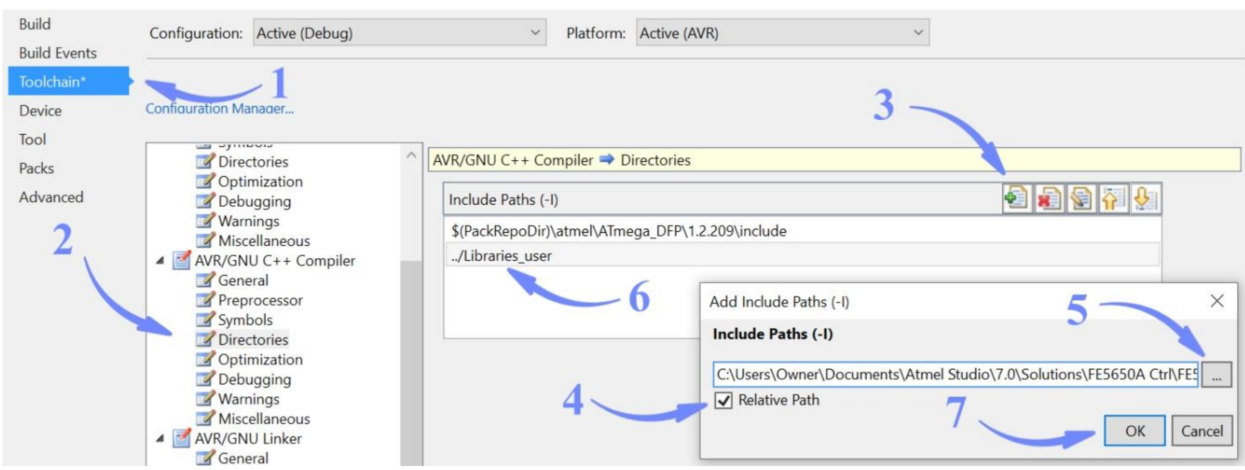


Figure 10.16: Informing Atmel Studio (and the compiler) as to the location of the user library files. The text describes the numerical sequence.

Configure A.S.: Next the Atmel Studio (AS) needs to be informed as to the location of the library files. The numbering in the following list corresponds to that in Figure 10.16.

0. From the Atmel Studio tool strip at the top, click the following sequence:
   Project > FE5650A Ctrl Properties …
1. Select ‘Toolchain’ from the left hand menu
2. Select ‘Directories’ under the heading of ‘AVR/GNU C++ Compiler’
3. Notice the dialog named ‘Include Paths’ and click the left most icon marked with a ‘+’ which means to add a new path.

4. The ‘Add Include Paths’ dialog will appear. Add a check mark to the check-box next to ‘Relative Path’ if it’s not already checked. The ‘relative path’ finds ‘Libraries_user’ inside the AS Solution folder.
5. Click the ellipsis next to the textbox. At this stage the textbox will be empty. Navigate to the directory/folder titled ‘Libraries_user’ in your ‘FE5650A Ctrl’ solution folder and select it. The path to the folder will appear to the left of the ellipsis as shown in Figure 10.16.
6. Once the OK button is clicked, the path will appear in the textbox of the ‘Include Paths’ dialog.
7. Exit out of all the dialogs by clicking ‘OK’ (etc.).

Populate the user libraries: Now populate the library .cpp and .h files with code. Start for example, by double clicking the header file ‘ADC328.h’ found under ‘Library_users’ in Solution Explorer. The page will open and a tab will appear near the top of the Atmel Studio. Appendix 6 has the source code. Copy the source code line-by-line into ‘ADC328.h’ and especially be careful to include the various braces { } and semicolons ‘;’. You don’t need to copy the comments indicated by the double slash // or the text between /* and */. Once the header .h file has been populated, double click the implementation file ADC328.cpp, and copy the code lines into the file. It’s probably a good idea to occasionally save the entire solution by either using the menu sequence ‘File > Save All’ or by clicking the icon in the toolbar showing the double diskette.

Populate main.cpp: Next the main.cpp page should be populated with code. Double click ‘main.cpp’ displayed in Solution Explorer on the right hand side of Atmel Studio. The ‘main.cpp’ tab will appear near the top of the Atmel Studio. Delete all of the lines in the page since they won’t be used here. Copy the source code line-by-line into ‘main.cpp’ and especially be careful to include the various braces { } and semicolons ‘;’. You don’t need to copy the comments indicated by the double slash // or the text between /* and */. As usual, it’s probably a good idea to occasionally save the entire solution by either using the menu sequence ‘File > Save All’ or by clicking the icon in the toolbar showing the double diskette.

Check for Errors: At the top menu strip in the Atmel Studio, click the sequence ‘Build > Build Solution’. In the event that all code has been properly entered, the result window at the bottom will show ‘build succeeded’ and you are ready to program the chip. Otherwise, the result window will show some errors. If the only error is ‘segmentation error’, try the build a few times till it clears up. Otherwise, note the line of the error and compare it with that in the applicable appendix code. All of the code in the appendices has been tested as follows: The source code was copied into a Microsoft Word document and that appears in the appendix. The code was then copied (using select and copy) from each doc file and pasted into main.cpp or the library files in an Atmel Studio 7 Solution. The files compiled without error. This means the appendix code does not have typo errors or missing braces or semicolons.

Load and Run the code: The program can be loaded into the ATMEGA328P. Place the ATMEGA328P into the circuit for the FE5650A controller or one with at least the 5V regulator and programming connections and preferably with the 16MHz crystal and associated 22pF capacitors. Connect the programmer to the circuit and the PC. Power the 328P circuit. Make sure Atmel Studio is running and all has been setup as in the previous sections and topics for programmer and fuses, and that the FE5650A program has been loaded into the Atmel Studio. The code from the ‘FE5650A Ctrl’ can be loaded into the 328P uC by clicking the triangle in the Atmel Studio tool bar (Figure 10.17)

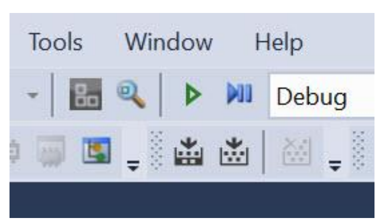


Figure 10.17: The forward pointing triangle (without the bars) loads and runs the program.

Topic 10.5.6: Option 3: Program the ATMEGA328P from an ATMEL Solution

The ATMEL Solution contains the source code for the ATMEGA328P controller for the FE5650A. With access to the source code, it is possible to modify the program to fit your needs. For example, the first version (actually an Alpha version) cannot cause the FE5650A itself to save a new Fcode or reference frequency – the source code can be quite easily modified to do so. Despite the convenience, the Solution in its present form does not program the fuses. Complete the previous topics as appropriate and transfer the code as described in the previous topic in conjunction with Figure 10.17.

Section 10.6: Description of the Controller Program: main

The controller consists of keypad, text LCD, temperature sensor, and the ATMEGA328P microcontroller (uC) for its computing capability, USART (TTL RS232), timer counter, and ADC. The 328P is programmed in C++ but primarily uses functions and methods found in C (without classes and other fancy constructs) [10.9-12]. The present section briefly discusses some of the routines in main.cpp. These routines rely on the libraries placed in the 'Libraries_user' directory. Some of the discussion will require access to the source code provided in Appendix 6 along with the data sheet for the ATmega328P [10.13] and perhaps some review of basic C concepts [10.9-12].

The present section can be skipped if the reader does not intend to modify the microcontroller source code.

Topic 10.6.1: Startup

The compiler/linker incorporates various statements prior to the main routine '`int main(void)`' including the directives #define and #include, and the declaration of variables and functions. Sometimes those various lines prior to the main routine are said to exist at the modular or Global Level since variables defined there have global scope in the sense that all routines on the same page can access and modify their values. The entry point of the program is the 'main' routine. For the present purpose, the first several lines after '`int main(void)`' initialize/configure the microcontroller (uC) ports, 16 bit timer-counter TC16, USART, and ADC, and the uC initializes the LCD.

Starting near the top of main.cpp, notice the #define directives for USART_TIMEOUT and TEMP_TIME of 500mS and 4000mS, respectively, for the USART timeout and the ADC inter-sample time. The KP refers to the proportionality between Rstored and Rtrue as in Rtrue=KP*Rstored. The uC can use non-volatile memory (i.e., eeprom) to save changes to the parameters entered by the user including Rstored, Fstored, KP, Rtrue, Fcode, Fout, and Baud. The program uses two string arrays as temporary string storage. The larger one holds 128 bytes

```
static char TemporaryMemory[128] = {0};
```

and can be found in 'myF64.cpp' for 64bit floating type. The smaller array is

```
char result[RESULTLEN] = {0}
```

where RESULTLEN represents the number 22 which is of sufficient size to store the characters on one line of the text-based LCD. The various arrays are sized by assigning typical numbers such as

```
char Rstored[] = "50255054.934100"
```

The very first time the uC runs, the strings will be stored in the eeprom at the address previously specified. During normal operation, the menus provide an option for saving changed parameters to eeprom. The next time the uC runs, the arrays will be updated with the new eeprom values. The baud rate will also be saved to eeprom.

As previously mentioned, part of 'startup' includes initializing and configuring the various uC modules including uC_Init, keyPad_init, TC16_config, LCD_init, and ADC_config which can be found as the first lines in `int main(void)`. All but uC_Init( ) will be discussed in the next section regarding the user libraries. The routine uC_Init( ) sets the 'Caps Lock' and 'Auto Send' pins as outputs and sets the logic states to zero. More interestingly, it also determines whether the program has been previously run on the current uC by searching eeprom for the letters ALL at the predefined address of 'ADDR_ALL' which, in this case, represents zero. If 'ALL' is not present, then the uC stores the various arrays in eeprom at their predefined eeprom addresses; these addresses were defined by the directive #define and the tokens start with ADDR_. If the program has previously run, the uC retrieves the content for the various arrays from the eeprom at the predefined addresses, and initializes the arrays prior to running the program. That is, the program overwrites the declarations found in the global area for main.cpp.

Topic 10.6.2: Main Menu

The main menu for operation of the program can be found in `int main(void)` and it functions mainly as a routing system for the functions of the program.

```
char keyMM = 0;
    while(1)
    {   L_clear();                                                   // Clear LCD
        L_cursorLinePos(0,0); L_writeStr("1:Bau 2:S&K 3:R");         // at line 0, pos 0 write Bau, S&K, R
        L_cursorLinePos(1,0); L_writeStr("4:Opt 5:Tmp 6:Go");        // Opt=nothing, Tmp: degree C, Go: freq settings

        keyMM = keyGet();
        switch(keyMM)
        {   case '1': menuBaud(); break;
            case '2': menuStatus(); break;
            case '3': menuRtrue(); while(keyPad()==0); break;
            case '4': break;
            case '5': menuTemp(); break;                             // shows module temperature
            case '6': menuGo(); break;                               // runs frequency control part of program
            default: break;  }
    }
```

The menus operate in similar fashion to the main one except the while(1) statement prevents the main menu from terminating. The main menu clears the LCD and then writes the abbreviations

Number	Prompt	Expanded	Description
1	Bau	Baud	Set the baud rate
2	S&K	Status and Kp	Status enquiry and set Kp
3	R	Reference Frequency	Reference Frequency menu
4	Opt	Options	Does nothing
5	Tmp	Temperature	Continuously monitors temperature
6	Go	Frequency	Adjust output frequency

The call to the subroutine L_cursorLinePos(y,x) positions the cursor at line y=0 (top) or line y=1(bottom) and position x=0 to 15 along the horizontal direction. The next write to the LCD will occur at the location of the cursor. The user presses the number on the keypad corresponding to the secondary menu. The keyGet() routine waits until a key is pressed and returns the character to the switch block. Once the secondary menu returns to the calling switch case, the while(1) block rewrites the main menu and the process starts again.

Topic 10.6.3: Baud Menu: `void menuBaud(void)`

The baud menu enters, saves or retrieves a baud rate for the uC USART which provides communications between the FE5650A and the ATmega328P. The baud menu routine continues to run until the menu key (formerly the D on the keypad) or the escape 'E' key is pressed. The baud menu clears the LCD using L_clear() and then writes the currently used baud rate to line 0 on the LCD. It then writes menu options of 1: set new baud; 2: Save new baud to eeprom; 3: recall a value from eeprom.

```
void menuBaud(void)
{     char key = 0;
      while(key!='E')                                     // E = escape code
      {     L_clear();                                     // clear LCD
            L_writeStr("Baud: ");
            L_writeStr(uint32toa(baud,result,10));         // show current baud
            L_cursorLinePos(1,0);                          // set cursor to line 1, position 0
            L_writeStr("1:New 2:Sv 3:Rcl");                // New Baud, Save, Recall

            key=keyGet();                                  //wait for key press
            switch (key)
            { case '1':
                  L_erase(1,0,15);                         // erase LCD Line 1, positions 0-15
                  L_cursorLinePos(1,0);                    // cursor position 0, line 1
                  L_cursorVisBlink(true,true);             // show cursor and make it blink
                  getNumberDsply(1,0,result);              // get a typed number starting at line=1 position=0
                  L_cursorVisBlink(false,false);                       // invisible cursor, no blink
                  if ( result[0] != 0 ) baud = (uint16_t)atol(result);  // set variable 'baud'
                  break;

             case '2': //Save
                  eeprom_update_word( (uint16_t*)ADDR_BAUD, baud );    // save to eeprom
                  break;

             case '3':
                  baud = eeprom_read_word( (uint16_t*)ADDR_BAUD );     // recall from eeprom
                  break;

             default: key = 'E'; break; }                  // end switch

             USART_config(F_CPU,baud);
      }
}
```

The instruction key=keyGet() waits until the keypad returns a number as to the choice of menu item.

The case of key='1' causes the uC to erase the top line (i.e., line 0), make the cursor visible and blinking, and get a baud rate from the keypad using getNumberDisply(1,0,result). The 1,0 in the argument means the number will be on line 1, position 0. The array result[] in the argument list holds the string produced by getNumDsply() but the routine also returns a pointer to result[ ]. After getting a result, the cursor quits blinking and becomes invisible. The result is converted to an integer and saved to the global variable 'baud' and then the USART is updated.

The case of key='2' causes the uC to save the value of the variable 'baud' to eeprom memory. So entering a new baud by using the first option in the menu and then using this second option will cause the new entered value to be saved to eeprom.

The case of key = '3' causes the uC to recall the eeprom value and save it in the globals variable 'baud'.

Topic 10.6.4: Status and Kp Menu: `void menuStatus(void)`

The status menu checks for properly operating serial ports with the correct baud, returns the stored values of the reference frequency Rstored, which the FE5650A delineates by 'R=', and the Fcode, which the FE5650A delineates by 'F='. Additionally the menu allows the constant Kp for the relation 'Rtrue = Kp * Rstored' to be entered, saved to eeprom and recalled from eeprom. Generally, most of the menus will call keyGet( ) which will cause the uC to stop executing the program until the keypad returns an ASCII code. Unfortunately, the listing for the menu routine is quite large but it is included here in full for convenience. Refer to Appendix 6.

```
void menuStatus(void)
{
    char c = 0;                                     //used to save to Fstored
    char key = 0;
    while(key != 'E')
    {   L_clear();                                  // clear LCD
        L_writeStr( "1:S? 2:S\7 3:S\6" );           // Status, status save, status rcl
        L_cursorLinePos(1,0);                       // cursor at line 1 and pos 0
        L_writeStr( "4:K  5:K\7 6:K\6" );           // Kp, kp save, kp rcl

        key = keyGet();                             // wait for key press
        L_clear();                                  // clear LCD

        switch (key)
        { case '1':                                                  // get Rstored and Fstored from FE5650A

            USART_ClearBuffer();                                     // clear buffer for rvd chars
            USART_SendChar('S'); USART_SendChar(0x0D);               // sends S<cr>

            TC16_Start();                                            // start timeout timer
            while( !TC16_isTimeExpired() ) ;                         // wait for time expired

            if( USART_isBuffEmpty() ) {L_writeStr("No Comm.");  keyGet(); break; }         //FE5650A not connected
            if ( !USART_bufchr('R') ) { L_writeStr("Xmt Err/Encrypt"); keyGet(); break;}   //wrong baud
            if ( !USART_bufchr('F') ) { L_writeStr("Xmt Err/Encrypt"); keyGet(); break;}   //wrong baud

            while ( USART_ReadBuffChar() !='R' && !USART_isBuffEmpty() );   // repeatedly reads char from buffer
            L_writeChar('R');                                        // print R to LCD
            USART_ReadBuffChar();                                    // eliminate '='
            //place Rstored in global variable and print on LCD
            for(int i = 0; (i<15) && !USART_isBuffEmpty(); i++)      // 15 chars or until buffer empty
                { Rstored[i] = USART_ReadBuffChar();L_writeChar(Rstored[i]); } // 0 already in last element
```

```
        while ( !USART_isBuffEmpty() && USART_ReadBuffChar() != '=' ) ;    // move to HEX str after 'F='
        L_cursorLinePos(1,0);                                               // move cursor to line 1, pos 0

        //extract 8 digits for the stored Fcode
        for(int i = 0; (i<16) && !USART_isBuffEmpty(); i++)                 // read 16 hex chars from FE
              { L_writeChar( c = USART_ReadBuffChar() );                    // set c to F digit and write on LCD
               if(i<8) Fstored[i]=c; }                                      // Fstored only uses first 8 digits

        while(keyGet()==0);                                                 // wait for key
        break;

    case '2':                                                   // Save Rstored and Fstored to eeprom
        L_writeStr(Rstored);
        L_cursorLinePos(1,0);                                   // cursor moves to line 1 position 0
        L_writeStr(Fstored); L_writeStr(" 0:N 1:Y");
        if (keyGet() == '1')                                         // save Rstore and Fcodestored
              { eeprom_update_block( (const void*)Rstored, (void*)ADDR_RSTORE, 16);
               eeprom_update_block( (const void*)Fstored, (void*)ADDR_FSTORE, 9);  }
        break;

    case '3':                                                        // Rstored and Fstored from eeprom
        eeprom_read_block( (void*)Rstored, (const void*)ADDR_RSTORE, 16 );
        eeprom_read_block( (void*)Fstored, (const void*)ADDR_FSTORE, 9 );

        L_writeStr(Rstored);                                         // write saved Rstored
        L_cursorLinePos(1,0);                                        // cursor line 1, position 0
        L_writeStr(Fstored); L_writeStr(" Key OK");                  // show save Fstored
        break;

    case '4':                                                   // Show old K on line 0 and enter K on line 1
        L_writeStr(Kp);
        L_cursorLinePos(1,0);
        L_cursorVisBlink(true,true);                            // make cursor visible and blinking

        getNumberDsply(1,0,result);                             // get number starting at line 1 pos 0

        if ( (strlen(result) != 0) && (result[0]!='0') )        // if not blank, put result into global variable
              { strcpy( Kp, result ); }
        L_cursorVisBlink(false,false);                          //invisible cursor, no blink
        break;

    case '5':                                                   // Save to eeprom if ok
        L_writeStr(Kp);
        L_cursorLinePos(1,0);
        L_writeStr("Save: 1:Yes 2:No");
        if(keyGet()=='1') { eeprom_update_block( (const void*)Kp, (void*)ADDR_KP, 14); }
        break;

   case '6':                                                    // RCL from eeprom
        eeprom_read_block( (void*)Kp, (const void*)ADDR_KP, 14 );
        L_writeStr(Kp);
        keyGet();                                               // wait for key press
        break;

    default: key='E'; break;  } //end switch
  }     // end while
}
```

The sensors/Kp menu loops until receiving the menu key (formerly D on the keypad) or escape key ‘E’. The routine writes the menu to the LCD according to the following table

Number	Prompt	Expanded	Description
1	S?	Status?	Send Status Request and disply Rst Fcd
2	S=>	Save Rst and Fc	Save Rstored and Fcode to uC eeprom
3	S<=	Recall Rst and Fc	Recall Rstored and Fcode from uC eeprom
4	K	Show and enter Kp	Shows current or enters new Kp
5	K=>	Save Kp	Save Kp to uC eeprom
6	K<=	Recall Kp	Recall Kp from uC eeprom

For example, the line `L_writeStr( "1:S? 2:S\7 3:S\6" )` has \7 and \6 which correspond to the custom characters right arrow and left arrow, respectively. As well known, C/C++ can include control characters (those normally non-printable characters) in a string by preceding the ASCII code by the ‘\’. The ASCII code 6 and 7 were defined for the LCD as custom characters in the uC_Init routine.

The instruction `’key = keyGet()’` waits until the keypad returns a key and then the switch block decides on the proper response to the key.

The longest list of instructions occurs for case ‘1’ for the status enquiry since it tests for proper serial port operation. The routine first sends the status enquiry ‘S <Cr>’ to the FE5650A, then starts the timer counter TC16 using TC16_Start( ), and then waits for the approximately 500mS time to expire using `while( !TC16_isTimeExpired() )` since that gives the FE5650A sufficient time to send the response. The next couple of lines check the circular buffer for any response and if none, returns the response that the serial port is not properly connected. It then checks for the readable characters ‘R’ and ‘F’ since if the FE5650 sent ‘something’ but if it does not read as ‘R’ and ‘F’, then most likely the uC and the FE5650A have mismatched baud rates. Assuming proper communications, the line

```
Rstored[i] = USART_ReadBuffChar();L_writeChar(Rstored[i]);
```

places the returned value of the stored reference frequency into the global array Rstored[ ] and then writes it to the LCD. Similarly it writes 16 characters returned for Fcode to the LCD but keep in mind, only the first 8 set Fcode for the FE5650A.

Case ‘2’ for Save simply stores both Rstored[ ] and Fstored[ ] in the uC eeprom and shows it on the LCD.

Case ‘3’ for Recall replaces the gobal Rstored[ ] and Fstored[ ] with that from the eeprom and shows it on the LCD.

Case ‘4’ shows the current value of the proportionality constant Kp on the LCD and then waits for ‘getNumberDsply’ to return a number or key and copies nonzero numbers to Kp from the global array result[ ].

Case ‘5’ writes the current value of Kp to the LCD and then asks if it should be save to eeprom and if so, saves it.

Case ‘6’ recalls Kp from eeprom and writes it to LCD.

Case: default: returns the escape key of ‘E’ whenever a key other than 1, 2, 3, 4, 5, or 6 is pressed.

Topic 10.6.5: Reference Frequency Menu: `void menuRtrue(void)`

The menuRtrue( ) routine allows the user to enter a new value of Rtrue, and use the value calculated from ‘Kp * Rstored’, and save/recall the value of Rtrue to/from eeprom. Unfortunately, the Atmel Studio 7 has one drawback regarding calculations – it does not distinguish between the 32 bit floating numbers (float) and the doubles. As a result, it was necessary to write or find either string or math methods of calculating numbers with up to 15 or 16 significant digits (base 10). People working with GPS coordinates have the same problem. Fortunately, as described in the ‘myF64’ library, one valiant developer wrote and released the code for 64 bit floating point [10.17]. For those not having investigated the floating point format for number stored in memory, it’s worth the read as shown in References [10.17].

In operation, the menu will continue to loop using the instruction `while(key != 'E')` until the keyboard returns the escape character ‘E’ or a character other than 1, 2, 3, 4, 5 ,6. Not all six menu options have functions and can be customized by the developer.

```
void menuRtrue(void)                                    // Enter, save and recall Rtrue
{     char key = 0;
      while(key != 'E')                                 // continue until key = E = escape
      {     L_clear();                                  // clear LCD

            L_writeStr( "1:R? 2:KRs 3:..." );           // show/enter Rtr, Rtr from Rst, nothing
            L_cursorLinePos(1,0);                       // cursor to line 1 pos 0
            L_writeStr( "4:R\7 5:R\6  6:..." );         // R save, R rcl, nothing

            key = keyGet();                             // wait for key to be pressed
            switch (key)
            { case '1':                                 // show Rtrue or get new Rtrue
                  L_clear();                            // clear LCD
                  L_writeStr(Rtrue);                    // write existing Rture to LCD
                  L_cursorLinePos(1,0);                 // set cursor to line 1, position 0
                  L_cursorVisBlink(true,true);          // make cursor visible and blinking

                  getNumberDsply(1,0,result);           // input a number printing at line 1, pos 0
                  if ( (strlen(result) != 0) && (result[0]!=0) ) strcpy(Rtrue,result);   // put in global var Rtrue if ok

                  L_cursorVisBlink(false,false);                  // invisible cursor, no blink
                  break;

             case '2':                                            // use R = Kp * Rstored
                  {float64_t fnum = f_strtoF64(Kp);     // need braces {} when defining new variables here
                  float64_t RstoredTemp = f_strtoF64(result FreqStrConditioner(Rstored );
                  float64_t RtrueTemp = f_mult(fnum,RstoredTemp);    // multiply Kp*Rstore, store in temp var
            char* pRes = f_to_string(RtrueTemp,15,0);   // convert temp Rtrue from float64 to char array
                  strcpy(result,pRes);                  // copy to temporary string result[]
                  L_clear();                            // clear LCD
                  L_writeStr( result);                  // write result[] of K*Rst to LCD
                  L_cursorLinePos(1,0);                 // place cursor at line 1, position 0
                  L_writeStr( "1:Yes 2:No" );           // Save to global Rtrue if ok to use as Rtrue
                  if ( (keyGet() == '1') && (  f_compare(RtrueTemp, 0) > 0  ) ) { strcpy(Rtrue,result); } }
                  break;
             case '3': break;                                   // nothing, available for development
             case '4': eeprom_update_block( (const void*)Rtrue, (void*)ADDR_RTRUE, 16); break;    // save eeprom
             case '5': eeprom_read_block( (void*)Rtrue, (const void*)ADDR_RTRUE, 16 );  break;      //recall eeprom
             case '6': break;                                   // nothing, available for development
```

```
            default: key='E'; break;                                    // escape, done
        }
    } // end while
    return; // result;
}
```

The menuRtrue clears the LCD and writes the menu options according to the next table.

Number	Prompt	Expanded	Description
1	R?	Rtrue ?	Show current Rtrue and enter a new one
2	KRs	Kp Rstored	Use value calculated from Rtrue=Kp*Rstored
3	…	unused	Does nothing
4	R=>	Save Rtrue	Save current Rtrue to uC eeprom
5	R<=	Recall Rtrue	Recall Rtrue from uC eeprom
6	…	unused	Does nothing

The routine then uses key=keyGet( ) to wait for a key press and uses the switch block to sort through the possibilities.

Case 1 for 'R?' clears the LCD and writes Rtrue to the top line, sets the cursor to visible and blink, gets a number from the keypad if desired and saves it to the global variable Rtrue[ ], and then stops the cursor blink and makes the cursor invisible.

Case 2 for 'KRs' calculates Rtrue = Kp*Rstored using the 64bit floating point numbers in the User Library myF64. In order to prevent the number of bytes (8 per number) from growing too rapidly, the F64 variables are reused which tends to make the code a bit more confusing to read. First the routine converts the string Kp to temporary float64 using `float64_t fnum = f_strtoF64(Kp).` It next converts the string Rstored[ ] to a float64

```
float64_t RstoredTemp = f_strtoF64(Rstored)
```

The routine multiples these two numbers to form Kp*Rstored and again stores it in the temporary variable `RtrueTemp = f_mult(fnum,RstoredTemp).` The following two lines convert the RtrueTemp number to a string with pRes as the pointer to the string and then copies the string to the global array result[ ].

```
char* pRes = f_to_string(RtrueTemp,15,0);     // convert temp Rtrue from float64 to char array (pointer)
strcpy(result,pRes);
```

In actuality, the function f_to-string uses temporary storage and further manipulation could have been done there. Next the routine clears the LCD and writes result[ ] to the LCD and offers the chance to either discard or use the calculated value of Rtrue. The value will not be save to eeprom at this point.

Case 4 and 5 saves/recall to/from eeprom memory.

Topic 10.6.6: Temperature Menu: `void menuTemp(void)`

The temperature menu doesn't have any associated options but instead continuously displays the temperature returned by the LM35DZ module to the uC ADC on channel 0. The routine reconfigures timer counter TC16 for intervals of 4seconds to trigger the uC ADC. A similar routine is used to display the temperature in the goMenu.

```
void menuTemp(void)                                        //shows temperature
{    //configure ADC, modify timer for 4 secs; manual trig ADC; ADC function writes to LCD; get key to exit;
     //reset timer for main routine prior to exit

     TC16_config(TEMP_TIME, F_CPU, false, false, false, 0);      //4000mSec timing period, fcpu=16MHz
     ADC_Enable();
     ADC_StartConvert();                                   // manual convert for first run
     _delay_ms(5);                                         // need delay as the manual ADC start appears to interfere

     L_clear();                                            // clear LCD
     L_writeStr("Temp \4C ...");                           // write temperature info 1
     L_cursorLinePos(1,0);
     L_writeStr("Time: ");                                 // write temperature info 2
     L_writeStr( "TEMP_TIME " );
     L_writeStr("mSec");

     TC16_Start();                                         // ADC auto triggers from TC16 every 4seconds

     while ( keyGet() == 0 ) ;

     TC16_Stop();                                          // stop timer which stops ADC trigger
     ADC_Disable();                                        // disable ADC
     TC16_config(USART_TIMEOUT, F_CPU, false, true, true, 0); //back to 500mSec for USART, fcpu=16MHz
}
```

The menuTemp routine first configures TC16 for the 4000 mSec time interval

`TC16_config(TEMP_TIME, F_CPU, false, false, false, 0);`

but does not allow it to call a function, set a boolean, or to stop the timer in the ISR. Once the TC16 starts, it triggers the ADC interrupt every 4000 mSec. Next the ADC is enabled using `ADC_Enable()`. The ADC Interrupt Service Routine (ISR) will call a function in main.cpp that receives the output voltage from the sensor and converts it to a temperature for the LCD. The first ADC conversion is initiated by ADC_StartConvert() even though the ADC can trigger itself in normal operation. *One important note: the manual ADC trigger, namely ADC_StartConvert(), appears to interfere with the LCD operation and so a delay of 5mSec has been added and seems to work ok. Edit: There still appears to be some interference and the delay might best be set to 10mSec.* The required delay might depend on the ADC clock rate – we have not checked. The routine clears the LCD and writes “Temp ^{o}C” using L_writeStr("Temp \4C ...") where the \4 inserts the character for the ASCII code of 4 installed in the LCD during setup. The \4 produces the ‘degrees’ symbol next to the centigrade symbol C. The next few instructions show the interval time on line 1 of the LCD. After all of the LCD writes, the routine starts the TC16 routine which triggers the ADC every 4 seconds until the keypad returns a pressed key using: while (keyGet() == 0). Once the keypad returns a character, the TC16 stops and the ADC disables. The routine reconfigures the TC16 for approximately 500mSec time interval for the USART.

Topic 10.6.7: Go Menu: `void menuGo(void)`

The primary purpose of menuGo consists of changing the output frequency of the FE5650A. The LCD top line (#0) shows the Fcode corresponding to the output frequency Fout in the LCD bottom line (#1). For the LCD and for transmission to the FE5650A, the Fcode[ ] and Fout[ ] are placed in standard form in that leading zeros are added if needed to make Fcode[ ] have 8 hex digits and Fout[ ] have 8 integer digits, a decimal point, and 3 fractional digits.

The good news: the menuGo is the last menu to be considered; the bad news: its long. So rather than a single long listing prior to the discussion, the menuGo routine will be pulled apart and embedded into the relevant discussion. Refer to Appendix 6 for a single listing.

The menuGo routine starts by providing some definitions. The integer variables iLine and iPos have been moved to the main globals area. The variable iLine refers to the LCD line number (top:0 and bottom:1) and iPos refers to positions (0-15) within the line. The leading character 'i' refers to the integer INDEX (starts at 0). The iLine and iPos must retain values between operations. The iFo and iFc are the indices (start at 0) into the arrays for the output frequency Fout[ ] and Fcode[ ] once placed in standard form. Without the 'AutoSend' engaged, the changes made to the frequency do not take place immediately but instead populate the Fo_try[ ] and Fc_try[ ] arrays and updated the global variables Fo[] and Fc[] when the new frequency is sent to the FE5650A. By the way, iLine and iPos necessarily refer to the LCD line (0:top, 1:bottom) and cursor position and also to Fc_try (for iLine=0) and Fo_try (for iLine=1). Likewise iPos refers to given digits of the display lines and the arrays although a conversion routine is required.

```
void menuGo(void)
{
    //uint8_t iLine = 1;          //made global: always updated; cursor line index at start
    //uint8_t iPos = 5;           //made global: always updated; cursor position index init.
    uint8_t iFo = 1;              //index in Fout
    uint8_t iFc = 0;              //index in Fcode
    char Fo_try[18] = {0};        //proposed Fout - will be in standard form
    char Fc_try[10] = {0};        //proposed Fcode - will be in standard form
    char FTemp[18] = {0};         //for calculation
    char key = 0;

    bool flagAuto = false;        //auto send means each LCD update is sent to FE5650A
    bool flag_exitIF = false;     // used to exit an IF
    bool flag_Fchanged = false;   //the frequency has changed in GoMenu when true

    strcpy(Fc_try, Fcode);        // note must always have 8 digits
    strcpy( Fo_try, FreqStrConditioner(Fout,true,result) ); //std: 8 int & 3 fract digits
```

The boolean flagAuto refers to the option of sending an update to the FE5650A after each key press; this option makes it easy to slowly change the actual output frequency for calibration purposes.

Next the routine configures the timer for 4 second intervals to trigger the ADC and it enables the ADC and manually initiates the first conversion. The ADC ISR will place the temperature at the end of the top LCD line. Only after the manual ADC conversion using 'ADC_StartConvert()' does the routine start the timer counter TC16.

```
    //========= Start ADC to show temperature
    TC16_config(TEMP_TIME, F_CPU, false, false, false, 0); //4 Sec time period, fcpu=16MHz
    ADC_Enable();
    ADC_StartConvert();                         //must start even for free run
    _delay_ms(10);                              //the adc conversion interferes with writeStr!?!? Might need 15
    TC16_Start();                               //runs the adc
    //=====
```

The routine clears the LCD and writes the Fcode_try on the top line and Fout_try on the bottom line of the LCD. The cursor is visible but not blinking. A 'while()' block continuously retrieves a keypad 'key' but exits if the returned character is 's'. The boolean 'flag_Fchanged' signals to the autoSend routine when Fcode has changed.

```
L_clear();                                                // clear LCD
L_writeStr("Fc: "); L_writeStr(Fc_try);                   // write proposed Fcode
L_cursorLinePos(1,0); L_writeStr("Fo: "); L_writeStr(Fo_try);   // write proposed Fout

L_cursorLinePos(iLine,iPos);                              // set cursor to previous position
L_cursorVisBlink(true, false);                            // show cursor but no blink

while ( ( key = keyGet() ) != 's' )                       // exit only if receive an 's'
{     flag_Fchanged = false;                              //Auto Send needs to know if freq changed
```

A 'switch()' block with lots of cases processes the pressed key. For case 't' (toggle), the iLine changes between 0 and 1 using the 'exclusive or'. When the iLine changes, the cursor must be properly positioned (i.e., iPos) based on the previous value iLine. For example, when transitioning the cursor from the top LCD line (iLine=0) to the bottom line (iLine=1), the new iPos must be determined to skip over the decimal point on the bottom LCD line and in Fo_try[]. Similarly when changing between LCD lines as well as Fo_try[] and Fc_try[], the best practice would position the cursor in such a way that changes to a digit in one line correspond to changes in the other.

```
switch( key )
{     case 't':                 //toggle current LCD row
      if (  (iLine ^= 0b01) == 0 )     //Line number went from 1 to 0, so set new array index and cursor position
      { iFc = FcodeIndexFromFoutIndex(iFo); L_cursorLinePos(0, iPos=cursorPosFromFcFoIndex(iFc) ) ;   }
      else                      //Line number went from 0 to 1
      { iFo = FoutIndexFromFcodeIndex(iFc); L_cursorLinePos(1, iPos=cursorPosFromFcFoIndex(iFo) ) ;   }
      break;
```

If the keypad returns the auto-send key 'a', the boolean flagAuto is inverted as is the corresponding LED state.

```
case 'a':                                                 //toggle auto send
flagAuto = !flagAuto;
if (flagAuto) { PORT_AUTO  |= (1 << PIN_AUTO); }          //light LED for auto send
else { PORT_AUTO &= ~(1 << PIN_AUTO); }                   // extinguish LED for auto send
break;
```

The UP Arrow 'U' changes the digit at the cursor position to larger values until it assumes the maximum value of 'F' for the Hex Fcode and '9' for the decimal Fout. The UP arrow does not change position within a line nor does it change the currently selected line nor does it cause a carry to the next digit. For the Up Arrow 'U', the routine must distinguish between the starting line of iLine=1 (bottom) or iLine=0 (top) since it is really the Fcode[ ] that must change and be sent to the FE5650A. Changing the Fout[ ] must eventually be translated to change the Fcode for both the LCD and for the FE5650A. Of course, a change of Fcode must be translated for display of Fout on the LCD. For the case of iLine=1 (bottom line), an 'if' statement checks that the current digit at index iFo is less than '9' and then if so, the routine increases the ASCII value by 1, erases the character using L_erase(1,iPos,iPos), writes the new character and returns the cursor, puts Fout in standard calculation form without leading zeros using FreqStrConditioner(Fo_try,false,result) calculates the new Fcode using FcodeCalc(FTemp, Rtrue) , puts Fcode in standard form with leading zeros, and writes the string Fc_try[ ] to line 0 (top line), and then sets the boolean flag_Fchanged=true in case the autosend is active.

```
case 'U':                                                //increase digit, calc new Fout, put Fout in Std Form
    if(iLine == 1)                                       // a decimal digit is increasing
    { if (  (Fo_try[iFo] >= '0') && (Fo_try[iFo] < '9') )   // don't exceed '9'
     { Fo_try[iFo] += 1;                                 // increase digit at cursor
        L_erase(1,iPos,iPos); L_cursorLinePos(1,iPos);   // prepare spot for new char
        L_writeChar( Fo_try[iFo]); L_cursorLinePos(1,iPos);        // update the display
        strcpy(FTemp, FreqStrConditioner(Fo_try,false,result) );   // remove lead zeros for calcs
        strcpy( FTemp, FcodeCalc( FTemp, Rtrue) );                 // calc Fcode and store in FTemp
        strcpy( Fc_try, FcodeStrConditioner(FTemp,true,result) );  // make Fc have standard form
        L_cursorLinePos(0,4); L_writeStr(Fc_try);                  // write new Fc for new Fout
        L_cursorLinePos(1,iPos);                                   // put cursor back at changed char
         flag_Fchanged = true;  }                                  // values have changed
     } //end of if(iLine == 1)
```

However, for the case of iLine=0 (top line changing at cursor), the first several lines of code test whether the digit can be changed and the manner in which it can be changed. The digit '9' would be increased to 'A', whereas 'E' would change to 'F'. If the digit is already 'F' then the routine exits. Assume a change has taken place. Then similar to the previous case Fo line=1, the routine erases the currently changing digit, writes the new one, puts Fcode in standard form for calculations (removes leading zeros) using

FcodeStrConditioner(Fc_try,false,result)

calculates the new Fcode, places the new Fcode in standard form with leading zerros, writes it to the LCD and sets the boolean flag_Fchanged=true in case the auto-send is operating.

```
    else                                                           // increase HEX Fcode on line zero
    {       flag_exitIF = false;
            if ( Fc_try[iFc] >= '0' && Fc_try[iFc] < '9' ) {Fc_try[iFc] += 1; }      //[0,9) => increase char
            else if ( Fc_try[iFc] == '9' ) { Fc_try[iFc] = 'A'; }                    // if char=9 then make it A
            else if ( Fc_try[iFc] >= 'A' && Fc_try[iFc] < 'F' ) { Fc_try[iFc] += 1; } // [A,F)
            else { flag_exitIF = true; }                                             //at limits of numbering

            if ( ! flag_exitIF )                                       //nothing changed so exit
            {  L_erase(0,iPos,iPos); L_cursorLinePos(0,iPos);          //erase char to be changed
                 L_writeChar( Fc_try[iFc] ); L_cursorLinePos(1,iPos);  //update the display
                 strcpy(FTemp, FcodeStrConditioner(Fc_try,false,result) ); //remove lead zeros
                 strcpy( FTemp, freqOutCalc( Rtrue, FTemp) );          //calc Fout and store in FTemp
                 strcpy( Fo_try, FreqStrConditioner(FTemp,true,result) ); // put calc Fout in Fo_try
                 L_cursorLinePos(1,4);  L_writeStr(Fo_try);            // overwrite old Fout
                 L_cursorLinePos(0,iPos);                              // cursor at present char
                 flag_Fchanged = true;}
    } //end else
    break;
```

The DOWN Arrow key 'D' causes the digit at the position of the cursor to decrease in a manner very similar to the up arrow key. The primary differences include decrementing the digit instead of incrementing and noting no digit can decrease below '0' and for iLine=0 the Hex digit of 'A' jumps to '9'.

```
case 'D':
    if(iLine == 1)                                             // decrease a decimal digit
    {      if ( Fo_try[iFo] > '0' && Fo_try[iFo] <= '9' )      // cannot decrease below 0
           {      Fo_try[iFo] -= 1;                            // decrease digit
```

```
                L_erase(1,iPos,iPos); L_cursorLinePos(1,iPos);              // erase changed digit
                L_writeChar( Fo_try[iFo] ); L_cursorLinePos(1,iPos);        // update the char
                strcpy(FTemp, FreqStrConditioner(Fo_try,false,result) );    // remove lead zeros for calc
                strcpy( FTemp, FcodeCalc( FTemp, Rtrue) );                  // calc Fcode, store in FTemp
                strcpy( Fc_try, FcodeStrConditioner(FTemp,true,result) );   // std form: 8 hex digits
                L_cursorLinePos(0,4); L_writeStr(Fc_try);                   // overwrite old Fc
                L_cursorLinePos(1,iPos);                                    // place cursor at changing char
                flag_Fchanged = true;  }                                    // freq has changed
        }
        else                                                                // decrease HEX Fcode digit
        {  flag_exitIF = false;
            if ( Fc_try[iFc] > '0' && Fc_try[iFc] <= '9' ) {Fc_try[iFc] -= 1; }    // (0,9] => decrease digit
            else if ( Fc_try[iFc] == 'A' ) { Fc_try[iFc] = '9'; }           // if char=A then decr to 9
            else if ( Fc_try[iFc] > 'A' && Fc_try[iFc] <= 'F' ) { Fc_try[iFc] -= 1; }    // (A,F] decrease by one
            else { flag_exitIF = true; }                                    // exit if no change in Freq

            if ( ! flag_exitIF )                                            // exit if not change in Freq
            {   L_erase(0,iPos,iPos); L_cursorLinePos(0,iPos);              // erase changed char
                L_writeChar( Fc_try[iFc] ); L_cursorLinePos(1,iPos);        // update the char
                strcpy(FTemp, FcodeStrConditioner(Fc_try,false,result) );   // remove lead zeros
                strcpy( FTemp, freqOutCalc( Rtrue, FTemp) );                // calc Fout and store in FTemp
                strcpy( Fo_try, FreqStrConditioner(FTemp,true,result) );    // standard form for Fout
                L_cursorLinePos(1,4); L_writeStr(Fo_try);                   // overwrite Fout
                L_cursorLinePos(0,iPos);                                    // return cursor to char position
                flag_Fchanged = true; }                                     // frequency changed
        }
        break;
```

The cases of the Left Arrow key 'L' and the Right Arrow key 'R' cause the cursor to mover left and right, respectively, without changing lines and without changing any of the digits. The cursor on iLine=0 can move across 8 digits while on iLine=1, the cursor can move across 8 integer digits and 3 fractional digits but must skip the decimal point.

```
    case 'L':                           //move Left in row. If row 1, skip dp
        if (iLine == 0)
            { iFc=FcIndexMoveLeft(iFc); L_cursorLinePos(0, iPos=cursorPosFromFcFoIndex(iFc)); }
        else
            { iFo=FoIndexMoveLeft(iFo); L_cursorLinePos(1, iPos=cursorPosFromFcFoIndex(iFo)); }
        break;

    case 'R':                           //move Right in row. if row 1, skip dp, and not beyond end
        if (iLine == 0)
            { iFc=FcIndexMoveRight(iFc); L_cursorLinePos(0, iPos=cursorPosFromFcFoIndex(iFc)); }
        else
            { iFo=FoIndexMoveRight(iFo); L_cursorLinePos(1, iPos=cursorPosFromFcFoIndex(iFo)); }
        break;
```

The calculation `iFc=FcIndexMoveLeft(iFc)` provides the next left-side position for Fcode[ ] on iLine=0 and similarly `iFc=FcIndexMoveRight(iFc)` does the same for moving the cursor to right. Basically, the calculations prevent the cursor from moving outside the positions for Fcode. On the other hand, for iLine=1, the calculations `iFo=FoIndexMoveLeft(iFo)` and `iFc=FcIndexMoveRight(iFc)` do similar but must skip over the decimal point.

The Escape key 'E' causes all changes to be dropped and Fo_try[ ] and Fc_try[ ] to be reloaded with the original Fout[ ] and Fcode[ ], respectively. The cursor is reset to the original position and the boolean flag_Fchanged=false.

```
case 'E':                                   // E=escape ... delete changes; if A ON, then this doesn't help
    strcpy(Fo_try, FreqStrConditioner(Fout,true,result));   // Use orig Fout; make sure in proper form
    strcpy(Fc_try, FcodeStrConditioner(Fcode,true,result));   // Use orig Fout; use standard form
    L_cursorLinePos(0,4); L_writeStr(Fc_try);               // print to LCD
    L_cursorLinePos(1,4); L_writeStr(Fo_try);               // print to LCD
    L_cursorLinePos(iLine,iPos);                            // go to last LCD position
    flag_Fchanged = false;
    break;                                                  // delete changes; return to main
```

The Carriage Return character 'c' causes the temporary strings in Fc_try and Fo_try to overwrite the original values in Fcode[ ] and Fout[ ], respectively, and then the routine sendFreq(Fcode) sends the changed Fcode to the FE5650A.

```
        case 'c':                                   // Cr = Manually send Fc to FE5650A;
            sendFreq(Fc_try);                       // send to FE5650A
            flag_Fchanged = false;
            break;

        default: break;
    } //end switch
```

As part of the top while() loop, the next routine (outside of the switch block) sends the changed Fcode to the FE5650A provides there has been a change and provided the autosend option has been selected.

```
    if(flagAuto && flag_Fchanged)                   // if Auto, save to global and send to FE
    {   strcpy( Fcode, Fc_try );                    // Fc2 in std form and so will be Fcode
        strcpy(Fout,Fo_try);
        sendFreq(Fcode);                            // send to FE5650A
        flag_Fchanged = false; }
} //end while
```

Finally, outside of the top while() statement, the routine writes 'try' variables to the global ones, makes the cursor invisible, stops the timer counter TC16, disables the ADC, and reconfigures the TC16 to the USART timeout of about 500mSec.

```
    strcpy( Fcode, Fc_try );                        // Save to global vars
    strcpy(Fout,Fo_try);

    L_cursorVisBlink(false, false);                 // Set cursor off, no blink

    //======= Stop ADC and return TC16 to USART service
    TC16_Stop();                                    // Stop timer
    ADC_Disable();                                  // Stop ADC
    TC16_config(USART_TIMEOUT, F_CPU, false, true, true, 0);   // reconfigure timer for use with USART
    _delay_ms(5);
    //=======
return;
}
```

Topic 10.6.8: Supporting Routine: getNumberDsply

The 'getNumberDsply' routine has elements of the keypad and LCD in that it retrieves a character from the keypad, writes the character to the LCD and, at the end, returns the entire number as a string.

```
char* getNumberDsply(uint8_t Line, uint8_t Pos, char* result)
{     char c = 0;
      uint8_t len = 0;                          //    index into result[], also give num characters in array
      uint8_t dispPos = Pos;                    //    dispPos is display position on LCD line; next available position
      bool flag_dp = false;                     //    dp is found
      bool flag_ESC = false;                    //    Escape E has been pressed
      bool flag_CR = false;                     //  Cr c has been pressed
```

A 'do block' processes key closures until the keypad returns the Escape key 'E' or Carriage Return key 'c'. The array result[ ] holds the assembling string while the variable 'len' serves the dual role of being the index to the next character and being the length of the assembled string so far. A series of 'if else-if' stages provide the appropriate processing.

The leading character cannot be a decimal point and if the leading character is '0' then the next character must be a decimal point. In either case, the routine will insert the two characters '0.' which is 'zero dp'; consequently, the variable 'len' has increased by two and also the display position 'dispPos'.

```
      do
      {     c=0;
            while(c==0) {c=keyPad();}                 // waits for a key to be pressed; could use keyGet()

            if(  ( c=='0' || c=='.' ) && len==0)      //leading 0 must be followed by a dp
                  { result[len++] = '0'; L_cursorLinePos(Line,dispPos); L_writeChar('0');
                   result[len++] = '.'; L_writeChar('.'); dispPos+=2; }
```

If the keypad returns a backspace key 'B', then it might be possible to enter 0123.56 or maybe 00.123 and so the next few lines prevents that case but it will allow characters to be added to the string.

```
            else if( c >= '0' && c <= '9' )
                  {if ( len != 1 || result[0] != '0')       //prevent backspace from writing 00.123 or 0123.56
                        { result[len++] = c; L_cursorLinePos(Line,dispPos); L_writeChar(c); dispPos++; } }
```

The case for the backspace key 'B' decreases 'len' by one and inserts an ASCII 0 into result[ ] and erases the character on the LCD by overwriting it with a space character.

```
            else if (c == 'B' && len > 0)             //backspace key deletes
                  { result[--len] = 0; L_cursorLinePos(Line, --dispPos); L_writeChar(' '); L_cursorLinePos(Line,dispPos);}
```

For the decimal point '.', the routine first checks that a dp has not already been inserted into the string using a 'for' loop to search through the assembling string in result[ ]. If there isn't any previous dp then insert one and write it to the LCD.

```
            else if ( c == '.' )                      //only one dp allowed ... none if max=0
                  { for (int i=0; i<len; i++)
                        { if ( result[i] == '.' ) flag_dp = true; }        //check all chars for a dp
```

```
                if (flag_dp == false)
                      { result[len++] = c; L_cursorLinePos(Line,dispPos); L_writeChar(c); dispPos++; }
          flag_dp = false;      }//end else if ( c == '.' )
```

The last couple of cases intercept the Escape key ‘E’ and carriage return ‘c’. If the keypad returns the Escape key, then the routine erases the line and resets the string result[] to the null string by placing the terminator at the first position.

```
        else if ( c == 'E' ) { flag_ESC = true; }                //escape wo changing
        else if ( c == 'c') {flag_CR = true;}
        else {;}

        } while( !( flag_ESC || flag_CR ) );

        result[len]=0;                                          //add string terminator
        if(flag_ESC)                                            // delete display line and string in result[]
        { L_erase(Line, Pos, dispPos);
        for (int i = 0; i<len; i++) result[i]=0; }

    return result;
}
```

Topic 10.6.9: Supporting Routine: FreqStrConditioner

The routine FreqStrConditioner applies to the decimal form of the output frequency Fout. The routine to place the Fcode into standard form is very similar.

```
char* FreqStrConditioner(char* existStr, bool Flag_T2AddZeros_F2RemoveZeros, char* result)
{       // For the display, form = 08388608.001: 3 fract digs, 8 int digs: 8+1+3=12, DP at i=8
        char* pExist = existStr;                              // pointer to first char in existStr
        char* pDP = strchr(existStr,'.');                     // pointer to dp in existStr, returns 0 if not there
        for(int8_t i = 0; i < 16; i++) { result[i] = 0; }     // set result[] entries to \0 just to clear it out
        result[8] = '.';                                      // place dp at proper place
        int len = 0;
        int j = 0;
```

For example, one number would be 8388608.000 Hz. The array ‘Fo[ ]’ passes into the routine through the pointer parameter existStr. The routine

`char* FreqStrConditioner(char* existStr, bool Flag_T2AddZeros_F2RemoveZeros, char* result)`

ensures that existingStr has a decimal point (dp), and three digits after the dp, and either (i) 8 digits prior to the dp by the addition of leading zero characters ‘0’ if needed or (ii) only significant digits prior to the dp by removing any leading zero characters. The boolean `Flag_T2AddZeros_F2RemoveZeros`, has a name to indicate whether to add leading zero characters ‘0’ or remove them. For numerical calculations, the ‘0’ characters should be removed while for the LCD, they should be present. The array result[ ], a global array in main.cpp, will contain the results of the routine while the routine returns a pointer to the array.

Notice first the declarations of the variables and the related comments. The pointer pDP points to the dp in existStr if it exists but pDP has the value of zero if the dp cannot be found. The routine inserts a dp into the element at index #8, which is character #9.

If the original string existStr did not have a dp, then it also did not have any fractional digits; consequently, three character zeros '0' need to be appended after the dp in the string result[ ].

```
// FRACTION PART make sure three digits after dp
if (pDP == 0) { for(int i = 9; i<12; i++) { result[i] = '0'; } }        // if no dp in original str, then place three 0s
```

Otherwise, the routine retrieves up to 3 digits from the right hand side of the dp in existStr. The pointer pExist is set to the first character after the dp (assuming it's not the terminator \0). The character *pExist is copied into the array result[ ] to the first position after the dp. Both the pointer pExist and the index j are then incremented. The process continues until the terminator \0 is found or 3 digits have been copied. In the case that the routine finds fewer than 3 digits, '0's will be added.

```
else
// get 3 digits past dp, if not there, add '0'
{           j = 0;
            pExist = pDP + 1;                                   //skip dp; each char only 1 byte in memory
            while( (j<3) && (*pExist != 0) ) { result[9 + j++] = *pExist++; } //insert chars until \0 in existingStr
            for(int i = j; i<3; i++) { result[9+i] = '0'; }                   //add zeros so x.1 => x.100
}
```

Next, regardless of whether character zeros '0' should be added or removed, they are added. If the boolean `Flag_T2AddZeros_F2RemoveZeros = false`, the added leading character zeros '0' will be removing at the end. The next two lines calculate the length 'len' of the string from the first character up to the dp if present. In the case of a dp, the length 'len' is calculated as the difference of the pointer to the dp and that for the start of the string. The pointer pExist is then set to the last character of the string or to the last character prior to the dp.

```
j = 0;                                                          //can use j with char pointers
// INTEGER PART of existStr: make sure 8 digits before dp -- add zeros if not
if (pDP == 0) { len = strlen(existStr);  }                      // j should count number of zeros to add
else { len = pDP - existStr ; }                                 // need number of digits prior to dp
pExist = existStr + len - 1;                                    //set pExist pointer to last entry before dp or end
```

The existing characters, starting at the end or just prior to the dp, are copied to the array result[] first setting an index j=7 and storing the character given by *pExist into result[j]. Afterwards, both j and the pointer pExist are decremented. By the way, because the pointers reference a character, changing the pointer by 1 causes the pointer to move to the next character. The process continues using a while() block until 8 characters are copies or till the end of the string 'existStr' but less than 8 characters. Character '0's are added as leading zeros if needed. After theses steps, the string existStr has been transformed to standard form for the present purposes.

```
j = 7;
//save in result[] at left of DP
while(  ( j >= 0 ) && ( pExist >= existStr  ) ) { result[j--] = *pExist--; } //copy starting at back until pointers equal
for ( ; j >= 0 ; j--) { result[j] = '0'; }                      //add zeros to front if needed
```

Finally, if the boolean `Flag_T2AddZeros_F2RemoveZeros` has the value of false, then the leading zeros must be removed while retaining the dp and the three fractional digits. The process starts by setting the pointer to the starting character in result[ ] and for eaching leading '0' shift the string to the left by 1 position so as to cover over the '0'.

```
//now remove leading zero if needed; even though they might have been added.
```

```
if ( !Flag_T2AddZeros_F2RemoveZeros )
{
        char* pRes = result;                              // point to start of result[]
        j = 0;
        //use this as a counter
        while(  ( *pRes++ == '0' ) && ( *pRes != '.' )  ) j++; // j=num of '0' to remove;  '++' checks next char

        uint8_t lenRes = strlen(result);
        for(int i = j; i < lenRes; i++) { result[i-j] = result[i]; } //j = # shifts of result[] to left,
        result[ lenRes - j ] = 0;                         //terminator
}
return result;  }
```

Section 10.7: Brief Description of the Controller Program: Libraries

The controller consists of keypad, text LCD, temperature sensor, and the ATMEGA328P microcontroller (uC) for its computing capability, USART (TTL RS232), timer counter, and ADC. The 328P is programmed in C++ but primarily uses functions and methods found in C (without classes and other fancy constructs) [10.9-12]. The brief discussion in this section covers the constructions more exclusive to the FE5650A controller. The reader will need to refer to the source code in Appendix 6 and to the Microchip-Atmel ATmega328P microcontroller (uC) data sheet [10.13].

The remaining sections can be skipped if the reader does not intend to modify the microcontroller source code.

Topic 10.7.1: KeyPad4x4.cpp

The uC scans a voltage pulse across rows R1-R4 of the keypad and watches for a return pulse from one of the keypad columns C1-C4. The key is debounced in software. The row and column corresponding to a key closure is converted to an ASCII code. Most of the keypad routines are found in the keypad library (see Appendix 6).

The most basic keypad routine

```
char keyRead( uint8_t row, uint8_t col )
```

in KeyPad4x4.cpp simply determines if the key at ‘row’ and ‘col’ is pressed. First the uC needs to know to which uC pins the keypad rows and columns connect. This is handled by the arrays ‘pinsRows[ ]’ and ‘pinsCols[ ]’ both of which are defined in the File Global Area of the KeyPad4x4.cpp. The File Global Area is the region of main.cpp above ‘main’ but below the top where the first few declarations can be made. By the way, those variables declared in the File Global Area cannot be directly accessed in other libraries/modules; they are global only to those routines on the same page. Here are the declarations for pinsRows and pinsCols:

```
static uint8_t pinsRows[4] = {0, 1, 2, 3}; //PortB pin0 addresses Row1 (the keys labeled as 1 2 3 A) … etc.
static uint8_t pinsCols[4] = { 2, 3, 4, 5 }; //PinD pin2 queries Col1 (the keys labeled as 1 4 7 *) … etc.
```

Arrays allow the designer freedom to choose any pins in a given uC port to connect the rows and cols of the keypad. The array is viewed as a function mapping the index i to a port pin such as pinsCols[i]. The index of pinsRows[i] where i= 0 to 3 corresponds to row number. For example, consider pinsCols[col] where col is the array index instead of i. Index col=0 corresponds to column C1 connected to PORTD pin 2 (PIND for input), and index col=1 corresponds to column C2 connected to PORTD pin 3 (PIND for input), and so on. Similarly, consider pinsRows[row]. Index row=0 corresponds to keypad Row R1 connected to PORTB pin 0, and index row=1 corresponds to Row R2 connected to PORTB pin 1 and so on. The KeyPad4x4.h file associates PORTB with the rows using the compiler directive #define and the token PORT_KEYROW. So PORT_KEYROW should be considered to be synonymous with PORTB. The program associates PIND (meaning, 'input') with the columns using the token PORT_KEYCOL, which should probably be written as PIN_KEYCOL but wasn't. So now another routine placing a logic 1 (i.e., +5V) on the physical pin corresponding to the 'row'. The same routine calls keyRead(row,col) and stops if a logic 1 is detected (i.e., a key is pressed). In nearly all of the routines, the variable 'char key = 0' defines the null character since 0 (zero) is the null char in the ASCII table (see Table 6.2 in Section 6.2) and it is also the terminator for strings. For the keypad routines, 0 is returned when none of the keys have been pressed.

The keyRead(row,col) returns the ASCII code corresponding to the pressed key. The routine defines two variables

```
uint8_t rowPin = pinsRows[row];     // select a row
uint8_t colPin = pinsCols[col];     // select a col
```

The uC then places logic 1 (i.e., 5V) onto the selected rowPin on the uC pin using

```
PORT_KEYROW |=  1 << rowPin;  // apply +5V to row
delayK_ms(1);     // wait for voltage to stabilize
```

A delay of 1mSec is added to make sure the voltage has stabilized. Next, if the logic 1 appears on the specified column, the program executes a delay until it detects a logic zero after a 20mSec delay to allow for key debouncing.

```
while( ( PORT_KEYCOL & ( 1 << colPin ) ) == ( 1 << colPin ) )  // if yes then set key
{           key = keys[caps][row][col];
            delayK_ms(20);  }            // make sure bounce stops
```

Notice the array keys[caps][row][col] in the above code. The array converts the pressed key to an ASCII code for the 'key' variable. The 'caps' index can be either 0 for lower case or 1 for upper case. The array is defined as 'static' in the keyRead(row,col) routine where the elements provide the ASCII code:

```
static uint8_t keys[2][4][4] = {
        {
                { '1', '2',  '3', 'z' },  // 1      2       3       A='z' => caps Lock
                { '4', '5',  '6', 't' },  // 4      5       6       B='t' => toggle Row
                { '7', '8',  '9', 'a' },  // 7      8       9       C='a' => autoSend
                { '.', '0',  'c', 's' }   // *=.    0       #=<cr>  D='s' => select ..prev menu
        },
        {
                { 'B', 'U',  'E', 'z' }, // 1=Backsp       2=Up    3=ESC   A=lock
                { 'L', 0,    'R', 't' },  // 4=Left        5=na    6=Rght  B= t
                { 'M', 'D',  'C', 'a' }, // 7=Menu         8=dwn   9=Clr   C =auto send
                { '.',  0,   'c', 's' }  // *=.            0=na    #=<cr>  D=s=select
        }
        };
```

The first index for the keys array refers to the Caps Lock condition (see below). If the caps are not locked to 'on' then the index is caps= 0 otherwise it is 1. This caps index provides access to either the top 4x4 matrix for the index caps=0, or to the bottom 4x4 matrix for index caps=1. So for example, if the caps have not been locked (i.e. caps=0), and a key was found at index row=1 and index col = 2 then the key pressed was '6'. Notice the matrix elements can be set to any 8bit number including the typical scan codes for a microsoft keypad. The controller only needs a number to associate an action. For example, caps=1, row=0, col=1 returns a 'U' which the uC will interpret as the 'up' arrow on the keyboard. Some keys return the same number regardless of the caps lock.

'z'=caps lock, 't'=toggle LCD row top to bottom and vice versa,

'a' = autosend any change of the set frequency, 's' did stand for select, 'c' = carriage return.

The other entries for the caps=0 mean what they show. The entries for caps=1 have the following meaning

'B' = backspace, 'U'=up arrow, 'E'=escape, 'L'=left arrow, 0=undefined,
'R'=right arrow, 'M'=menu?, 'D'=down arrow, 'C'=clear

Next the case of key='z' for caps lock is intercepted so that the static variable 'caps' can be toggled between 0 and 1 for the array index and to set the caps-lock LED.

```
if (key == 'z') {caps ^= 0b01; key = 0;}          // specially handle caps ... toggle caps but key=0
if (caps == 0) { PORT_CAPSLOK &= ~(1 << PIN_CAPSLOK);} // set cap Lock LED
else { PORT_CAPSLOK |= (1 << PIN_CAPSLOK);  }

PORT_KEYROW &=  ~(1 << rowPin);          //apply 0V to row
return key;
```

Finally, the voltage to the row and col returns to logic 0 (i.e., 0V).

The routine `'char keyPad(void)'` scans the entire keypad ONCE to determine if a key is pressed and then returns the scan code.

```
while (  (row < 4) && (key == 0) )                    // for each row
        {       col = 0;
                while ( (col < 4) && (key == 0) )      // check each column until key pressed
                { key = keyRead(row,col); col++; }
        row++;  }
```

For each value of the row index, each column is checked for a key press. The column index is set to zero using col=0 after changing the row index so that all four columns will be checked for each row. The number of keypad scans can be controlled by software.

Finally, `char keyGet(void)` repeatedly scans the keypad until a key is pressed.

Topic 10.7.2: LCD16x2_ST7032.cpp

As previously discussed, the controller incorporates an Ada Fruit HD44780-based LCD. However, an ST7032 text LCD would be a good substitute because of the wider voltage range of 3-5V but different pinout. Initially, we anticipated using the ST7032-base LCD and hence so-named the library but ended up using the HD44780-based LCD. We hope this does not cause confusion. The LCD displays two lines of text with 16 characters per line. As an essential feature, it can be programmed for custom characters. The display requires two data lines for Enable and Data/Command and it can operate in either a 4 or 8 data line mode; although a 2-line I2c (i.e., TWI) would be better. The enable line is used as a type of clock input whereby pulsing/strobing the line from logic 0 to 1 and back has the effect of transferring data/commands into the ST7032 and executing the commands.

The code in this LCD library integrates the LCD functions described in its data sheet. The .h file contains the #define directives and the related tokens. The MASK tokens are used to mask off either the upper or lower 4 bits of the PORTB data transfers. The seven tokens including COM_WAKEUP through COM_ENTRYMODE correspond to numbers required by the display to set specific modes. The data sheet can be consulted for the exact functions.

The LCD16x2_ST7032.cpp provides the code for the various functions. Often times, it's the initialization of the displays that causes the most trouble especially for some of the graphical displays. The text display initialization is straight forward except possibly for switching from the 8 to the 4 data line mode and the timing. The display does include a ‘busy’ line that indicates the display cannot accept new data and commands; however, we eliminate the ‘busy’ line and instead insert delays into the code. The duration of the delay matches the time required for the LCD to execute commands based on the times stated in the data sheet. Many delays are in the sub mSec range which can be promoted to 1mSec.

Consider the initialization routine ’void LCD_init(void)‘ found in LCD16x2_ST7032.cpp. The LCD can be started by issuing the command ’COM_WAKEUP‘, which is 0x30, followed by a series of enable pulses. Notice 0x30 has non-trivial data only on the upper 4 bits which can be sent by the 4 data lines connected to the LCD. Immediately afterwards, the 4 bit mode must be set using

```
PORTDAT = (COM_INIT4Bit>>4) & MASK_DATA;
```

otherwise the LCD will expect to see the 8 bit commands whereas there are only 4 data lines. The 4-bit mode command in the .h file is defined by

```
#define COM_INIT4Bit 0x20
```

which shows the command only uses the upper 4 bits. Once the 4-bit mode has been set, the display simply clocks in two 4-bit packets to complete 8-bit commands. The code

```
L_command(COM_4BIT2LINE5X8,1);
```

sets the text size to 5 pixels wide by 8 pixels tall. Finally, the initialization routine constructs custom characters each time the LCD is powered-on. An array ‘char p[8]’ holds the bits representing custom characters but an additional routine L_buildChar(num,p) creates them. Each row of p[ ] holds the bits for a given row of the character starting at the top, but given the character size of 5x8, only 5 bits are used. A bit = 1 causes the corresponding pixel to be colored. For example,

```
char p[8]  = {0x04, 0x0E, 0x0E, 0x0E, 0x1F, 0x00, 0x04, 0x00};
```

produces the bell pattern shown in Table 10.1. A number of patterns have been included as examples – only those patterns for ‘degrees’ and ‘arrows’ need to be retained for the program.

HEX	Right 5 bits					Color pattern				
0x04	0	0	1	0	0					
0x0E	0	1	1	1	0					
0X0E	0	1	1	1	0					
0X0E	0	1	1	1	0					
0X1F	1	1	1	1	1					
0X00	0	0	0	0	0					
0X04	0	0	1	0	0					
0X00	0	0	0	0	0					

Table 10.1: The HEX numbers stored in the array p[ ] produce the bell pattern at the right side. P[0]=0x04, p[1]=0x0E (and so on) start at the top of the bell pattern. The nonzero bits indicate pixels that should be colored.

The function to pulse the display enable line:

```
void L_enablePulse(void)
{       PORTCON |= BIT_E;         // Set enable bit to high
        L_delay_ms(1);            // E pulse width
        PORTCON &= ~BIT_E; }      // New Haven claim: trigger on falling edge
```

As previously mentioned, a pulse on the display’s Enable pin causes the data or command to transfer into the LCD. The instruction PORTCON |= BIT_E; writes logic 1 to the Enable line. The 1mSec delay maintains the voltage and then the instruction ’PORTCON &= ~BIT_E‘ returns the Enable line to zero to complete the pulse.

One of the most basic routines simply writes a single character to the display. Consider the routine

```
void L_writeChar(char c)
```

found in the LCD16x2_ST7032.cpp file. The process of writing a character requires assertion of data and control lines at the ports specified by the tokens PORTDAT and PORTCON, respectively. The .h file defines PORTDAT as PORTB and PORTCON as PORTD. First, the uC sets up the 'write chars' by setting control line for the RS/CD pin on the LCD to logic 1

```
PORTCON |= BIT_CD;
```

and then the uC writes the character to the data lines. Second, the upper four bits of a character c (8 bit ASCII code) is placed on the data lines by the instruction

```
PORTDAT |= (c >> 4) & MASK_DATA;
```

Third, the Enable control line is pulsed using the routine

```
L_enablePulse();
```

to clock-in the upper 4 bits of c. Fourth, the routine writes the lower 4 bits of c using

```
PORTDAT |= c & MASK_DATA;
```

and finally sends a pulse on the Enable line to clock-in the bits. This last clock pulse also causes the LCD to execute a write so the character c appears on the LCD at the *last position of the cursor*, which is not necessarily visible. After the write, the cursor advances to the next position. The uC writes strings using the function `void L_writeStr(const char* str)` which simply calls the function `L_writeChar(c)` for each character c in the string. As a note, the port pin assignments to the LCD are 'fixed' in the sense that the LCD DATA and CTRL lines must be connected to specific uC pins unlike those for the keypad; however, while the data lines connect to the uC port pins 0, 1, 2, 3, any port will work.

Given that the LCD writes a character at the position of the cursor, a routine should be developed to place the cursor at a desire position on the LCD. The following routine sets the cursor

```
void L_cursorLinePos(uint8_t lineIndex, uint8_t posIndex )      // set cursor to 'line' and 'Pos'
{       int ddramAddr = 0b10000000;                             // address line 0 … binary number!
        if (lineIndex != 0) {ddramAddr = 0b11000000;}           // address line 1
        if (posIndex > 15) {posIndex=15;}                       // keep position in 0 to 7
        ddramAddr += posIndex;                                  // calculate address
        L_command(ddramAddr,1);         }                       // send address and execute
```

to the top or bottom line, lineIndex = 0 or 1 respectively, and at position posIndex = 0 through 15. The LCD chooses the cursor position according to a 'base' address of 0b10000000 for line 0 and 0b11000000 for line 1 on the LCD. Notice these numbers have been written as the binary representation rather than HEX. The position on the LCD is found by adding the position index to the base address for the line.

```
ddramAddr += posIndex;
```

Notice the addresses for Line 1 can add numbers as large as 0b00111111 which is 2^6 = 64 base 10 since 0b11000000 + 0b00111111 = 0b11111111 which is the maximum for an 8 bit register. And so each line can in principle support up to 64 characters but this particular display only uses 16 per line. Finally the routine calls the function `L_command(ddramAddr,1);` which will be discussed next.

Next consider the command function in LCD16x2_ST7032.cpp.

```
void L_command(char cmd, int post_mS)

{       PORTDAT &= ~MASK_DATA;                  //clears only data pins
        PORTCON &= ~MASK_CTRL;                  //clears control pins

        PORTDAT |= (cmd >> 4) & MASK_LOW;       // upper 4bits on data lines
        L_enablePulse();                        // pulse clocks in the command bits

        PORTDAT &= ~MASK_DATA;                  // clears 4 data bits
```

```
PORTDAT |= cmd & MASK_DATA;           // place lower 4 bits on data lines
L_enablePulse();                      // clock in bits and execute command

L_delay_ms(post_mS);    }             // delay time for command to execute
```

Peruse the various functions in LCD16x2_ST7032.cpp and notice that only the function for writing a character to the display is free of the L_command. The difference between 'data' (i.e., character) and a 'command' is that the RS/CD line is placed at logic 1 for data and logic 0 for other 'commands'. Setting the 'display modes' constitute 'commands' while displaying text makes use of previous issued 'commands'. The routine clears the RS/CD control line (i.e., sets it to logic 0) by

```
PORTCON &= ~MASK_CTRL;
```

As shown in the ST7032 data sheet, commands 'cmds' consist of 8 bits. The routine first places the upper 4 bits on the data lines

```
PORTDAT |= (cmd >> 4) & MASK_LOW;
```

and clocks them into the LCD by pulsing the enable line. It then clocks in the lower 4 bits using

```
PORTDAT |= cmd & MASK_DATA;
```

This time, pulsing the enable line also executes the command which changes one of the display modes. The remaining functions are similar.

Topic 10.7.3: USART.cpp

The USART provides TTL-RS232 serial communication with the FE5650A at a user-specified baud rate. For the ATMEGA328P, the USART receiver (Rx) and the transmitter (Tx) correspond to pin D0 (pin 2) and pin D1 (pin 3), respectively, on the 328P DIP. The various USART register settings can be found in the ATMEGA328P data sheet/document. The USART supports either a polling or interrupt mode for either/both the Rx/Tx and either mode would be suitable for the present purposes. However, use of the interrupt makes it easy for the data received from the FE5650A to be placed in a circular buffer for a later read or scan as needed. In addition, the main.cpp routines implement a timeout timer so the program doesn't hang-up if the FE5650A doesn't properly respond to an enquiry. Of primary interest are the routines for the USART configuration, circular buffer, basic character and string transmission, and the interrupt service routines (ISR),

The configuration routine `USART_config( double Fcpu, double baud )` configures/reconfigures the USART as needed because the first two lines disable the receiver, transmitter and interrupt

```
UCSR0B &= ~( (1<<RXEN0)|(1<<TXEN0) ); //disable receiver and transmitter
UCSR0B &= ~(1<<RXCIE0);               //disable interrupt
```

The UBRR0 register is set for the desired baud using a calculation similar to that provided in the data sheet

```
UBRR0 = floor( (uint16_t) ( ( Fcpu / ( 16 * baud ) ) - 1 ));
```

A motivated reader could modify the routine to search for the optimal baud rate for the FE5650A. The UCSR0C register is set for 8 data bits, 1 stop bit and no parity. The variables associated with the circular buffer are initialized and the global interrupts are enabled using 'sei()'. The global interrupts could be cleared with 'cli()' but it won't be needed here. The USART is not actually enabled until required by other routines.

The 'send character' routine `void USART_SendChar( uint8_t data )` first checks whether the transmit register is empty and ready to accept a new character to transmit and, if so, then second, loads the UDR0 with the new character. The 'send string' routine

```
void USART_SendStr(const char* str)
```

simply sends the individual characters using the SendChar(data) routine.

As mentioned, the USART triggers an interrupt when the data register receives a character. The interrupt service routine `ISR(USART_RX_vect)` can be found in the USART.cpp library file.

```
ISR(USART_RX_vect)
{       // the interrupt signals when a byte is received by the USART
        // store the data into the buffer
        USART_Buffer[USART_Next_Write_Loc++] = UDR0;
        if(USART_Next_Write_Loc == USART_BufferLen) {USART_Next_Write_Loc=0;}  }
```

Each received character triggers the interrupt and the ISR places the character into the array named 'USART_Buffer[ ]'. The array should be declared 'volatile' as should any variable changed within an ISR otherwise the compiler might remove it. The array is declared in the global area of the USART.cpp file. As a reminder, those variables declared in the file global area cannot be directly accessed in other files and libraries/modules; they are global only to those routines on the same page. The buffer array

```
volatile char USART_Buffer[USART_BUFFLEN] = {0};
```

has 50 elements because of the directive and token #define USART_BUFFLEN 50. Again, notice 'volatile' alerts the preprocessor/compiler not to optimize away the array. Two variables provide markers into the array

```
volatile uint8_t USART_Next_Write_Loc = 0;
volatile uint8_t USART_Next_Read_Loc = 0;
```

The ISR uses the variable USART_Next_Write_Loc to mark the next position in the array available to receive a character. When the variable has the value `USART_BufferLen` then it is set to 0. The resetting to 0 provides a wrap-around function. So for example, if the next position to be read is

```
USART_Next_Read_Loc = 48
```

and the wrap around has occurred giving, for example,

```
USART_Next_Write_Loc = 2
```

Then a read routine can read entries 48, 49, 0, 1. So no further new characters can be read when

```
USART_Next_Read_Loc == USART_Next_Write_Loc
```

This relation shows a maximum of 50 characters can be read until the Write_Loc passes the Read_Loc after a wrap-around.

The situation with the circular buffer can be more clearly seen by considering the USART_ReadBuffChar( ) functions.

```
char USART_ReadBuffChar(void)
{       char Ch = 0;
        if(USART_Next_Read_Loc != USART_Next_Write_Loc)
        {       Ch = USART_Buffer[USART_Next_Read_Loc++];
                if(USART_Next_Read_Loc == USART_BufferLen) {USART_Next_Read_Loc=0;}  }
        return Ch;  }
```

So long as the two markers have differing values, a routine can read another array location and store it in 'Ch'. If the Read marker-value coincides with the buffer length, the routine resets the marker to 0 and increments continue from there.

Topic 10.7.4: TC16.cpp

The program uses the 16 bit Timer Counter, termed TC16 here, for two basic purposes. The first purpose provides a 500mSec timeout function for the USART. If the USART fails to communicate with the FE5650A, the TC16 triggers an interrupt that sets a time expired flag. The second purpose triggers the ADC every 4 seconds when measuring the temperature of the physics package in the FE5650A. The program implements the second purpose by configuring the ADC to trigger from the TC16 interrupt while disabling the TC16 ISR from performing some other functions through the ISR. Some 'extra' TC16 functions have been included but keep in mind only those TC16 functions associated with the present application have been tested.

The TC16_config routine, namely

```
void TC16_config(double mS, double fcpu, bool callFunction, bool setFlag, bool stopInISR, uint16_t msCounterIncr)
```

initializes (and re-initializes) the TC16 for the number of milliSeconds (mS up to just over 4000), and enables the ISR to set a flag (setFlag) in TC16.cpp, to stop the TC16 from within the ISR (stopInISR), and to access times larger than 4000 mS by setting the variable 'msCounterIncr' to numbers other than zero. The timer can be reinitialized using the same routine as initializes/configures it since the routine stops the timer and disables the interrupt near the start of the routine (and no need to disable the global interrupts)

```
TCCR1B &= ~ ( (1 << CS12) | (1 << CS11) | (1 << CS10) );      // stop timer; all CS zero
TIMSK1 &= ~(1 << OCIE1B);                                      // disable interrupt
```

The datasheet shows the bits CS10-CS12 normally set the prescaler value but can also be used to start and stop the timer.

The TC16_config routine passes the received booleans (i.e., flags) to the corresponding booleans in the TC16_config global level.

```
flag_callFunction = callFunction;      // Save booleans
flag_setFlag = setFlag;
flag_stopInISR = stopInISR;
```

Keep in mind that other libraries/modules/pages cannot access these variables without additional treatment. We add methods to TC16_config to return the values. The situation is similar to the idea of classes and properties for object oriented programming. The registers TCCR1A and TCCR1B are set for 'normal' port operations and register ICR1 is set to 'TOP' for the timer-counter compare match mode CTC. TC16 counts (prescaled) clock pulses from 0 to the TOP value in the ICR1 register and then resets; consequently, the ICR1 register must be loaded with the proper value to provide the timer duration of mS milliseconds. Interrupt vector _VECTOR(12) (a.k.a, `TIMER1_COMPB_vect)` for the ISR can be used to trigger the ADC and this vector corresponds to the ICR1 as TOP.

The time interval is controlled by setting the number of (prescaled) clock pulses to 'count'.

```
if (msCounterIncr < 1) //triggers every mS millisecs
    {  maxCount = 0;
       ICR1 = (uint16_t) (  (mS/1000.0) * (fcpu/1024.0) - 1  );  }
else                        //will trigger every msCounterIncr and adds 1 to count until MaxCount
    {  maxCount = (uint16_t) ( ( (float)mS + 0.1) / (float)msCounterIncr);
       ICR1 = (uint16_t) (  (msCounterIncr/1000.0) * (fcpu/1024.0) - 1  );  }
```

For typical operation, msCounterIncr = 0, and so the ICR1 value can be calculated using the desired mS and the CPU clock rate (fcpu=16MHz) and the prescaler value (1024). For longer times, the TC interrupt will trigger every msCounterIncr (such as 1000mS = 1 second) at which time the ISR increments a 'count' variable until count > maxCount. Note variables with ms part of the name means milliseconds. The variable maxCount is given a value to match the long delay. For example, if msCounterIncr = 1000, then the interrupt fires every 1 sec and increments 'count'. If maxCount =100 then the ISR won't do anything of value until count > maxCount which means the timer interval will be maxCount * msCounterIncr = 100 seconds. Finally, the initialization routine enables the timer and global interrupts.

The operation of the interrupt shows the use of the extended timing range and the booleans.

```
ISR ( _VECTOR(12) )      // can write TIMER1_COMPB_vect instead of _VECTOR(12)
{           count++; //PORTB ^= (1<<4);
            if (count > maxCount)
            {           if(flag_stopInISR) TC16_Stop();
                        if(flag_setFlag) flag_TimeExpired = true;
                        if(flag_callFunction) TC16_calledFunction();
                        count = 0; }
}
```

The ISR executes for the TIMER1_COMPB_vect, which can also be written as _VECTOR(12). For 'non-extended' typical operation, maxCount = 0 and so when the ISR runs, the various booleans (i.e., flags) will be queried which can cause a function to be executed. For example, in the case of flag_setFlag, the Time Expired flag will be set. For the case of extended time, the ISR simply increments a 'count' variable until count exceeds the maxCount . Notice the ISR appears in TC16.cpp rather than main.cpp and that causes some additional programming since the variables in such a library don't automatically communicate with main.cpp.

Some comments need to be made on the 'locally-declared' global variables and functions especially in relation to the ISR – how about that oxymoron for objects with restricted scope being called locally-declared global. Consider first the boolean 'flag_TimeExpired' which is declared for use by TC16.cpp. The main.cpp knows of methods/functions in TC16.cpp by virtual of the header file TC16.h through the statement in main.cpp of #include "TC16.h". The values stored in those variables declared in the TC16.cpp globals area can be communicated to main.cpp by using the methods/functions similar to the 'properties' for classes. The following function

```
bool TC16_isTimeExpired(void)
{ return flag_TimeExpired;  }
```

returns the value of flag_TimeExpired to a calling routine in main.cpp. Second, as another issue, the ISR should be able to call a function in main.cpp (or elsewhere). One procedure to do so consists of defining a function name 'TC16_calledFunction()' in main.cpp with a declaration in main.cpp globals and in the TC16.h file. The ISR can then call the function TC16_calledFunction and expects it to execute in main.cpp.

As a final note, TC16 can be started and stopped using the two routines

`void TC16_Start(void)` and `void TC16_Stop(void)`

that operate, respectively, when they set TCCR1B to the 'prescaler value' and 'zero', respectively.

Topic 10.7.5: ADC328.cpp

The ADC_Config routine sets various modes for the Analog to Digital Converter (ADC)

```
void ADC_Config(ADC_Ref aRef, enum ADC_TrigSrc aTrig, enum ADC_Prescale aPresc,
                uint8_t muxChan, bool aAutoTrig, bool aInterruptEnable, bool callFunction, bool FrawTmv )
```

Notice the argument list for ADC_Config requires several enums, namely ADC_Ref, ADC_TrigSrc, and ADC_Prescale. Both the calling routine and the configuration routine need to have access to the enums. For this reason, the enum definitions have been placed in the .h header file ADC328.h – the contents of header files are dumped into the spot where the #include appears. The integer values for the enums have been adjusted to match the requirements of the uC registers that consume them. The Config routine transfers the booleans to local-global variables. The Config routine sets the clock rate for the ADC between 50kHz and 200kHz using the ADC prescaler

```
ADCSRA |= aPresc;           //set prescalar; the ADC needs 50kHz to 200kHz
```

It would be good to include a routine to calculate the prescaler value since the present one assumes a cpu clock rate of 16MHz. The calculation would simply scan through the prescale values and select the one closest to the range of 100kHz to 150kHz. The incoming booleans are used to set the Auto Trig and Interrupt Enable bits in the ADCSRA

```
if (aAutoTrig) ADCSRA |= ( 1 << ADATE );
        else ADCSRA &= ~( 1 << ADATE );

if (aInterruptEnable) { ADCSRA |= ( 1 << ADIE); }
        else { ADCSRA &= ~( 1 << ADIE ); }
```

Finally, the global interrupts are enabled by calling sei() and if desired, they can be disabled using cli().

As with the previous topic, the ADC.cpp includes variables for use with the ADC328.cpp routines including the ISR:

```
ISR(ADC_vect)
{       flag_ADCdone = true;
        If ( flag_FrawTmv )
                { adcResult = uint16_t ( ( 1100.0 * ADCW ) /1024.0 ); }
        else
                { adcResult = ADCW; }

        If ( flag_allowCallFunction ) ADC_calledFunction( adcResult );    }
```

The ISR executes when the ADC completes a conversion (and the interrupts are enabled). The ISR sets the flag 'flag_ADCdone' even if it's not needed anywhere. Of particular importance, note that the ADC result can be either the number of milliVolts or the raw ADC measurement in bits. The choice is controlled by flag_FrawTmv, which can be read as Flag False=>Raw, True=>mV, means to set flag_FrawTmv=true for the mV version and set flag_FrawTmv=false for the bits version. The variable adc_Result appears in the globals specific for ADC328.cpp. Similar to the TC16 ISR, the ADC ISR can call the function ADC_calledFunction(adcResult) in main.cpp and pass the result of the measurement to the caller.

The config routine disables the ADC and clears the register ADMUX that specifies ADC channel and reference.

As a final but important comment, only those methods used with main.cpp have been tested beyond 'they seem to work ok'.

Section 10.8 References

[10.1] Software accompanies the book. Preprogrammed chip see Reference 10.25.

Instructables: The following instructables show the basic methods of working with the Atmel microcontrollers and Atmel Studio. As a note, toward the bottom of the FIRST page, there should be a link to download a pdf version that does not require a person to be a subscriber to the instructables.com. Also refer to the references in the linked sites.
[10.2] https://www.instructables.com/id/Atmel-Startup-1-Atmel-Studio-and-Programmer/
[10.3] https://www.instructables.com/id/Atmel-Startup-2-Microcontroller-Circuits-and-Fuses/
[10.4] https://www.instructables.com/id/Atmel-Startup-3-Binky-One-PORT-PIN-DDR-and-LED/
[10.5] https://www.instructables.com/id/Atmel-Startup-4-Blinky-Two-Switches-Pull-Up-Resist/
[10.6] https://www.instructables.com/id/Atmel-Startup-5-Lifeline/

[10.7] Without a doubt, visit avrfreaks.net for tutorials as well as project help
https://www.avrfreaks.net/forum/newbie-start-here?name=PNphpBB2&file=viewtopic&t=70673
https://www.avrfreaks.net/forums/tutorials?name=PNphpBB2&file=viewforum&f=11

[10.8] The SparkFun website has a variety of tutorials ranging from electronics to software:
https://learn.sparkfun.com/tutorials/tags/concepts

[10.9] The classic masterpiece:
B.W. Kernighan, D. M. Ritchie, "The C Programming Language", Prentice, 2nd Ed. (1988).

[10.10] Good C++ review after reading the Kerninghan-Ritchie book and some background in OOP. "C++ Super Review" written by The Staff of Research and Education Association, published by Research and Education Associated (2000). ISBN: 0878911812. Amazon.com for $8.

[10.11] A good C++ tutorial site can be found at
https://www.cprogramming.com/tutorial.html#c++tutorial

[10.12] Good overview of Atmel concepts and C but uses the AVR Butterfly:
J. Pardue, "C Programming for Microcontrollers Featuring ATMEL's AVR Butterfly and the free WinAVR Compiler", Published by Smiley Micros (2005). ISBN: 0976682206. Amazon.com

[10.13] ATmega328P data sheet
http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf

[10.14] Microchip-Atmel Studio 7 (free of charge):
https://www.microchip.com/mplab/avr-support/atmel-studio-7

[10.15] Microchip-Atmel ICE Programmer:
Digikey.com stock number: ATATMEL-ICE-ND, mfg #ATATMEL-ICE

[10.16] The F64.c/h can be found on the Internet through the following links
https://www.avrfreaks.net/comment/596905#comment-596905
The original was written by 'Detlef _. (detlef_a)' on or about '02.12.2007'. It was posted at https://www.mikrocontroller.net/topic/85256 (need to translate from German). There have since been updates/edits by subsequent authors such as Florian Königstein. Also see https://www.avrfreaks.net/forum/working-64-bit-floats-atmega?skey=64 bit float extension
Be aware that some type-o bugs have been eliminated for the FE5650A software here.

[10.17] Some information on floating point number format, and examples
https://en.wikipedia.org/wiki/Single-precision_floating-point_format
https://en.wikipedia.org/wiki/Double-precision_floating-point_format
http://mathonline.wikidot.com/ieee-single-precision-floating-point-format-examples-1
https://www.doc.ic.ac.uk/~eedwards/compsys/float/
https://cs.nyu.edu/courses/spring16/CSCI-GA.1144-001/IEEE_754_Note.pdf

[10.18] 1N4001 data sheet
http://www.comchiptech.com/admin/files/product/1N4001-G%20Thru.%201N4007-G%20RevB.pdf
[10.19] Schottky diodes
https://www.st.com/content/ccc/resource/technical/document/datasheet/26/db/14/60/52/47/47/5b/CD00001625.pdf/files/CD00001625.pdf/jcr:content/translations/en.CD00001625.pdf
[10.20] Alkaline battery reverse bias current
https://electronics.stackexchange.com/questions/416420/allowable-reverse-current-into-alkaline-battery
[10.21] Traditional/Standard RS232 and TTL-RS232 https://www.sparkfun.com/tutorials/215
[10.22] LM35DZ data sheet
www.ti.com/lit/ds/symlink/lm35.pdf?HQS=TI-null-null-mousermode-df-pf-null-wwe

[10.23] discussion of ADC input impedance
https://electronics.stackexchange.com/questions/67171/input-impedance-of-arduino-uno-analog-pins

[10.24] ELF file programs FLASH and FUSES:
https://www.kanda.com/blog/microcontrollers/avr-microcontrollers/atmel-studio-elf-production-files-avr/

[10.25] Source code, listings, downloads, flash drives, preprogrammed microcontroller:
(A) Chapter 10 and Appendix 6 provide the source code for the C/C++ program
(B) Appendices 7-9 provide the PC software
(C) Check EBay.com in connection with the title of this book: software and preprogrammed MEGA328P
(D) check www.AngstromLogic.com for current offerings, or write to Sales@AngstromLogic.com
Keep in mind that Angstrom Logic, LLC is not setup/able to answer questions on the book or software.
(E) Check Amazon in connection with the title of this book.
(F) Note: The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the words) and the contents are copyrighted with all rights reserved (also see front copyright page). The software/firmware is scanned for viruses before posting. Download only from secure sources and rescan for viruses before using.

Appendix 1: HEX Math and Calculator2

For work with microcontroller registers, a good Hexadecimal calculator (a.k.a., programmer's calculator) is indispensable. Probably most people would know the hexadecimal number system as base 16 numbers. Some calculators are available for the smart phone and Windows 10. The calculator should show at least 10 or 11 digits and be capable of displaying the number in at least Hexadecimal (i.e., base 16, HEX) and Decimal (base 10, DEC) for application to the Rubidium Frequency Standard (RFS). The best software-based calculator tested is called 'Calculator2' and it is available for iPhone, Android and Windows10 (and probably other platforms). None of the calculators tested can display or calculate fractional HEX numbers; however, fractional HEX won't be necessary since the FE5650A uses only whole numbers. The present appendix introduces the HEX numbers, by-hand calculations and the Windows calculator. The internet has many tutorials for HEX math for more information (c.f., [A1.1-2]).

Section A1.1: HEX Digits

As a brief review, base 10 (i.e., decimal, dec) uses ten distinct symbols

0, 1, …, 9

Adding the two digits 1 and 9 produces a double digit configuration of 10. A Dec number such as wxy.z can be represented as powers of ten

$$wxy.z = w * 100 + x * 10 + y * 1 + \frac{z}{10}$$

or better

$$wxy.z = \ (w * \ 10^2) + (x * 10^1) + (y * 10^0) + (z * 10^{-1})$$

where the 10 is the decimal 10 as usual.

The base 16 numbers (a.k.a., Hexadecimal, HEX) make use of 16 distinct symbols

0, 1, …, 9, A, B, …, F

The Dec and Hex numbers compare as follows

Base 10 (Dec):	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18 …	255	256 …
Base 16 (HEX):	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12 …	FF	100 …

As can be seen, HEX represents each number in the range 0-15 (base 10) by a unique symbol. Adding together the digits 1 and F produces 1+F=10 Hex. Similarly 2+F=11 Hex and F+F=1E Hex. As another example, D+5=12 Hex. These can be found using the conversions.

For the purposes of notation (and computer programming), **the HEX number is most often preceded by '0x'** to distinguish between Hex, Base 10, and Base 2 (a.k.a., binary). Sometimes the distinction is clear without the 0x and at other times, no. For example 110 could be a number in any of the bases. Consider that 0x110 means 272 base 10, but 0b110 (base 2, binary) means 6 in base 10. Once in a while, hex might be seen written as h110 or 110h. Sometimes the base will be written as a subscript on the number such as 110_{16} for hex or 110_2 for binary.

By hand, a number in hex such as abc.de$_{16}$ is converted into decimal by using powers of 16. Let Dec(hex#) be notation for mathematical operations (i.e., function) that converts the base 16 number 'hex#' into a decimal base ten. The mathematical operation is defined as

$$\text{Dec(abc.de}_{16}\text{)} = (a * 16^2) + (b * 16^1) + (c * 16^0) + \frac{d}{16^1} + \frac{e}{16^2}$$

or in another form

$$\text{Dec(abc.de}_{16}\text{)} = (a * 256) + (b * 16) + (c * 1) + \frac{d}{16} + \frac{e}{256}$$

Example A1.1: Convert FE.1 to decimal

$$\text{Decimal FE.1} = (F * 16) + (E * 1) + \frac{1}{16} = (15 * 16) + (14 * 1) + \frac{1}{16} = 254.0625$$

A decimal number such as abc.de can be converted from Dec to Hex by using two stages. For the first stage, repeatedly divide the whole part by 16 and form the remainders into the Hex whole number. For the second stage, sequentially multiply by 16 and keep the whole number to form the hex fraction. Section A1.3 will provide the details.

Example A1.2: Convert the Dec number 254.0703125 to HEX.

Start with 254 and note remainder of 254/16 is 14 => E
The new number to be divided is 254 – 14*16 = 15
Divide the new number as 15/16 and note the remainder is 15 => F
Collect the remainders to form the whole part of the number: FE
Next work with the fraction part 0.0703125
Multiply the number by 16 and keep the whole part 0.0703125*16=1.125 => 1
Remove the whole part. The new number to be multiplied is 0.125
Multiply the new number by 16 and keep the whole part 0.125 * 16 = 2
Collect the whole parts to form 0.12 hex
Put the whole and fractional parts together: FE.12 hex

Section A1.2: Calculator2

Setting the output frequency Fout for a FEI Rubidium Frequency Standard (RFS) requires a really good calculator in the sense of displaying 14 decimal digits or more and also capable of displaying HEX digits. There are many HEX calculators (often termed 'programmer' calculators) but for our purposes, the Calculator2 accompanying Windows 10 appears to be sufficient in terms of digits and functions (scientific and programmer). The Windows 10 calculator used with these notes is

Calculator 10.1811.3241.0
© 2018 Microsoft.

None of the HEX calculators display fractional HEX numbers. The HEX numbers are primarily used with computers and microcontrollers (as is binary) and the microcontroller registers do not natively store fractions. Keep in mind that this calculator is the same as the Calculator2 previously mentioned at the start of this appendix.

Now fire up the windows calculator (Figure A1.1). Click the three lines (menu/options) on the upper left corner and select 'programmer' (the menu lines might be on the other side for iPhone and

Android). The HEX, DEC, OCT, and BIN refer to Hexadecimal Base 16, Decimal Base 10, OCTAL Base 8, and Binary Base 2, respectively. The keypad changes according to the selected label.

Click the HEX in the programmer calculator and notice the keyboard shows the 0-F symbols. Click D, for example, and notice D appears in the calculator window. But maybe more importantly, D appears next to the HEX label and 13 next to the DEC label which is the decimal version of the displayed hex. The HEX, DEC, OCT, and BIN will always show the current number written in the four different bases. We need this conversion capability to program the Rb unit.

Figure A1.1: upper left corner of the Windows Calculator.

We also need the large number of digits available in the *scientific* option for the calculator. Click the menu (lines) and select ‘scientific’. Consider the example of ‘2 to the power of 32’ (2^{32}, sometimes written as 2^32) which is a number cited in the Analog Devices DDS AD9830A data sheet and in the online FE5650A docs. To find the value, click 2 on the keyboard, then x^y in the upper function row, and then 32 on the keyboard and then the ‘=’. The value we need in the RFS programming is then

2^{32} = 4 294 967 296 decimal

As another example, go back to the menu (lines upper left) and select programmer and click the DEC. Type in 4,294,967,296 (result from previous calculation) and subtract 1. Notice in the HEX window the result of FFFF FFFF. Each ‘F’ represents 4 bits and so the entire number represents 32 bits which is the largest that can be transferred to the FE-5650A. Notice also that FFFF FFFF = $2^{32} - 1$ as used in the main chapters.

Try adding some numbers using the DEC part of the programmer calculator and observe the HEX window and vice versa.

Section A1.3: HEX Conversion Details

To convert a number to base 10 and back, it is easiest to use a calculator as described in a subsequent section. But it’s worth taking a quick look at the math behind the calculator buttons. For specific examples, refer to Examples A1.3-A1.6 below. A base 16 number consists of a sequence of hex digits adjacent to each other:

$$H_n H_{n-1} \cdots H_2 H_1 H_0$$

For the purpose of converting to base 10, the hexadecimal number is written in the base 10 equivalent as

$$y = (H_n * 16^n) + (H_{n-1} * 16^{n-1}) + \cdots + (H_2 * 16^2) + (H_1 * 16^1) + (H_0 * 16^0) \qquad \text{(A1.1)}$$

where each H_i is one of the hex digits {0, 1, 2, … , E, F} and the ‘16’ and the exponents are Dec (base 10). Obviously one must substitute the base 10 equivalent for the hex digit H_i

Base 16 (HEX):	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Base 10 (Dec):	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Refer to the Example A1.3 to see how it works

Base 10 numbers can be converted to base 16 by successively dividing the base 10 number by 16 and collecting the remainder of each division to form the base 16 number. Look at Equation A1.1 and assume that y is actually the base 10 number while we want to find all the Hi on the right side.

Start by dividing y by 16 which causes the right side to give the following result.

$$(H_n * 16^{n-1}) + (H_{n-1} * 16^{n-2}) + \cdots + (H_2 * 16^1) + (H_1 * 16^0) \quad \text{with remainder } H_0$$

Divide the result again by 16 to find

$$(H_n * 16^{n-2}) + (H_{n-1} * 16^{n-3}) + \cdots + (H_2) \quad \text{with remainder } H_1$$

The remainders can be collected after completing all of the divisions by 16 to form

$$y(decimal) = 0x\ H_n H_{n-1} \cdots H_2 H_1 H_0$$

which produces the hex number from the base 10 number. Refer to Example A1.4 below to see how this works.

Although not normally seen, base 16 numbers can form fractions in the form of digits after a 'decimal' point (maybe better to call it a hex point?)

$$y = H_0.H_{-1}H_{-2}\cdots \tag{A1.2}$$

The base 10 equivalent would be

$$y = (H_0 * 16^0) + (H_{-1} * 16^{-1}) + (H_{-2} * 16^{-2}) + \cdots \tag{A1.3}$$

Notice that multiplying a number such as in Equation A1.3 by 16 moves H_{-1} to the left of the decimal point when the number is a hex number. So, to convert a decimal number to fractional base 16, remove any digits to the left of the decimal point, and then successively multiply by 16 and collect the digits that move to the left of the decimal point to construct the Hex number. Refer to Example A1.6 below to see how it works.

Example A1.3: Convert base 16 number 'AB' to base 10. Using Equation A1.1, substitute the hex digits H_1=A and H_0=B.

$$(A * 16^1) + (B * 16^0) = (A * 16) + (B * 1) = 10 * 16 + 11 * 1 = 160 + 11 = 171$$

Or more simply stated, multiply each hex digit by successive powers of 16.

Example A1.4: Convert the base 10 number '161' to hex as follows. Divide the number by 16 to find a result of 10 and a remainder of 1 and so $H_0 = 1$. The result of 10 can be divided by 16 but the result is 0 with a remainder of 10. So convert 10 to the hex digit A and write $H_1 = A$. Collect the digits together and form the hex number as

$$H_1 H_0 = A1$$

Example A1.5: Convert the hex number 1.B1 to the base 10 equivalent. Use Equation A1.3 and substitute the hex digits H_0=1, H_{-1}=B, H_{-2}=1.

$$(H_0 * 16^0) + (H_{-1} * 16^{-1}) + (H_{-2} * 16^{-2}) = (1 * 1) + \left(\frac{B}{16}\right) + \left(\frac{1}{256}\right) = 1 + \left(\frac{11}{16}\right) + \left(\frac{1}{256}\right) = 1.6914$$

Example A1.6: Convert the base 10 number 0.6914 to a hex fraction. No need to drop the leading digit as it is already 0. Multiply by 16 to find the result 11.0624. Remove the 11=B and put it aside for later. The new result is 0.0624. Next multiply the new result by 16 to find 0.9984 which is roughly 1. Put together all the digits to find 0.B1 as the hex fractional number.

Section A1.4: References

[A1.1] https://learn.sparkfun.com/tutorials/hexadecimal/all
[A1.2] https://www.tutorialspoint.com/computer_logical_organization/hexadecimal_arithmetic.htm

Appendix 2: Calibrating a Low-Cost Frequency Counter

Many different procedures will adequately calibrate a frequency counter to the 1 Hz level although fewer suffice for the higher resolution counters. Some inexpensive frequency counters such as the Sinometer VC2000 have Oven Controlled Crystal Oscillators (OCXOs) or Temperature Controlled Crystal Oscillators (TCXOs) that provide reasonably good resolution to the level of 0.5 Hz to 1 Hz for a 10 second gate time (at 10MHz) although they definitely need to be calibrated. The OCXOs and TCXOs can be purchased as standalone modules for approximately $25 and up [c.f., Ron Schmidt Ref A2.1] and as will be seen below, some come calibrated to within about 0.5Hz and can be used to calibrate the frequency counter. Other OCXOs and TCXOs do need calibration. One suitable method for calibrating stable crystals (such as the OCXOs) and stable frequency counters to within about 0.5Hz consists of using the free National Institute of Standards and Technology NIST, WWV broadcast signal at 10MHz (or 5MHz or 2.5MHz) [A2.1]. For calibrating higher resolution equipment such as the rubidium standard, a Global Positioning System Disciplined Oscillator (GPSDO) with known output frequency can be used to calibrate to within 0.01 – 0.0005 Hz. At the time of this writing, 'used' GPSDOs can be purchased from EBay.com for about $100 as can FE5650As. Actually a second Rubidium standard with known output frequency can be used to calibrate the first rubidium standard or other high resolution equipment. Most of the FE5650As can be used right out of the box to calibrate a frequency counter to better than 0.1Hz.

This appendix first provides some reminders on the use of the 'parts-per-million (ppm)' figure of merit for calibrating the frequency counter. Then it shows how an inexpensive, single crystal frequency counter can be adequately calibrated for all frequencies. In all, the calibration was accomplished by at least four different methods including an accurate 10MHz OCXO, a Rubidium Calibration Standard, a Global Positioning System Disciplined Oscillator GPSDO, and the NIST WWV radio broadcast at 10MHz. To use the WWV calibration, we calibrated another oscillator to the WWV signal and then read the oscillator frequency on the VC2000.

Section A2.1: Accurate Frequency Source Method

The calibration of a frequency counter requires a stable frequency source accurate to at least the same number of digits as the frequency counter. The Rubidium Frequency Standard FE-5650A works well for the VC2000 frequency counter available from Amazon at the time of this writing for approximately $80-$100.

Sinometer VC2000, Amazon Stock ID: B0078MYO00

The VC2000 should be allowed to warm up the oven for at least 20minutes but an hour is better. The FE-5650A was known to operate at the default frequency F_o of

$$F_o = 8.3886080 \text{ MHz}$$

once the frequency lock occured after 3-5 minutes of powering the unit. Note that Fo it typically termed the nominal frequency when it is considered to be the expected/accurate frequency. It's only necessary to find the counter error in parts-per-million (ppm) and then scale the error to whatever frequency is being used.

The frequency standard produces an indicated frequency F_i (in units of MHz) on the frequency counter. The Error E_i (units of Hz) for the indicated frequency is then

$$E_i = F_i - F_o \qquad \text{(A2.1)}$$

The frequency-normalized error in units Hz/MHz, which is more commonly termed the part-per-million ppm error, can be written as

$$E_{fn} = \frac{E_i}{F_o} = \frac{F_i - F_o}{F_o} \qquad \text{(a.k.a, ppm) (Absolute error)} \qquad \text{(A2.2a)}$$

This last equation gives the absolute error because it's measured with respect to the accurate/nominal frequency Fo. However, it's more convenient (and more accurate) to use the error relative to the frequency counter as discussed in Example A2.1 below which is defined as

$$E_{fr} = \frac{F_o - F_i}{F_i} \qquad \text{(a.k.a, ppm) (Relative error)} \qquad \text{(A2.2b)}$$

For the frequency it's more convenient to divide by the indicated frequency F_i rather than the true frequency F_o from the FE5650A since the value of F_i appears on the frequency counter and the resulting value of E_{fr} won't be much different. For nominal frequencies on the order of 10MHz with errors on the order of 10s of Hz then either A2.2a or A2.2b can be used. The expected error E (in Hz) for the displayed frequencies F (in units of MHz) can then be found from the simple scaling relation

$$E = F\, E_{fr} \text{ or } E = F\, E_{fn} \qquad \text{or equivalently E = F * ppm} \qquad \text{(A2.2c)}$$

where F is measured in units of MHz. An example can be found in the next paragraph or two.

Prior to calibration, the VC2000 was operated for approximately 30 hours total and then remained off for at least 24 hours. The FE5650A was allowed to operate a few minutes past the lock condition which required approximately 4mins. The VC2000 was switched 'on' and the data recorded every 30 seconds. Figure A2.1 shows the ppm error vs. time (minutes) for the VC2000 warmup period starting at 18.6C ambient. As for the FE5650A, its temperature was approximately 28C when the lock condition occurred and its temperature rose to 42C when the last data point was taken at 6 minutes later.

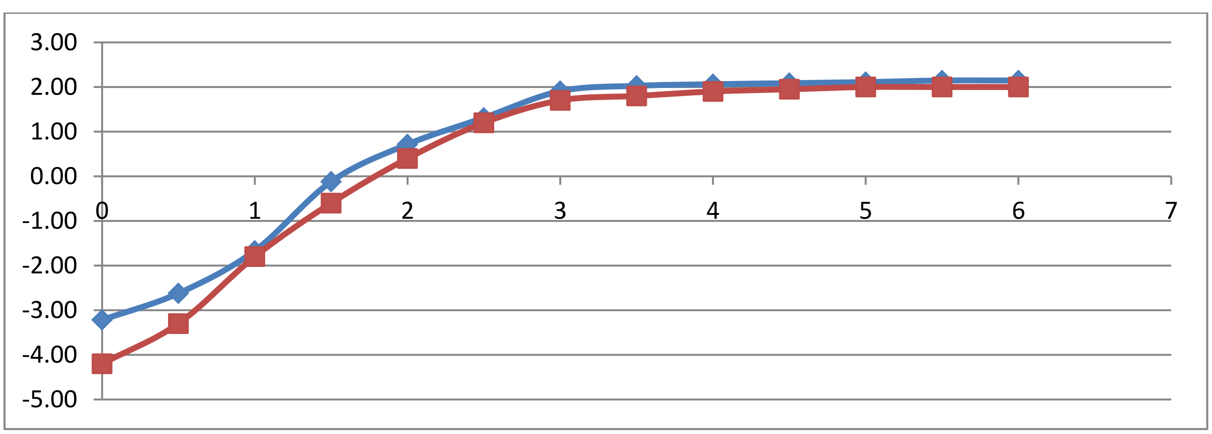


Figure A2.1: The ppm error for the VC2000 versus time in minutes during its warmup period. Top plot: After approximately 30 total hours of previous operation. The ppm leveled at 2.15. Bottom plot: After approximately 53 total hours of previous operation. The ppm leveled at 2.0.

The VC2000 should be operated more than 20 minutes prior to making a measurement – an hour is better. However, as an example, we found the displayed value on the frequency counter quit changing and showed the indicated frequency of

F_i = 8.388626 MHz

for the rubidium output frequency of

F_o = 8.388608 MHz

The normalized error (Eq. A2.2a, absolute error) is then (note the units)

$$E_{fn} = \frac{F_i - F_o}{F_o} = \frac{8388626\text{ Hz} - 8388608\text{ Hz}}{8.388608\text{ MHz}} = 2.15\ Hz/Mhz = 2.15\text{ ppm}$$

Rather than dividing by F_o, we should divide by the *indicated* frequency of 8388626 to obtain the relative fractional error E_{fr} (i.e., ppm) since it will be applied to the indicated frequency read directly on the meter for other input frequencies. Here, for 10-50 Hz errors, it makes little difference for using E_{fn} or E_{fr} – see Example A2.1 where it makes a difference as to using the absolute versus the relative ppm

To continue the example, suppose another circuit is connected to the frequency counter and it indicates a frequency of F_i = 10.000 071/72 where the 71/72 means the last two digits flip back and forth between 71 and 72. So taking an average, the indicated frequency is

F_i = 10.000 0715

What is the actual frequency? Equation A2.2b shows the expected at 10MHz is

E = 10.000,0715 MHz * 2.15 ppm = 21.5 Hz

So the actual input frequency to the frequency counter will be

F_{actual} = F_i – E = 10,000,071.5 – 21.5 = 10,000,050 MHz

The flipping last digit on the frequency counter suggests a method for better estimating the frequency of the applied signal. As before, assume the VC2000 has a gate time of 10 seconds and receives the 10MHz output signal from a GPSDO. Suppose on the one hand, the display flips from 10.000,021 to 10.000,022 MHz and back at equal rates (i.e., 10 secs at 1 and 10 secs at 2 and back etc.). The best estimate of the indicated frequency is 10.000,021,5 MHz. The extra digit of 5 appears since the average of 1 and 2 is 1.5. Suppose on the other hand, the '1' digit appears every 4 out of 6 times, while the '2' digit appears 2 out of 6 times, then the average would be (4/6)1+(2/6)2 = 1.33 and so the best estimate would be 10.0000213.

Keep in mind that the crystal frequency of the VC2000, and other similar frequency counters, do drift with time on the order of weeks or months of operation. The units require periodic calibration. The frequency counter can be slightly affected by external temperatures depending on the accuracy of the unit's temperature controller. One solution to overcome the drift and other factors would be to add simple circuitry (such as a connector) to the frequency counter that would allow the FE5650A to be substituted for the internal crystal. The programmability of the FE5650A allows the frequency to be tailored until it matches that of the original crystal in the counter. Without such a modification, one must use the ppm (2.15ppm in our case) and periodically recalibrate.

A comment should be made regarding the drift of crystal-based clocks. Figure A2.1 shows the recalibration of the VC2000 after continuously operating it for an additional 23 hours. This time the GPSDO at 10MHz was used as the frequency source. The relative error (ppm) leveled at 2.0ppm. The result was checked by setting the FE5650A to 10.0000000 MHz and the value of 2.03 ppm was obtained.

So it would appear the VC2000 drifted by about -0.12ppm. A number of other crystal effects besides aging might play a role such as slight temperature differences and possible slight hysteresis effects.

Example A2.1: A 10.000 Hz oscillator is attached to a frequency counter that reads the frequency as 5 Hz. An unknown oscillator is attached to the counter and it reads 15 Hz. What is the frequency of the unknown oscillator?
Solution: Here it makes more sense to find the error with respect to the frequency counter than the absolute error given in Equation A2.2a since we only know the frequency on the frequency counter for the 15Hz reading. The error with respect to the frequency counter would be

$$Ecounter = \frac{Fnominal - Fcount}{Fcount} = \frac{10-5}{5} = 1$$

So the actual frequency corresponding to a 15 Hz reading would be

$$Fnominal = Fcount + Fcount * Ecounter = 15 + 15 * 1 = 30Hz$$

Notice that the fractional error is being multiplied by the indicated frequency on the counter. We could not multiply by the nominal frequency since it's not known. Consequently only the relative fractional frequency works here. Had we calculated the absolute fractional error, we would have found E_{fn}=0.5. So then multiplying by the counter frequency would have produced the wrong nominal frequency of Fnom=22.5 Hz.

Section A2.2: WWV Method

Now an oldie but goodie method for calibrating 10MHz crystal-based systems to within about ½ Hz consists of listening for the beat produced by mixing the signal from the uncalibrated oscillator with the signal received from the WWV 10MHz shortwave (SW) broadcast signal transmitted by the National Institute of Standards (NIST) from Ft. Collins Colorado [A2.1-A2.4]. The objective is to discern beats produced by adding together two RF signals in the receiver. The WWV 10MHz signal can be used for calibration even though it intermittently carries Amplitude Modulation (AM) in the form of voice and tones for approximate UTC time synchronization. In the case of frequency calibration, a shortwave radio is set to 10MHz (digital radios will be quite exact but the accuracy is not really necessary) and arranged to receive both the WWV 10 MHz signal and the perhaps-not-so-accurate 10MHz from a tunable, stable oscillator to be calibrated. When the two signals have slightly different frequencies, the radio will produce a warbling/trilling sound (during transmitted tones) and then, as the two signal frequencies come closer together by tuning the uncalibrated oscillator, the frequency of the warble will decrease to a whooshing sound (during those periods without a tone and the ticks) and then stop changing when the two frequencies match (to within about ½ Hz).

Some people maintain the beats (see Chapter 9 and Appendix 3) have too low of frequency (1/2 Hz) to be heard by humans. While true, the change in intensity of the noise (hissing) produced by the radio can be heard and the calibration can be made to within about ½ Hz. In actuality, the NIST carrier frequency is much more accurate than 0.5 Hz, but reflections of the WWV signal from an ascending or descending ionosphere [A2.5] cause the frequency to Doppler shift by at most 0.5 Hz [A2.6-A2.7]; however, judicious choice of time of day lower the shift to 0.1Hz or less. Another issue concerns the strength of the received WWV signal. The frequencies below 10MHz propagate best after sundown while those above 10MHz propagate best during the day. The WWV 10MHz signal appears to be easiest to receive near sun-down and sun-up although there will be quite a range of acceptable times. The times were important for us to optimize the WWV reception as we are on the east coast of the US (New Jersey) and the NIST WWV station is located approximately 1700 miles to the west in Ft. Collins, Colorado.

We tested the WWV method of calibration on an HP33120A function generator since it shows 1 Hz resolution at 10 MHz. As a note, we did not change the internal frequency offset of the HP33120A. The calibration could then be transferred to the VC2000 if successful. For the calibration, we setup an inexpensive Radio Shack shortwave radio [A2.8] on a second floor with 'relatively' unobstructed view of the western horizon in front of a large sliding glass door. The radio was placed on a metal stool (slightly improves the signal) next to the glass door with its antenna vertically extended. Several radios can be considered for this purpose as suggested in the references [A2.9-A2.11]. The Software Defined Radio SDR listed in Ref. [A2.10] did not have as good performance as the Radio Shack unit. However, the SDR offers an exceptionally broad range of signal frequencies (100kHz – 1.5GHz) and various modulation types. In addition, the software provides a dynamically-updated frequency spectrum that makes clear the differences between sidebands, amplitude modulation and frequency modulation.

In our initial test, we calibrated a HP33120A function generator that displays to the 1 Hz level at 10 MHz although in some settings it can do better at 0.1Hz. Once calibrated, the 10MHz signal can then be applied to the VC2000 to obtain its calibration. For the HP33120A, a small dipole antenna was attached to the output - the wire length was only 4 inches and attached right at the output with one pointing up and the other pointing down. The function generator was about 6 feet from the radio. We allowed the function generator to warmup for about 15mins. The signals from the HP33120A and the WWV should have roughly the same amplitude. To adjust the signal strength, we set the HP33120A to 10% Amplitude Modulation (AM) at 1kHz and then adjusted the generator's 10MHz output voltage until the resulting radio tone roughly matched the volume of the WWA voice/tone. Then, switching off the AM, it was easy to hear a warbling sound when the frequency mismatch was large and then a slowly changing woosh sound for small frequency mismatch. We found the HP33120A to have a 5Hz error. However, after waiting another 15 mins, the error appeared to be 3 Hz. After another brief period, the offset dropped further. Later upon using the oscilloscope method for calibration we did reproduce the results but found that the HP33120A had to be operated for about one hour before the frequency would more or less stabilize at approximately 9.999,999.3 MHz. Here again, the function generator would benefit by including a connector for an external stabile oscillator such as the programmable FE5650A. Some HP33120A do have the optional circuitry for such external oscillators.

Section A2.3: Crystal Calibration

Calibrating an Oven Controlled Crystal Oscillator OCXO using the WWV method is similar to that for the function generator [A2.1] although we did not do so. A small dipole should be attached at the output – one to circuit ground and one to the output. The distance to the radio receiver needs to be adjusted so that the OCXO and WWV signals are roughly the same. Then as before, tune the OCXO with an insulated tool – the tuning port is usually accessibly on the top/side through a hole, likely under a label. In our case, the crystal was close enough to the 10.000000Mhz and so we did not break the calibration. We later found that the Monitor Products Co. (MPC) 7400B2A1 TCXOs were within about 0.5Hz of 10MHz and could have been used to determine the VC2000 calibration [A2.1].

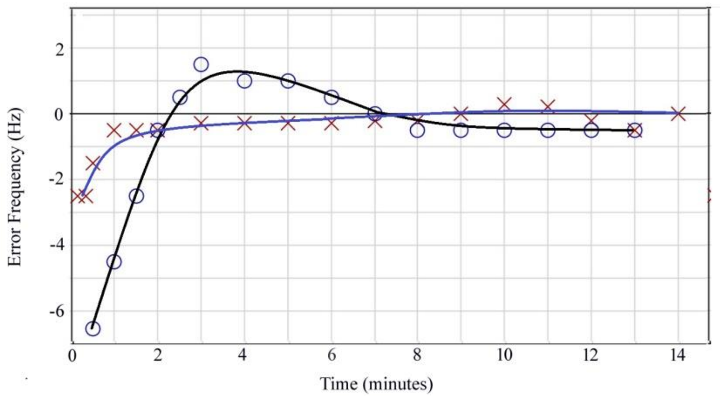


Figure A2.2: The warmup period for two different 7400B2A1 TCXOs. The graph shows the error frequency in Hz versus the time in minutes. The crystals are within about 1/2Hz right out of the box.

Section A2.4: References

[A2.1] Ron Schmidt (WA5QBA), EBay.com store ID: PBSN6040. Credit for the WWV calibration of crystals and for selling tunable TCXO. Lots of good parts at EBAY… especially TCXO.
https://www.ebay.com/str/bootspartselectroniccomponents?rt=nc&_oac=1

[A2.2] C.W. Gantt (W4CWG), "Calibration of Frequency Reference using HF WWV: Use an easy to build 10MHz WWV TRF receiver to simplify calibration." www.w4cwg.com/wwv10rx.html , Other related information: C.W. Gantt (W4CWG), "Frequency Calibration using HF WWV", wwv10mdoc.pdf, 2007. Download: http://www.w4cwg.com/wwv10mdoc.pdf Email: W4CWG@W4CWG.COM

[A2.3] A good blog with lots of calibration information and links
https://www.eevblog.com/forum/testgear/tracking-10mhz-wwv-using-a-spectrum-analyzer/

[A2.4] G.K. Nelson, M.A. Lombardi, D.T. Okayama, "NIST Time and Frequency Radio Stations: WWV, WWVH, and WWVB" NIST Special Publication 250-67 (2005). Time and Frequency Division Physics Laboratory, National Institute of Standards and Technology, 325 Broadway, Boulder, Colorado 80305. Download at
https://www.nist.gov/sites/default/files/documents/calibrations/sp250-67.pdf

[A2.5] Wikipedia
"Skywave" https://en.wikipedia.org/wiki/Skywave
"Ionosphere" https://en.wikipedia.org/wiki/Ionosphere

[A2.6] J. A. Bennett, "The Ray Theory of Doppler Frequency Shifts," Aust. J. Phys. 21, 259 (1968)
http://adsabs.harvard.edu/full/1968AuJPh..21..259B

[A2.7] Excellent plots of frequency shift versus time for various frequencies
J. Ackermann, "HF Signal Propagation and Frequency Accuracy,"
https://www.febo.com/pages/hf_stability/

[A2.8] Radio Shack "Compact Portable AM/FM Shortwave Radio" Catalog #: 2000658

[A2.9] Review of short-wave radios … see embedded links in
https://swling.com/blog/shortwave-radio-reviews/
such as http://swling.com/Radios.htm

[A2.10] Software Defined Radio guide. SDR might work for wwv. But gives great teaching on the idea of sidebands, AM, FM.
(A) C. Laufer, "The Hobbyist's Guide to the RTL-SDR: Really Cheap Software Defined Radio."
Publisher: CreateSpace Independent Publishing Platform; 1 edition (June 26, 2015)
ISBN-13: 978-1514716694. Available from Amazon.com as paperback or kindle.
(B) Related SDR kit:
RTL-SDR Blog R820T2 RTL2832U 1PPM TCXO SMA Software Defined Radio with 2x Telescopic Antennas

[A2.11] Idea of ground plane
https://www.electronics-notes.com/articles/antennas-propagation/grounding-earthing/antenna-ground-plane-theory-design.php

[A2.12] https://www.nist.gov/pml/time-and-frequency-division/nist-radio-broadcasts-frequently-asked-questions-faq

[A2.13] Monitor Products Co. Inc. Model 7400B2A1
Available from Reference [A2.1] as Ebay Item

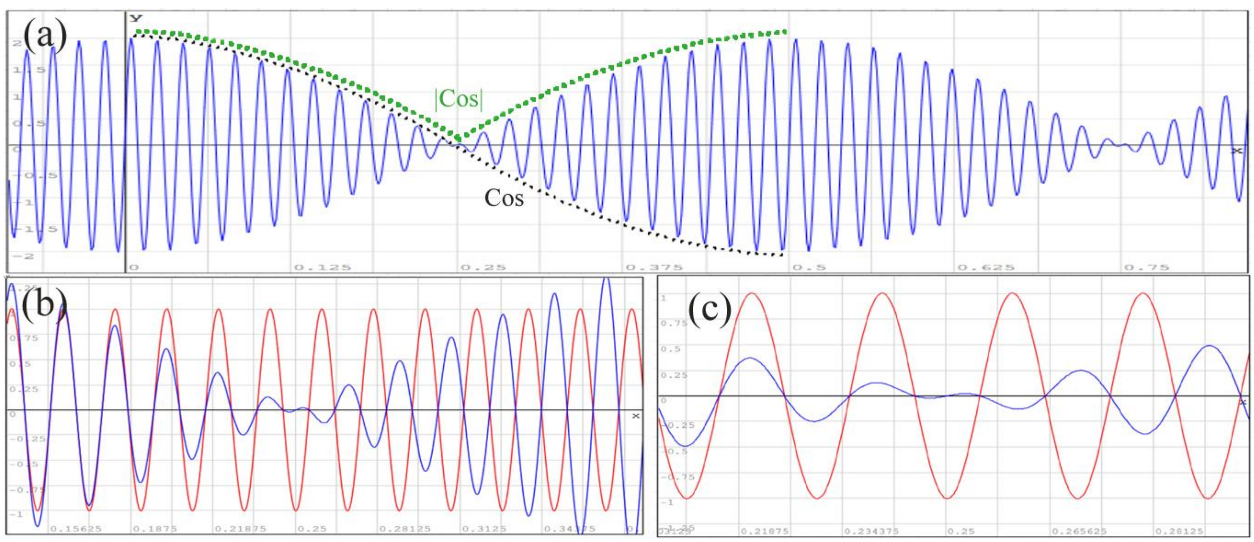
[A2.14] Modify existing equipment with OCXO
https://gerrysweeney.com/racal-dana-199x-diy-high-stability-diy-timebase-hack-for-under-25/

Appendix 3: Beats Math

One of the easiest methods to calibrate a stable oscillator against a frequency standard is to observe the beats when the two signals, with very similar frequencies, are mixed (i.e., added together) and detected. This appendix shows the math behind the beats and more importantly, derives the beat frequency and period. It is important to distinguish between the period of the beat and the period of the summed signals.

Adding together two sinusoidal waves with nearly matched frequencies will periodically produce two overlapping crests (and troughs) and the summed signal achieves a maximum (i.e., crest), and at other times, the crest and trough overlap and producing a minimum (i.e., zero). This collapse and reconstitution of superposed signals of nearly equal frequency produces beats (Figure A3.1). The beat is defined by the envelope (i.e., amplitude) of the summed sinusoidal signals as labelled by |Cos| in Figure A3.1a. For the circuits and apparatus developed in the main body of this book, the beats might not be so obvious because of the long time scale involved (many seconds between maxima). Figure A3.1a shows an example of the beats produced by superposing a signal of 50Hz with a signal of 52Hz but of equal amplitude.

Figure A3.1: (a) Beats produced by superposing a signal of 50Hz with a signal of 52Hz but of equal amplitude. Figure produced from SMath. (b) Blue curve is a magnified view of curve a near 0.25 seconds. Red curve shows the sinusoidal signal for the average frequency of 51Hz. Notice the phase offset for times larger than 0.25 seconds – the blue peaks line up with the red troughs. (c) Blue curve is the further magnified view near 0.25 seconds showing the phase reversal there.



As indicated, the beats are formed by adding together two sinusoidal signals (i.e., sinewaves) such as from a fixed-frequency calibrated source (subscript F) and a tunable source (subscript T) – the type of source is identified by F and T to match the notation used in the main chapters.

$$v_F + v_T = A\,Sin(\omega_F t) + A\,Sin(\omega_T t) \tag{A3.1}$$

where as usual, the angular frequency ω (radians/sec) relates to the Frequency F (Hz) by $\omega = 2\pi F$ and so, for the 'fixed' calibrated frequency $\omega_F = 2\pi F_F$ and the 'tunable' uncalibrated frequency $\omega_T = 2\pi F_T$.

Now, proceeding as is common for analysis of beats, define the half-sum ω_s and half-difference frequencies ω_d as

$$\omega_s = \frac{\omega_T+\omega_F}{2} \qquad\qquad \omega_d = \frac{\omega_T-\omega_F}{2} \tag{A3.2}$$

so that Equation A3.1 becomes

$$v_F + v_T = A\ Sin(\omega_s t - \omega_d t) + A\ Sin(\omega_s t + \omega_d t) \tag{A3.3}$$

Next, using the trigonometric identity for the sine of added angles, we find

$$v_F + v_T = 2A\ Cos\left(\frac{\omega_T-\omega_F}{2}t\right) Sin\left(\frac{\omega_T+\omega_F}{2}t\right) \tag{A3.4}$$

The high frequency component, the sine term, corresponds to the average of the two frequencies. Figure A3.1a shows the example for the high frequency component of 51Hz (blue curve). The cosine term in Equation A3.4, causes the beat as indicated for example in Figure A3.1a, by the black dotted curve bounding the top for times before 0.25 seconds and the bottom of the second beat. Notice the period, given by $T = 2\pi/\omega = 1/F$, for the cosine term is

$$T_{cos} = \frac{2\pi}{\left(\frac{\omega_T-\omega_F}{2}\right)} = \frac{2}{F_T-F_F} \tag{A3.5}$$

The cosine period actually extends across two beats. The dotted block curve in Figure A3.1a shows only a half period of the cosine term. The full period for the displayed 'cosine' curve for this example is

$$T_{cos} = \frac{2}{F_T - F_F} = \frac{2}{52-50} = 1$$

What went wrong? Why can't we read the beat period from Equation A3.4 especially given the the equation gives the proper form for the beat. The answer: the cosine term is not the amplitude since the amplitude should not be negative. The amplitude at any particular time should be given by the dotted green curve that bounds the upper portion of the high frequency signal. Notice the upper bounding curve has half the period of the cosine term. Said another way, the beat is defined by the bounding envelop wave. To find the actual amplitude, we will rearrange Equation A3.4 in such a way as to make the cosine term positive.

To do this, consider defining a Sign function S(x,f) where f(x) is another function. The value of the sign function is defined to be S=-1 for those values of x where f(x) is negative and S=+1 where f(x) is non-negative. So any real-valued function can be written as

$$f(x) = S(x)\ |f(x)|$$

where the vertical lines around f(x) refer to the absolute value so $|f(x)|$ is always non-negative. So for the simple case of the cosine function we would have

$$Cos(\omega t) = S(\omega t)\ |Cos(\omega t)|$$

The sign-function S is negative (i.e., -1) in those regions where the cosine is normally negative and positive (i.e., +1) in other regions.

Now apply the S function to Equation A3.4, where in this case

$$x = \frac{\omega_T - \omega_F}{2} t \qquad \text{and} \qquad S = S\left(\frac{\omega_T - \omega_F}{2} t, Cos(x)\right) \tag{A3.6a}$$

and S is negative where $Cos\left(\frac{\omega_T - \omega_F}{2} t\right)$ is negative. Grouping S with Sin, we find

$$v_F + v_T = 2A \left|Cos\left(\frac{\omega_T - \omega_F}{2} t\right)\right| \left\{ S\, Sin\left(\frac{\omega_T + \omega_F}{2} t\right) \right\} \tag{A3.6b}$$

The term $\left|Cos\left(\frac{\omega_T - \omega_F}{2} t\right)\right|$ is the amplitude as shown by the dotted green line bounding the top of the signal in Figure A3.1a and it has half the period of the original cosine term. The beat period is therefore

$$T_{beat} = \frac{1}{F_T - F_F} \tag{A3.6c}$$

The S function in "$S\, Sin\left(\frac{\omega_T + \omega_F}{2} t\right)$" of Equation A3.6a, flips the superposed sinewave upside down at times but does not affect the envelope. To better see this, consider Figure A3.1c, where the constant-amplitude sinusoidal curve in red is the term $Sin\left(\frac{\omega_T + \omega_F}{2} t\right)$ which has its frequency as the average of 51Hz for the two frequencies (50Hz, 52Hz). To the left of the first null at 0.25 seconds, the blue sinusoidal wave defining the beat is in phase with the signal for the average frequency. After the null point, the two are ½ cycle out of phase or in other words, one entails a negative sign compared to the other. This reversal happens because of the S function in Equation A3.6a. For reference, Figure A3.1c shows a magnified view of the region near the null between the first two beats where the reversal occurs.

The issue regarding the amplitude is on the verge of subtle and has very important consequence for calibrating the tunable source. The frequency difference between the tunable and fixed source is related to the beat period T_{beat} according to Equation A3.6b

$$|F_T - F_F| = 1/T_{beat} \tag{A3.7}$$

where the period of the beat T_{beat} is measured in seconds. In particular, the beat period is not double the value that a person might think based on Equations A3.4 and A3.5.

As a note, occasionally Equation A3.6a might be written as

$$v_F + v_T = A\sqrt{2[1 + Cos(\Delta\omega\, t)]}\, Sin(\omega_F t + \varphi)$$

where $\Delta\omega = \omega_T - \omega_F$ and where *the square root is always positive* and comes from the trigonometric identity of

$$Cos(\Delta\omega\, t) = Cos\left(\frac{\Delta\omega\, t}{2} + \frac{\Delta\omega\, t}{2}\right) = 2\, Cos^2\left(\frac{\Delta\omega\, t}{2}\right) - 1$$

The $Sin(\omega_F t + \varphi)$ term makes use of the fact that for nearly equal frequencies at high frequency, $\omega_T \sim \omega_F$ so that $\frac{\omega_T + \omega_F}{2} \sim \omega_F$. Also the phase slip is missing without proper use of the sign function.

Appendix 4: Regression and SMATH

Linear Regression finds the 'best' line to pass through a collection of data points. The 'best' line minimizes the 'distance' between itself and the data points. The collection of data points most often comes from experimental data where a change in an independent parameter (often denoted as x) causes a change in a dependent parameter (often denoted as y). Usually there is some question as to how well correlated are the two variables x and y. Noise or some degree of randomness in the y and x variables will cause the data points to scatter about a well-defined line. In some cases, the fit parameter for regression is used to measure the amount of randomness and whether y and x have any relation to one another at all. One example would be the relation between atmospheric pressure (dependent variable) and air temperature (independent variable). Another example would be the FE5650A output frequency as the dependent variable and the Fcode as the dependent one.

Section A4.1 briefly describes the idea for linear regression as minimizing the error between a line and a set of data points and states, without proof, the procedure for finding the best fit line. Section A4.2 then provides a couple of examples for calculating the regression by hand. The first example considers two data points and shows the regression line is the same as the line exactly passing through the two points. The second example finds by linear regression, the line that best fits three data points. Section A4.3 shows a relatively easy method of deriving the equations for linear regression as well as an extension for polynomial regression. The results can easily be incorporated into software or firmware. As the main drawback for regression of an nth order polynomial, the numerical value of the terms exceed the capacity of the variables. Section A4.4 shows a typical regression procedure for the SMath mathematical software package (available free).

Sometimes for regression, the data points might be assigned a number describing the certainty with which the point represents an actual or outlier data point. For example, imagine ten people spread along a road measure the time and position of a speeding car. Suppose when the fifth person makes a measurement, a flash of light momentarily blinds the person just as the car speeds past. The person might say the measurement is only 50% sure. Weighted regression provides a method of including the uncertainty of the data point. The appendix briefly mentions the weighted regression.

There are many math packages for PCs and smart phones with Linear Regression ranging in price from free to thousands of dollars. SMath is an excellent free package similar to the high priced MathCad. The SMath software by Andrey Ivashov can be downloaded at

http://en.smath.info/view/SMathStudio/summary.

Please consider making a contribution to help support the extensive work for this package. It is also possible to write excel spreadsheets for similar purposes or use your favorite programming language (even using a microcontroller).

Section A4.1: Methods of Unweighted Linear Regression Analysis

The simplest regression analysis consists of fitting a straight line to a collection of data points. The data can be represented as a data set

$$DS = \{(x_1, y_1), (x_2, y_2), (x_3, y_3) \dots (x_N, y_N)\} \tag{A4.1}$$

which assumes N points of data have been collected. Assume that (x_i,y_i) represents one of those points where 'i' can take an integer value from 1 to N. Often, it is sufficient to plot the data on a graph and draw a straight line that passes as close as possible to all of the data points. The mathematical method of fitting, namely linear regression, provides equations for the best straight line of the form

$$y = mx + b \tag{A4.2}$$

to fit the data points where m is the slope and b is the y-intercept (i.e., y=b at x=0). The 'best straight line' is one that comes as close as possible to all the data points. Such a best line requires one to define the distance from the data point to the line and then determine m and b so as to minimize the distance to all the points in the data set. All of the points in the set contribute equally to the values of m and b.

The most important step is to define the distance between the data points and an arbitrary straight line. In this case, the distance is defined along the vertical direction as

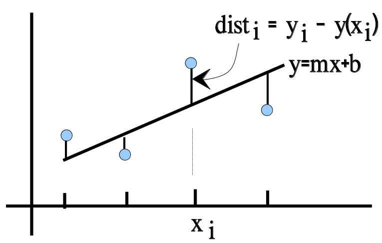


Figure A4.1: The distance between a data point and a straight line.

$$distance_i = y_i - y(x_i) \tag{A4.3a}$$

Here, y_i is the height for the data point (x_i,y_i), and $y(x_i)$ is the height of the line corresponding to the same x-value, namely x_i, as shown in Figure A4.1. Actually the distance so defined is also known as a residual. Now, if we sum all the distance terms to produce the total distance= $\sum distance_i$, the correct line would yield a total distance of zero since half the points would be on one side of the line and the other half on the other side of the line.

Rather than directly using Equation A4.3a, one defines the 'total distance' as a positive number which consists of the sum of squares of the terms in Equation A4.3a. Let's call the sum an 'error' as opposed to total distance to distinguish it from the sum of the linear distance.

$$Error = \sum_{i=1}^{N}[y_i - y(x_i)]^2 = \sum_{i=1}^{N}[y_i - mx_i - b]^2 \tag{A4.3b}$$

In this case, we want to minimize the Error term which is equivalent to minimizing the distance between the line and the data points. In other words, the least squares approximation means to minimize the sum of squared residuals in Equation A4.3b.

The ensuing section will show how minimizing the 'Error' in Equation A4.3b provides the values for the slope m and intercept b according to

$$m = \sigma_{xy}^2/\sigma_x^2 \tag{A4.4}$$

$$b = \bar{y} - m\bar{x} \tag{A4.5}$$

where we need to define the quantities in these two equations. The average value of x, namely $\bar{x}$, and the average value of y, namely $\bar{y}$, are obtained by adding up all the values of x_i and y_i, respectively, and dividing by the total number of values N.

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{A4.6a}$$

$$\bar{y} = \frac{\sum_{i=1}^{N} y_i}{N} \tag{A4.6b}$$

It is now possible to determine the variance of x

$$\sigma_x^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N} \tag{A4.6c}$$

which simply requires subtracting the average $\bar{x}$ from each value x_i then squaring that difference, then adding them all together and then dividing by the number of data points N. Similarly, the covariance can be determined as

$$\sigma_{xy}^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})}{N} \tag{A4.6d}$$

The sums in Equations A4.6 can either be computed by hand or by use of a math package.

The procedure then for finding the regression fit to data consists of the following steps in the following order.

1. Identify the number N of data points in the data set
2. Calculate $\bar{x}$ and $\bar{y}$ in Equations A4.6a,b since these occur in all of the other expressions.
3. Calculate σ_x^2 and σ_{xy}^2 in Equations A4.6c,d
4. Calculate m and b in Equations A4.4,5.
5. Write m and b in y=mx+b.

Example A4.1: Use linear regression to find the best straight line approximation to the following two points: (1,4) and (2,5). The data set is DS={ (1,4), (2,5) }

Finding the best fit line does not require the linear regression in this case. The line passing through the two points can be computed using the points $x_1=1$, $y_1=4$ and $x_2=2$, $y_2=5$. Recall, the equation for the straight line is y=mx+b where the slope is $m=(y_2-y_1)/(x_2-x_1)=1$ and the y-intercept is $b = y_1 - m\,x_1 = 3$. More than two points generally requires the linear regression to come up with the best fit line.

Now consider linear regression for this simple example and see that it reproduces the expected line of y=x+3.

(i) Identify the number N of data points N in the data set

The number of data points is N=2

(ii) Calculate $\bar{x}$ and $\bar{y}$ in Equations A4.6a,b

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1+x_2}{2} = \frac{1+2}{2} = 1.5 \qquad \bar{y} = \frac{\sum_{i=1}^{N} y_i}{N} = \frac{y_1+y_2}{2} = \frac{4+5}{2} = 4.5$$

(iii) Calculate σ_x^2 and σ_{xy}^2 in A4.6c,d

$$\sigma_x^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N} = \frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2}{N} = \frac{(1-1.5)^2 + (2-1.5)^2}{2} = 0.25$$

$$\sigma_{xy}^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N} = \frac{(x_1 - \bar{x})(y_1 - \bar{y}) + (x_2 - \bar{x})(y_2 - \bar{y})}{2}$$
$$= \frac{(1 - 1.5)(4 - 4.5) + (2 - 1.5)(5 - 4.5)}{2} = 0.25$$

(iv) Calculate m and b in Equations A4.4,5

$$m = \frac{\sigma_{xy}^2}{\sigma_x^2} = \frac{0.25}{0.25} = 1$$

$$b = \bar{y} - m\bar{x} = 4.5 - 1 \cdot 1.5 = 3$$

(v) Finally, y=mx+b becomes y=x+3 as expected.

Example A4.2: Use linear regression to find the best straight line approximation to the following three points: (1,4) and (2,4) and (4,12). The data set is DS={ (1,4), (2,4), (4,12) }

Following the linear regression steps provides the following.

(i) The number of data points is N=3

(ii) Calculate $\bar{x}$ and $\bar{y}$ in Equations A4.6a,b

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{1+2+4}{3} = 2.33 \qquad \bar{y} = \frac{\sum_{i=1}^{N} y_i}{N} = \frac{4+4+12}{3} = 6.67$$

(iii) Calculate σ_x^2 and σ_{xy}^2 in Equations A4.6c,d

$$\sigma_x^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N} = \frac{(1-2.33)^2 + (2-2.33)^2 + (4-2.33)^2}{3} = 1.54$$

$$\sigma_{xy}^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N} = \frac{(1 - 2.33)(4 - 6.67) + (2 - 2.33)(4 - 6.67) + (4 - 2.33)(12 - 6.67)}{3} = 4.44$$

(iv) Calculate m and b in A4.4,5

$$m = \frac{\sigma_{xy}^2}{\sigma_x^2} = \frac{4.44}{1.54} = 2.88$$
$$b = \bar{y} - m\bar{x} = 6.67 - 2.88 \cdot 2.33 = -0.040$$

(v) Finally, y=mx+b becomes y = 2.88 x - 0.04.

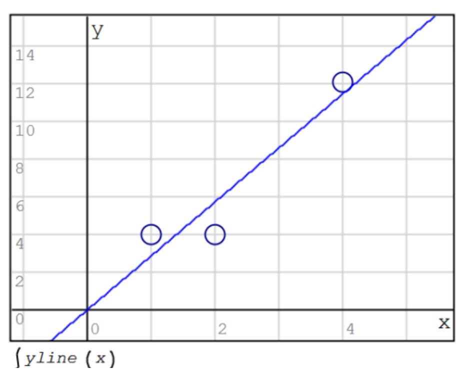


Figure A4.2: Data set and regression line for Example A4.2.

The SMath [www.smath.com] software can easily compute the linear regression for large numbers of points. Section A4.3 shows the SMath page and provides a listing of the corresponding SMath file. It's far simpler to learn to add equations to the SMath page than copy the listing (unless it can be copied and pasted).

Section A4.2: Derivation for Polynomial Regression

Linear regression is a special case of polynomial regression which seeks the best fit of a polynomial

$$y = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 \tag{A4.7}$$

to a collection of data points $DS = \{(x_1, y_1), (x_2, y_2), (x_3, y_3) \ldots (x_N, y_N)\}$. The constants a_n through a_0 must be determined to make the best possible fit. In this sense, the a_n through a_1 should be considered as variables. As before, assume (x_i, y_i) represents one of the points where 'i' can take an integer value from 1 to N. The procedure for the polynomial specializes to the linear case by using only the last two terms in Equation A4.7 in which case, a_1 is the slope m and a_0 is the intercept b.

Consider the case of the quadratic polynomial since it will be easy to generalize or specialize

$$y = a_2 x^2 + a_1 x^1 + a_0 \tag{A4.8}$$

The regression equations can be found by minimizing the total error

$$Error = \textstyle\sum_{i=1}^{N} [y(x_i) - y_i]^2 = \sum_{i=1}^{N} \left[a_2 x_i^2 + a_1 x_i^1 + a_0 - y_i\right]^2 \tag{A4.8}$$

between the polynomial y(x) the set of data points. The residual at the point x_i is the distance between the point $y(x_i)$ on the curve and the value y_i at x_i. Notice that the Error depends on a_2, a_1 and a_0 since all the x_i and y_i have known values from the data set. The minimum Error can be found by setting the derivatives with respect to the a_m to zero and assuming variations in a_m are independent of variations in a_n when $m \neq n$; that is,

$$0 = \frac{\partial Error}{\partial a_n} \qquad \text{and} \qquad \frac{\partial a_m}{\partial a_n} = \delta_{mn} = \begin{Bmatrix} 1 & for\ m = n \\ 0 & for\ m \neq n \end{Bmatrix} \tag{A4.9}$$

The partial derivatives of the Error produce the following results

$$0 = \frac{\partial Error}{\partial a_0} = 2 \textstyle\sum_{i=1}^{N} \left[a_2 x_i^2 + a_1 x_i^1 + a_0 - y_i\right] \tag{A4.10a}$$

$$0 = \frac{\partial Error}{\partial a_1} = 2 \textstyle\sum_{i=1}^{N} \left[a_2 x_i^3 + a_1 x_i^2 + a_0 x_i - x_i y_i\right] \tag{A4.10b}$$

$$0 = \frac{\partial Error}{\partial a_1} = 2 \textstyle\sum_{i=1}^{N} \left[a_2 x_i^4 + a_1 x_i^3 + a_0 x_i^2 - x_i^2 y_i\right] \tag{A4.10c}$$

Next for convenience, change the notation of the Average as 'a bar over the quantity' to the 'quantity in braces' such as for example $\bar{x} = < x >$; this new notation makes it easier to include the subscripts and powers under the averaging symbol. Notice that all of the terms can be rewritten as, for example, the following two

$$\textstyle\sum_{i=1}^{N} \left[a_2 x_i^3\right] = a_2 \sum_{i=1}^{N} x_i^3 = N a_2 \frac{\sum_{i=1}^{N} x_i^3}{N} = N a_2 < x^3 > \qquad \sum_{i=1}^{N} [a_0] = N a_0$$

Notice the averaging symbols mean to average the points in the data set. Equations A4.10 become, after dividing by 2N,

$$a_2 < x^2 > +a_1 < x^1 > +a_0 =< y > \tag{A4.11a}$$

$$a_2 < x^3 > +a_1 < x^2 > +a_0 < x >=< x\,y > \tag{A4.11b}$$

$$a_2 < x^4 > +a_1 < x^3 > +a_0 < x^2 >=< x^2 y > \tag{A4.11c}$$

So now, using matrices to solve for the coefficients a_o, a_1, a_2 provides

$$\begin{bmatrix} 1 & < x > & < x^2 > \\ < x > & < x^2 > & < x^3 > \\ < x^2 > & < x^3 > & < x^4 > \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} < y > \\ < \mathrm{x\,y} > \\ < x^2 y > \end{bmatrix} \tag{A4.12}$$

Computers can easily solve for the coefficients. Notice how the number of rows and columns match the number of coefficients to be determined. Also notice the matrix on the left is square and the power of x increases by one from left to right and up to down. Similarly, on the right hand side, the power of x increases by one from up to down. These facts make it easy to generalize the matrix for higher order polynomial regression. For example, regression to a 3rd order polynomial provides

$$\begin{bmatrix} 1 & < x > & < x^2 > & < x^3 > \\ < x > & < x^2 > & < x^3 > & < x^4 > \\ < x^2 > & < x^3 > & < x^4 > & < x^5 > \\ < x^3 > & < x^4 > & < x^5 > & < x^6 > \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} < y > \\ < \mathrm{x\,y} > \\ < x^2 y > \\ < x^3 y > \end{bmatrix} \tag{A4.13}$$

For the linear regression, the coefficient a_2 and the matrix entries related to it in Equation A4.12 need to be removed. We then have, using the slope as $m=a_1$ and the intercept as $b=a_0$.

$$\begin{bmatrix} 1 & < x > \\ < x > & < x^2 > \end{bmatrix} \begin{bmatrix} b \\ m \end{bmatrix} = \begin{bmatrix} < y > \\ < \mathrm{x\,y} > \end{bmatrix} \tag{A4.14}$$

Again, one can solve this for b and m using a matrix calculator. Equation A4.14 can be solved by hand to find

$$\begin{bmatrix} b \\ m \end{bmatrix} = \begin{bmatrix} \frac{<x^2 y><y>-<x\,y><x>}{\sigma_x^2} \\ \frac{<x\,y>-<x><y>}{\sigma_x^2} \end{bmatrix} = \begin{bmatrix} \bar{y} - m\bar{x} \\ \frac{\sigma_{xy}^2}{\sigma_x^2} \end{bmatrix} \tag{A4.15}$$

as was found in the previous sections.

Section A4.3: SMath Page

The SMath software package has the smorgasbord of mathematical function that display in a graphical interface – no programming required (although it's possible) beyond essentially dragging and dropping the mathematical symbols and operators. Figure A4.3 below shows the SMath page for calculating the linear regression. Several websites have excellent tutorials:
https://smath.com/wiki/MainPage.ashx
https://smath.com/wiki/GetFile.aspx?File=Tutorials/SMathPrimer.pdf

Figure A4.3: SMath page for linear Regression

Experimental Values Y=vertical, X=Horizontal

$$X := \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} \qquad Y := \begin{bmatrix} 4 \\ 4 \\ 12 \end{bmatrix}$$

Equations

$$N := \text{length}(X)$$

$$Xave := \frac{1}{N} \cdot \sum_{i=1}^{N} X_i \qquad Yave := \frac{1}{N} \cdot \sum_{i=1}^{N} Y_i$$

$$VarX := \frac{\left(\sum_{i=1}^{N} \left(X_i - Xave\right)^2\right)}{N} \qquad VarY := \frac{\left(\sum_{i=1}^{N} \left(Y_i - Yave\right)^2\right)}{N}$$

$$CovXY := \frac{\sum_{i=1}^{N} \left(X_i - Xave\right) \cdot \left(Y_i - Yave\right)}{N}$$

$$Fit := \frac{CovXY^2}{VarX \cdot VarY}$$

$N = 3$

$Xave = 2.3333$

$Yave = 6.6667$

$VarX = 1.5556$

$VarY = 14.2222$

$CovXY = 4.4444$

Results

$$slope := \frac{CovXY}{VarX} \qquad intercept := Yave - slope \cdot Xave$$

$slope = 2.8571$ $\quad intercept = 0$ $\quad Fit = 0.8928571429$

Graph

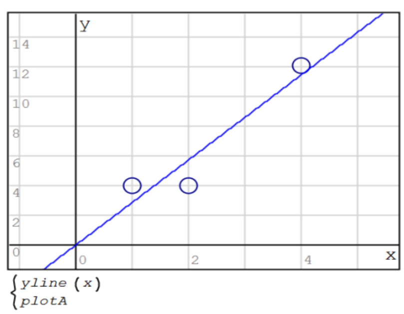
The program vert line is called 'line' in programming palette

```
plotG(x, y, char, size, color) := | n := length(x)
                                  | plot := augment(x_1, y_1, char, size, color)
                                  | for i ∈ [2..n]
                                  |    plot := stack(plot, augment(x_i, y_i, char, size, color))
                                  | plot
```

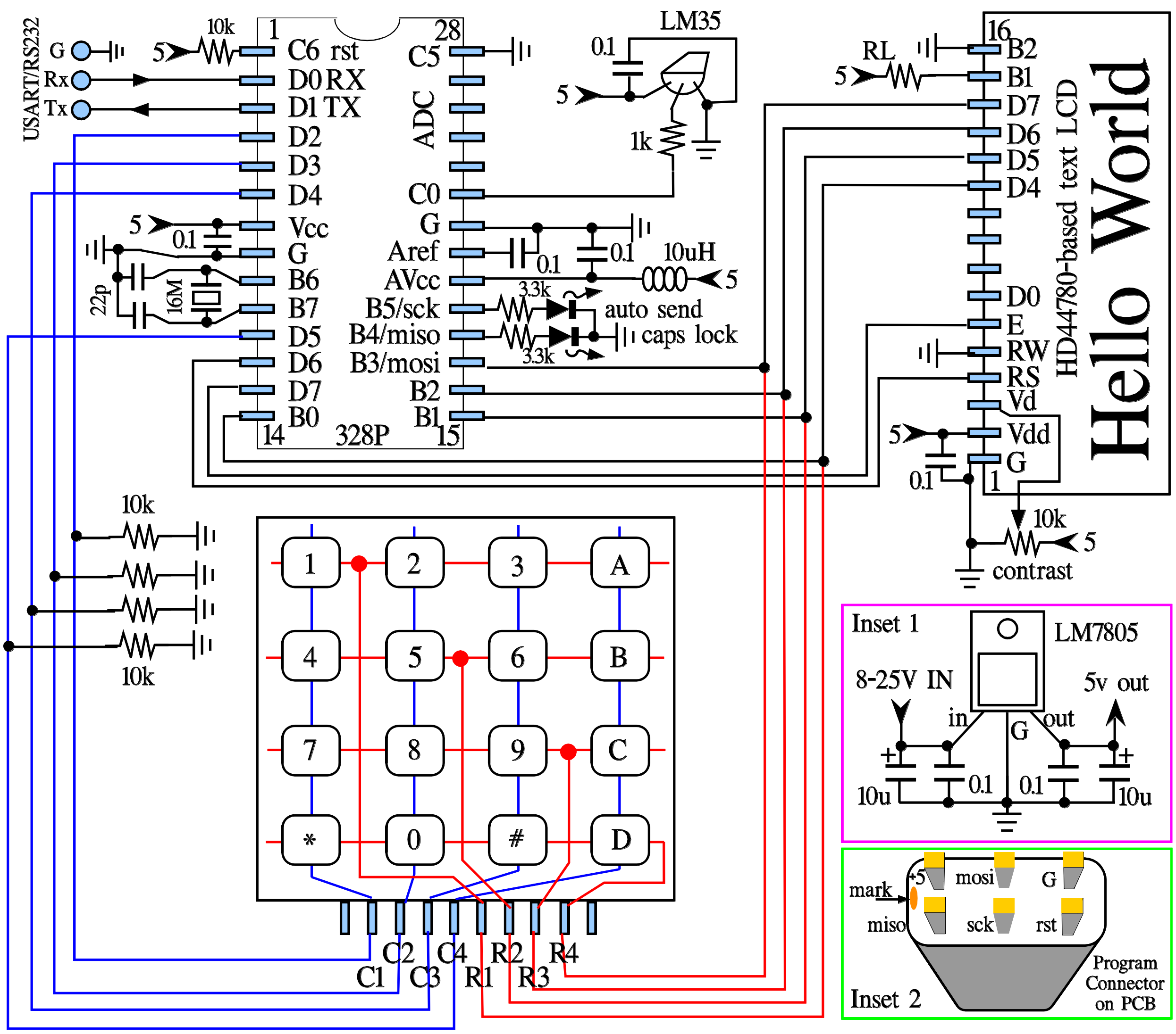
```
plotA := plotG(X, Y, "o", 20, "dark blue")
```

Char must be o,*,+,dec pnt

$$yline(x) := slope \cdot x + intercept$$



Appendix 5: Microcontroller Circuit Diagram



Appendix 6: Source Code for the FE5650A Controller

The following appendix sections provide the Source Code for main.cpp and the user libraries. Refer to Chapter 10 for information on its use and transfer to the ATMEGA328P.

Section 6.1: main.cpp

```
//    Copyright 2020 Angstrom Logic except for the myF64 Library.                                                  //
//    You may use 'myF64' Library free of charge for any purpose you wish and the remainder of the program and      //
//    libraries for personal use provided you agree to the following                                                //
//    License:                                                                                                      //
//    Warranty of Provenance and Disclaimer of Warranty. Licensors warrants that the copyright                      //
//    in and to the Original Work and the patent rights granted herein by Licensor(s) are owned by the              //
//    Licensors or are sublicensed to You under the terms of this License with the permission of the                //
//    contributor(s) of those copyrights and patent rights. Except as expressly stated in the immediately           //
//    preceding sentences, the Original Works are provided under this License on an "AS IS" BASIS and WITHOUT       //
//    WARRANTY, either express or implied, including, without limitation, the warranties of                         //
//    non-infringement, merchantability or fitness for a particular purpose. THE ENTIRE RISK AS TO THE              //
//    QUALITY OF THE ORIGINAL WORK IS WITH YOU. This DISCLAIMER OF WARRANTY constitutes an essential part           //
//    of this License. No license to the Original Works is granted by this License except under this                //
//    disclaimer.     All of this license must be included in the source code including the following paragraph.   //
//                                                                                                                  //
//    Limitation of Liability. Under no circumstances and under no legal theory, whether in tort                    //
//    (including negligence), contract, or otherwise, shall the Licensor be liable to anyone for any                //
//    indirect, special, incidental, or consequential damages of any character arising as a result of              //
//    this License or the use of the Original Work including, without limitation, damages for loss of              //
//    goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages             //
//    or losses.                                                                                                    //

#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include <string.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <avr/eeprom.h>

#include "myF64.h"
#include "LCD16x2_ST7032.h"
#include "KeyPad4x4.h"
#include "USART.h"
#include "TC16.h"
#include "StrNum.h"
#include "ADC328.h"

//see modules for their related #define
#define USART_TIMEOUT 500                         // 500mS to rcv next char
#define TEMP_TIME 4000                            // 4000mS between each ADC read
#define KP 1.00000000376                          //const of prop between Rtrue and Rstored

//eeprom addresses:
#define ADDR_ALL 0
#define ADDR_RSTORE 10
#define ADDR_FSTORE 30
#define ADDR_KP 50
```

```
#define ADDR_RTRUE 70
#define ADDR_FCODE 90
#define ADDR_FOUT 110
#define ADDR_BAUD 130

#define RESULTLEN 22                                    //number of elements in the temporary storage result[]
char result[RESULTLEN] = {0};

void Parms_init(void);
void delay_ms(int ms);                                  // delays specified milliseconds

// FE5650A only accepts character strings
// Initial values for first run
char Rstored[18] = {"50255054.934100"};                 // 15 digits + 1 for 0 terminator
char Fstored[10] = {"2ABB5050"};                        //8digits + 1 for 0 terminator
char Kp[16] = {"1.00000000376"};                        //13digits + 1 for 0 terminator
char Rtrue[18] = {"50255055.118773"};                   //for unit 1: 15digits + 1 for 0 terminator
char Fcode[10] = {"2ABB5050"};
char Fout[18] = {"8388608.001"};                        //output freq ... max 8 int digits, 3 fract digits = 12 + 1

// ======== MENU Relatedvoid menuBaud(void);
void menuStatus(void);
void menuRtrue(void);
void menuGo(void);
void menuTemp(void);

// ======== auxiliary functions related to moving through frequency strings
uint8_t FcodeIndexFromFoutIndex(uint8_t iFo);
uint8_t FoutIndexFromFcodeIndex(uint8_t iFc);
uint8_t cursorPosFromFcFoIndex(uint8_t ArryIndex);
uint8_t FcIndexMoveRight(uint8_t iFc );
uint8_t FcIndexMoveLeft(uint8_t iFc);
uint8_t FoIndexMoveRight(uint8_t iFo );
uint8_t FoIndexMoveLeft(uint8_t iFo);

// ======== auxiliary function related to frequency calculations
char* freqOutCalc(char* RtrueStr, char* FcodeStr);
char* FcodeCalc(char* FoutStr, char* RtrueStr);
char* FreqStrConditioner(char* existStr, bool Flag_T2AddZeros_F2RemoveZeros, char* result);
char* FcodeStrConditioner(char* existStr, bool Flag_T2AddZeros_F2RemoveZeros, char* result);
char* getNumberDsply(uint8_t Line, uint8_t Pos, char* result);
void sendFreq(const char* Fcode2);

// ======= ADC misc.
void ADC_calledFunction(uint16_t adcResult);            // ADC ISR calls this function
uint8_t mux = 0;                                        // ADC channel used for temp

// ======= TC16 misc.
void TC16_calledFunction(void);                         // TC16 ISR calls this function
uint16_t baud = 9950;                                   // initial baud rate

// ==== LCD line number and position controlled in GO menu for frequency changes
uint8_t iLine = 1;                                      // needed by GoMenu and helper functions
uint8_t iPos = 5;                                       // <<=== always updated

void uC_Init(void);

int main(void)
```

```
{   // ======= Initialize various uC modules
    uC_Init();
    keyPad_init();
    TC16_config(USART_TIMEOUT, F_CPU, false, true, true, 0); //500mSec timing period, fcpu=16MHz
    USART_config(F_CPU, baud);
    LCD_init();
    ADC_Config(Intrnl,TC1CompMatB, Div128, 0, true, true, true, true);
    delay_ms(200);

    // ======= Main Menu ========

    char keyMM = 0;
    while(1)
    {   L_clear();
        L_cursorLinePos(0,0); L_writeStr("1:Bau 2:S&K 3:R");        // Bau=Baud, S&K= Status enquiry and Kp, R=ref freq
        L_cursorLinePos(1,0); L_writeStr("4:Opt 5:Tmp 6:Go");      // Opt=nothing, Tmp shows degree C, Go: freq settings

        keyMM = keyGet();
        switch(keyMM)
        {   case '1': menuBaud(); break;
            case '2': menuStatus(); break;
            case '3': menuRtrue(); while(keyPad()==0); break;
            case '4': break;
            case '5': menuTemp(); break;                           // shows module temperature
            case '6': menuGo(); break;                             // runs frequency control part of program
            default: break;  }
    }
}

void TC16_calledFunction(void)
{ }

void ADC_calledFunction(uint16_t adcres)                           //ADC  ISR calls this function
{   L_cursorLinePos(0,13);
    L_writeStr( uint32toa( (uint32_t)(adcres/10), result, 10 ) );  // div10 since adc=530 is 53C
    L_writeStr("\5"); //degrees symbol
    L_cursorLinePos(iLine,iPos);}                                  //iLine iPos keeps track of cursor in GoMenu

void sendFreq(const char* Fcode2)                                  // used by goMenu to send Fcode to FE5650A
{   USART_SendStr( "F=" );
    USART_SendStr(Fcode2);
    USART_SendChar(0x0D);  }


// ===================================================
// ==================== MENUS =======================
// ===================================================

// stops, change baud, restarts comm
void menuBaud(void)
{
    char key = 0;
    while(key!='E')                                                  // E = escape code
    {   L_clear();                                                   // clear LCD
        L_writeStr("Baud: ");
        L_writeStr(uint32toa(baud,result,10));                       // show current baud
        L_cursorLinePos(1,0);                                      // set cursor to line 1, position 0
        L_writeStr("1:New 2:Sv 3:Rcl");                            // New Baud, Save, Recall
```

```
        key=keyGet();                                   //wait for key press
        switch (key)
        { case '1':
            L_erase(1,0,15);                            // erase LCD Line 1 positions 0-15
            L_cursorLinePos(1,0);                       // cursor position 0, line 1
            L_cursorVisBlink(true,true);                // show cursor and make it blink
            getNumberDsply(1,0,result);                 // get a typed number starting at line=1 position=0
            L_cursorVisBlink(false,false);              // invisible cursor, no blink
            if ( result[0] != 0 ) baud = (uint16_t)atol(result);   // set variable 'baud'
            break;

         case '2': //Save
            eeprom_update_word( (uint16_t*)ADDR_BAUD, baud );      // save to eeprom
            break;

         case '3':
            baud = eeprom_read_word( (uint16_t*)ADDR_BAUD );       // recall from eeprom
            break;

         default: key = 'E'; break;  }                             // end switch
         USART_config(F_CPU,baud);
    }
}

//Send Status inquiry to FE5650A and Enter Kp
void menuStatus(void)
{
    char c = 0;                                                    //used to save to Fstored
    char key = 0;
    while(key != 'E')
    {   L_clear();                                                 // clear LCD
        L_writeStr( "1:S? 2:S\7 3:S\6" );                          // Status, status save, status rcl
        L_cursorLinePos(1,0);                                      // cursor at line 1 and pos 0
        L_writeStr( "4:K  5:K\7 6:K\6" );                          // Kp, kp save, kp rcl

        key = keyGet();                                            // wait for key press
        L_clear();                                                 // clear LCD

        switch (key)
        { case '1':                                                // get Rstored and Fstored from FE5650A

            USART_ClearBuffer();                                   // clear buffer for rvd chars
            USART_SendChar('S'); USART_SendChar(0x0D);             // sends S<cr>

            TC16_Start();                                          // start timeout timer
            while( !TC16_isTimeExpired() ) ;                       // wait for time expired

            if( USART_isBuffEmpty() ) {L_writeStr("No Comm.");  keyGet(); break; }      //FE5650A not connected
            if ( !USART_bufchr('R') ) { L_writeStr("Xmt Err/Encrypt"); keyGet(); break;}//wrong baud
            if ( !USART_bufchr('F') ) { L_writeStr("Xmt Err/Encrypt"); keyGet(); break;} //wrong baud

            while ( USART_ReadBuffChar() !='R' && !USART_isBuffEmpty() );      // note: repeatedly reads a char from buffer
            L_writeChar('R');                                      // print R to LCD
            USART_ReadBuffChar();                                  // eliminate '='

            //place Rstored in global variable and print on LCD
            for(int i = 0; (i<15) && !USART_isBuffEmpty(); i++)              // reads 15 chars or until buffer empty
```

```
                { Rstored[i] = USART_ReadBuffChar();L_writeChar(Rstored[i]); }  // 0 already in last element
        while ( !USART_isBuffEmpty() && USART_ReadBuffChar() != '=' ) ;      // move to string after 'F=' (HEX code)
        L_cursorLinePos(1,0);                                                // move cursor to line 1, position 0

        //extract 8 digits for the stored Fcode
        for(int i = 0; (i<16) && !USART_isBuffEmpty(); i++)         // read 16 hex chars from FE
                { L_writeChar( c = USART_ReadBuffChar() );          // set c to F digit and write on LCD
                  if(i<8) Fstored[i]=c; }                           // Fstored only uses first 8 digits

        while(keyGet()==0);                                         // wait for key
        break;

    case '2':                                                       // Save Rstored and Fstored to eeprom
        L_writeStr(Rstored);
        L_cursorLinePos(1,0);                                       // cursor moves to line 1 position 0
        L_writeStr(Fstored); L_writeStr(" 0:N 1:Y");
        if (keyGet() == '1')                                        // save Rstore and Fcodestored
                { eeprom_update_block( (const void*)Rstored, (void*)ADDR_RSTORE, 16);
                  eeprom_update_block( (const void*)Fstored, (void*)ADDR_FSTORE, 9);  }
        break;

    case '3':                                                       // Rstored and Fstored from eeprom
        eeprom_read_block( (void*)Rstored, (const void*)ADDR_RSTORE, 16 );
        eeprom_read_block( (void*)Fstored, (const void*)ADDR_FSTORE, 9 );

        L_writeStr(Rstored);                                        // write saved Rstored
        L_cursorLinePos(1,0);                                       // cursor line 1, position 0
        L_writeStr(Fstored); L_writeStr(" Key OK");                 // show save Fstored
        break;

    case '4':                                                       // Show old K on line 0 and enter new K on line 1
        L_writeStr(Kp);
        L_cursorLinePos(1,0);
        L_cursorVisBlink(true,true);                                // make cursor visible and blinking

        getNumberDsply(1,0,result);                                 // get number starting at line 1 pos 0

        if ( (strlen(result) != 0) && (result[0]!='0') )            // if not blank, put result into global variable
                { strcpy( Kp, result ); }
        L_cursorVisBlink(false,false);                              //invisible cursor, no blink
        break;

    case '5':                                                       // Save to eeprom if ok
        L_writeStr(Kp);
        L_cursorLinePos(1,0);
        L_writeStr("Save: 1:Yes 2:No");
        if(keyGet()=='1') { eeprom_update_block( (const void*)Kp, (void*)ADDR_KP, 14); }
        break;

    case '6':                                                       // RCL from eeprom
        eeprom_read_block( (void*)Kp, (const void*)ADDR_KP, 14 );
        L_writeStr(Kp);
        keyGet();                                                   // wait for key press
        break;

    default: key='E'; break;  } //end switch
  }     // end while
}
```

```
void menuRtrue(void)                                            // Enter, save and recall Rtrue
{       char key = 0;
        while(key != 'E')                                       // continue until key = E = escape
        {       L_clear();                                      // clear LCD

                L_writeStr( "1:R? 2:KRs 3:..."  );              // show/enter Rtr, Rtr from Rst, nothing
                L_cursorLinePos(1,0);                           // cursor to line 1 pos 0
                L_writeStr( "4:R\7 5:R\6  6:..."  );            // R save, R rcl, nothing

                key = keyGet();                                 // wait for key to be pressed
                switch (key)
                { case '1':                                     // show Rtrue or get new Rtrue
                        L_clear();                              // clear LCD
                        L_writeStr(Rtrue);                      // write existing Rtrue to LCD
                        L_cursorLinePos(1,0);                   // set cursor to line 1, position 0
                        L_cursorVisBlink(true,true);            // make cursor visible and blinking

                        getNumberDsply(1,0,result);             // input a number printing at line 1, pos 0
                        if ( (strlen(result) != 0) && (result[0]!=0) ) strcpy(Rtrue,result); // put in global var Rtrue if ok

                        L_cursorVisBlink(false,false);          // invisible cursor, no blink
                        break;

                  case '2':                                     // use R = Kp * Rstored
                        {float64_t fnum = f_strtoF64(Kp);       // need braces {} when defining new variables here
                        float64_t RstoredTemp = f_strtoF64( Rstored );
                        float64_t RtrueTemp = f_mult(fnum,RstoredTemp);     // multiply Kp*Rstore and store in temporary var
                        char* pRes = f_to_string(RtrueTemp,15,0);        // convert temp Rtrue from float64 to char array (pointer)
                        strcpy(result,pRes);                    // copy to temporary string result[]
                        L_clear();                              // clear LCD
                        L_writeStr( result);                    // write result[] of K*Rst to LCD
                        L_cursorLinePos(1,0);                   // place cursor at line 1, position 0
                        L_writeStr( "1:Yes 2:No" );             // Save to global Rtrue if ok to use as Rtrue
                        if ( (keyGet() == '1') && (  f_compare(RtrueTemp, 0) > 0  )  ) { strcpy(Rtrue,result); } }
                        break;
                  case '3': break;                              // nothing, available for development
                  case '4': eeprom_update_block( (const void*)Rtrue, (void*)ADDR_RTRUE, 16); break;          // save to eeprom
                  case '5': eeprom_read_block( (void*)Rtrue, (const void*)ADDR_RTRUE, 16 );  break; //recall from eeprom
                  case '6': break;                              // nothing, available for development
                  default: key='E'; break;                      // escape, done
                }
        } // end while

        return; // result;
}

// display temperature till a key is pressed
void menuTemp(void) //shows temperature
{       //configure ADC modify timer for 4 secs; manual trig ADC; ADC function writes to LCD; get key to exit;
        //reset timer for main routine prior to exit

        TC16_config(TEMP_TIME, F_CPU, false, false, false, 0); //500mSec timing period, fcpu=16MHz
        ADC_Enable();
        ADC_StartConvert();                                  // manual convert for first run
        _delay_ms(5);                                        // need delay as the manual ADC start appears to interfere

        L_clear();                                              // clear LCD
```

```
    L_writeStr("Temp \4C ...");                              // write temperature info 1
    L_cursorLinePos(1,0);
    L_writeStr("Time: ");                                    // write temperature info 2
    L_writeStr( "TEMP_TIME " );
    L_writeStr("mSec");

    TC16_Start();                                            // ADC auto triggers from TC16 every 4seconds

    while ( keyGet() == 0 ) ;

    TC16_Stop();                                             // stop timer which stops ADC trigger
    ADC_Disable();                                           // disable ADC
    TC16_config(USART_TIMEOUT, F_CPU, false, true, true, 0);   //back to 500mSec timing period for USART, fcpu=16MHz
}

//change frequency using up down left right arrows and line toggle
//Fc=Fcode is hex code for FE5650A: standard form =  8digits even if must add leading zeros; no fractions no dec pnt.
//Fo=Fout must be in standard form of 8 integer digits, a dp, and 3 fract digits.
void menuGo(void)
{
    //uint8_t iLine = 1;               //made global: always updated; cursor line index at start
    //uint8_t iPos = 5;                //made global: always updated; cursor position index initialization
    uint8_t iFo = 1;                   //index in Fout
    uint8_t iFc = 0;                   //index in Fcode
    char Fc_try[10] = {0};             //9proposed Fcode - will be in standard form
    char Fo_try[18] = {0};             //16proposed Fout - will be in standard form
    char FTemp[18] = {0};              //16for calculation
    char key = 0;

    bool flagAuto = false;             //auto send means each LCD update is sent to FE5650A
    bool flag_exitIF = false;          // used to exit an IF
    bool flag_Fchanged = false;        //the frequency has changed in GoMenu when true

    strcpy(Fc_try, Fcode);             // note must always have 8 digits
    strcpy( Fo_try, FreqStrConditioner(Fout,true,result)); //put Fo2 into std form: 8 int digits, 3 fract digits

    //========= Start ADC to show temperature
    TC16_config(TEMP_TIME, F_CPU, false, false, false, 0);       //4 Sec timing period, fcpu=16MHz
    ADC_Enable();
    ADC_StartConvert();                                          //must start even for free run
    _delay_ms(10);                                               //the adc conversion interferes with writeStr!?!?
    TC16_Start();                                                //runs the adc
    //=====

    L_clear();                                                   // clear LCD
    L_writeStr("Fc: "); L_writeStr(Fc_try);                      // write proposed Fcode
    L_cursorLinePos(1,0); L_writeStr("Fo: "); L_writeStr(Fo_try);   // write proposed Fout

    L_cursorLinePos(iLine,iPos);                                 // set cursor to previous position
    L_cursorVisBlink(true, false);                               // show cursor but no blink

    while ( ( key = keyGet() ) != 's' )                          // exit only if receive an 's'
    {   flag_Fchanged = false;                                   //Auto Send needs to know if freq changed

        switch( key )
        {   case 't':                                            //toggle current LCD row
                    if (  (iLine ^= 0b01) == 0 ) //Line number went from 1 to 0, so set new array index and cursor position
                      { iFc = FcodeIndexFromFoutIndex(iFo); L_cursorLinePos(0, iPos=cursorPosFromFcFoIndex(iFc) ) ;    }
```

```
        else                                                //Line number went from 0 to 1
          { iFo = FoutIndexFromFcodeIndex(iFc); L_cursorLinePos(1, iPos=cursorPosFromFcFoIndex(iFo) ) ;    }

        break;

case 'a':                                                   //toggle auto send
        flagAuto = !flagAuto;
        if (flagAuto) { PORT_AUTO |= (1 << PIN_AUTO); }     //light LED for auto send
        else { PORT_AUTO &= ~(1 << PIN_AUTO); }             // extinguish LED for auto send
        break;

case 'U': //increase digit, calc new Fout, put Fout in Std Form
        if(iLine == 1)                                                      // a decimal digit is increasing
         {    if (  (Fo_try[iFo] >= '0') && (Fo_try[iFo] < '9') )           // don't exceed '9'
              { Fo_try[iFo] += 1;                                           // increase digit at cursor
                L_erase(1,iPos,iPos); L_cursorLinePos(1,iPos);              // prepare spot for new char
                L_writeChar( Fo_try[iFo]); L_cursorLinePos(1,iPos);         // update the display
                strcpy(FTemp, FreqStrConditioner(Fo_try,false,result) );    // remove lead zeros for calculations
                strcpy( FTemp, FcodeCalc( FTemp, Rtrue) );                  // calc Fcode and store in FTemp
                strcpy( Fc_try, FcodeStrConditioner(FTemp,true,result) ); // make Fc have standard form
                L_cursorLinePos(0,4); L_writeStr(Fc_try);          // write new Fc corresponding to new Fout
                L_cursorLinePos(1,iPos);                           // put cursor back at changed char
                flag_Fchanged = true;  }                           // values have changed
         } //end of if(iLine == 1)

        else                                                // increase HEX Fcode on line zero
        {       flag_exitIF = false;
                if ( Fc_try[iFc] >= '0' && Fc_try[iFc] < '9' ) {Fc_try[iFc] += 1; }      //[0,9) => increase char
                else if ( Fc_try[iFc] == '9' ) { Fc_try[iFc] = 'A'; }                    // if char=9 then make it A
                else if ( Fc_try[iFc] >= 'A' && Fc_try[iFc] < 'F' ) { Fc_try[iFc] += 1; } // [A,F)
                else { flag_exitIF = true; }                                             //at limits of numbering

                if ( ! flag_exitIF )                                                     //nothing changed so exit
                {   L_erase(0,iPos,iPos); L_cursorLinePos(0,iPos);                       //erase char to be changed
                      L_writeChar( Fc_try[iFc] ); L_cursorLinePos(1,iPos);               //update the display
                      strcpy(FTemp, FcodeStrConditioner(Fc_try,false,result) ); //remove lead zeros
                      strcpy( FTemp, freqOutCalc( Rtrue, FTemp) );                       //calc Fout and store in FTemp
                      strcpy( Fo_try, FreqStrConditioner(FTemp,true,result) );           // put calc Fout in Fo_try
                      L_cursorLinePos(1,4);  L_writeStr(Fo_try);                         // overwrite old Fout
                      L_cursorLinePos(0,iPos);                                           // cursor at present char
                      flag_Fchanged = true;}
        } //end else
        break;

case 'D':
        if(iLine == 1)                                                           // decrease a decimal digit
        {     if ( Fo_try[iFo] > '0' && Fo_try[iFo] <= '9' )                     // cannot decrease below 0
              {     Fo_try[iFo] -= 1;                                            // decrease digit
                    L_erase(1,iPos,iPos); L_cursorLinePos(1,iPos);               // erase changed digit
                    L_writeChar( Fo_try[iFo] ); L_cursorLinePos(1,iPos);         //update the char
                    strcpy(FTemp, FreqStrConditioner(Fo_try,false,result) );     //remove lead zeros for calc
                    strcpy( FTemp, FcodeCalc( FTemp, Rtrue) );                   //calc Fcode and store in FTemp
                    strcpy( Fc_try, FcodeStrConditioner(FTemp,true,result) );    //Fc  std from of 8 hex digits
                    L_cursorLinePos(0,4); L_writeStr(Fc_try);              // overwrite old Fc with new one on LCD
                    L_cursorLinePos(1,iPos);                               // place cursor back at changing char
                    flag_Fchanged = true;  }                               // freq has changed
        } // end if
        else                                                               // decrease HEX Fcode digit
```

```
            {   flag_exitIF = false;
                if ( Fc_try[iFc] > '0' && Fc_try[iFc] <= '9' ) {Fc_try[iFc] -= 1; } // (0,9] => decrease digit
                else if ( Fc_try[iFc] == 'A' ) { Fc_try[iFc] = '9'; }              // if char=A then decr to 9
                else if ( Fc_try[iFc] > 'A' && Fc_try[iFc] <= 'F' ) { Fc_try[iFc] -= 1; } // (A,F] decrease by one
                else { flag_exitIF = true; }                                        // exit if no change in Freq

                if ( ! flag_exitIF )                                                // exit if not change in Freq
                {       L_erase(0,iPos,iPos); L_cursorLinePos(0,iPos);              // erase changed char
                        L_writeChar( Fc_try[iFc] ); L_cursorLinePos(1,iPos);        // update the char
                        strcpy(FTemp, FcodeStrConditioner(Fc_try,false,result) );   // remove lead zeros
                        strcpy( FTemp, freqOutCalc( Rtrue, FTemp) );                // calc Fout and store in FTemp
                        strcpy( Fo_try, FreqStrConditioner(FTemp,true,result) );    // standard form for Fout
                        L_cursorLinePos(1,4); L_writeStr(Fo_try);                   // overwrite Fout
                        L_cursorLinePos(0,iPos);                                    // return cursor to char position
                        flag_Fchanged = true; }                                     // frequency changed
            } //end else
            break;

    case 'L':                                   //move Left in row. If row 1, skip dp
            if (iLine == 0)
                { iFc=FcIndexMoveLeft(iFc); L_cursorLinePos(0, iPos=cursorPosFromFcFoIndex(iFc)); }
            else
                { iFo=FoIndexMoveLeft(iFo); L_cursorLinePos(1, iPos=cursorPosFromFcFoIndex(iFo)); }
            break;

    case 'R':                                   //move Right in row. if row 1, skip dp, and not beyond end
            if (iLine == 0)
                { iFc=FcIndexMoveRight(iFc); L_cursorLinePos(0, iPos=cursorPosFromFcFoIndex(iFc));        }
            else
                { iFo=FoIndexMoveRight(iFo); L_cursorLinePos(1, iPos=cursorPosFromFcFoIndex(iFo));  }
            break;

    case 'E':                           // E=escape ... delete changes; if A ON, then this doesn't help
            strcpy(Fc_try, FcodeStrConditioner(Fcode,true,result));     // Use orig Fout; make sure in proper form
            L_cursorLinePos(0,4); L_writeStr(Fc_try);                   // print to LCD
            strcpy(Fo_try, FreqStrConditioner(Fout,true,result));       // Use orig Fout; make sure in proper form

            L_cursorLinePos(1,4); L_writeStr(Fo_try);                   // print to LCD
            L_cursorLinePos(iLine,iPos);                                // go to last LCD position
            flag_Fchanged = false;
            break;                                                      // delete changes; return to main

    case 'c':                                                           // Cr = Manually send Fc to FE5650A;
            strcpy(Fcode,Fc_try);                                       // copy to global Fcode
            strcpy(Fout,Fo_try);                                        // copy to global Fout
            sendFreq(Fcode);                                            // send to FE5650A
            flag_Fchanged = false;
            break;

    default:                                    // does not handle s key due to while() ... D at pad 4,4
            break;

} //end switch



if(flagAuto && flag_Fchanged)                               // if Auto then save to global and send to FE
{       //strcpy(Fcode,Fc_try );                            // Fc2 in standard form and so will be Fcode
```

```
            //strcpy(Fout,Fo_try);
            sendFreq(Fc_try);                                   // send to FE5650A
            flag_Fchanged = false;  }
    } //end while

    strcpy(Fcode,Fc_try);                                       // copy to global Fcode
    strcpy(Fout,Fo_try);                                        // copy to global Fout

    L_cursorVisBlink(false, false);                             // Set cursor off, no blink

    //======= Stop ADC and return TC16 to USART service
    TC16_Stop();                                                // Stop timer
    ADC_Disable();                                              // Stop ADC
    TC16_config(USART_TIMEOUT, F_CPU, false, true, true, 0);    // reconfigure timer for use with USART
    _delay_ms(5);
    //=======
return;
}


// ===========================================================
// =================== KEYPAD and LCD ======================
// ===========================================================

//get a number from keypad while entering digits to display
//pressing ESC erases all results and returns 0
//pressing Cr returns the entered numbers as string of chars
char* getNumberDsply(uint8_t Line, uint8_t Pos, char* result)
{   char c = 0;
    uint8_t len = 0;                            //   index into result[], also give num characters in array
    uint8_t dispPos = Pos;                      //   dispPos is display position on LCD line; next available
position
    bool flag_dp = false;                       //   dp is found
    bool flag_ESC = false;                      //   Escape E has been pressed
    bool flag_CR = false;                       //  Cr c has been pressed

    do
    {   c=0;
        while(c==0) {c=keyPad();}               // waits for a key to be pressed; could use keyGet()

        if(  ( c=='0' || c=='.' ) && len==0)    //leading 0 must be followed by a dp
            { result[len++] = '0'; L_cursorLinePos(Line,dispPos); L_writeChar('0');
              result[len++] = '.'; L_writeChar('.'); dispPos+=2; }
        else if( c >= '0' && c <= '9' )
            {if ( len != 1 || result[0] != '0')         //prevent backspace from writing 00.123 or 0123.56
                    { result[len++] = c; L_cursorLinePos(Line,dispPos); L_writeChar(c); dispPos++; } }
        else if (c == 'B' && len > 0)           //backspace key deletes
            { result[--len] = 0; L_cursorLinePos(Line, --dispPos); L_writeChar(' '); L_cursorLinePos(Line,dispPos);}
        else if ( c == '.' )                    //only one dp allowed ... none if max=0
            { for (int i=0; i<len; i++)
                    { if ( result[i] == '.' ) flag_dp = true; }     //check all chars for a dp
                    if (flag_dp == false)
                        { result[len++] = c; L_cursorLinePos(Line,dispPos); L_writeChar(c); dispPos++; }
            flag_dp = false;     }              //end else if ( c == '.' )
        else if ( c == 'E' ) { flag_ESC = true; }   //escape wo changing
        else if ( c == 'c') {flag_CR = true;}
        else {;}
```

```
        } while( !( flag_ESC || flag_CR  )  );

        result[len]=0;                              //add string terminator
        if(flag_ESC)                                // delete display line and string in result[]
        { L_erase(Line, Pos, dispPos);
        for (int i = 0; i<len; i++) result[i]=0; }

    return result;
}


// =================          ===============================================
// ================= FUNCTIONS: ARRAY AND CURSOR ==================
// =============================================================

// Gives cursor position when moving between LCD Fcode (line 0)
// and LCD Fout (line 1) since Fcode has fewer characters than
// Fout and must skip dp. Used in GoMenu.
uint8_t FcodeIndexFromFoutIndex(uint8_t iFo)
{   uint8_t iFc = 0;
    switch (iFo)                                     // position index in Fout[]: for ex., iFo=2 gives '3'
    {   case 9: case 10: case 11: iFc = 7; break;    // dec digits 9-11 primarily set by iFc digit 7
        case 7: iFc = 5; break;                      // dec digit 7 mostly associated with changes at iFc=5
        case 6: iFc = 4; break;
        case 5: case 4: iFc = 3; break;
        case 3: iFc = 2; break;
    default: iFc = iFo; break;  }
return iFc;     }

// Gives cursor position when moving between LCD Fcode (line 0)
// and LCD Fout (line 1) since Fcode has fewer characters than
// Fout and must skip dp. Used in GoMenu.
uint8_t FoutIndexFromFcodeIndex(uint8_t iFc)
{   uint8_t iFo = 0;
    switch (iFc)
    {   case 7: iFo = 10; break;
        case 6: iFo = 9; break;
        case 5: iFo = 7; break;
        case 4: iFo = 6; break;
        case 3: iFo = 5; break;
        case 2: iFo = 3; break;
    default: iFo = iFc;  }
    return iFo;   }

// LCD display, for Fout, must add 4
uint8_t cursorPosFromFcFoIndex(uint8_t ArryIndex)
{   return ArryIndex + 4; }

uint8_t FcIndexMoveRight(uint8_t iFc )
{ if ( (iFc >= 0) && (iFc < 7) ) { iFc++; }          // cannot move above index = 7
return iFc;   }

uint8_t FcIndexMoveLeft(uint8_t iFc)                 // cannot move below index = 0
{   if (  (iFc > 0) && (iFc <= 7) ) iFc--;
return iFc;  }


uint8_t FoIndexMoveRight(uint8_t iFo )
```

```
{ if ( (iFo >= 0) && (iFo < 7) ) iFo++;                          //for [0,7] can incr array index (i.e., move right)
      else if ( iFo == 7 ) iFo += 2;                             //skip dp
      else if (  (iFo > 8) && (iFo <11) ) iFo++;                 //move right but not past index 11: third digit after dp
      else ;
return iFo; }

// move left through Fout array: Fout in standard form: 8digits before dp, then dp, then 3dig after dp.
uint8_t FoIndexMoveLeft(uint8_t iFo)
{ if (  (iFo > 0) && (iFo < 8)  ) iFo--;                         // OK to decrease iFo but not past the first char
      else if (  iFo == 9  ) { iFo -= 2; }                       // skip dp
      else if (  (iFo > 9) && (iFo <= 11) )  iFo--;              // Only as low as the digit to right of dp
      else ;
      return iFo;  }

//adds or removes leading zeros that bring the Hex number to eight digits
//Note: assumes no decimal points -- if found, they will be removed
char* FcodeStrConditioner(char* existStr, bool Flag_T2AddZeros_F2RemoveZeros, char* result)
{     char* pExist = existStr;                                   //pointer to first char in existStr
      char* pDP = strchr(existStr,'.');                          // points to dp, if none then pDP=0
      for(int8_t i = 0; i < 16; i++) { result[i] = 0; }          // set result[] to \0
      int len = 0;
      int j = 0;

      if (pDP != 0)  //replace unwanted chars w \0;
      {   while( *pDP != 0 ) { *pDP++ = 0;  }      }              // remove dp if present and any fractional digits

      len = strlen( existStr ) ;
      pExist = existStr + len - 1;                               //set pExist pointer to last char
      j = 7; //num of required chars
      while(  ( pExist >= existStr ) && ( j >= 0 ) ) { result[j--] = *pExist--; } //copy starting at back. front slots empty
      for ( ; j >= 0 ; j--) { result[j] = '0'; }                 //add zeros to front if needed

      //now remove leading zero if needed
      if ( !Flag_T2AddZeros_F2RemoveZeros )
      {     char* pRes = result;                                 // pointer to first char in result[]
            j = 0;                                               // use this as a counter
            while(  *pRes++ == '0'  ) j++;                       // j will be # shifts of result[] to left to remove '0's

            uint8_t lenRes = strlen(result);
            for(int i = j; i < lenRes; i++) { result[i-j] = result[i]; }
      result[ lenRes - j ] = 0;   }
      return result;
}

// provides leading zeros ( or removes them for flag = false)
char* FreqStrConditioner(char* existStr, bool Flag_T2AddZeros_F2RemoveZeros, char* result)
{     // For the display, form = 08388608.001: 3 fract digs, 8 int digs: 8+1+3=12, DP at i=8
      char* pExist = existStr;                                   // pointer to first char in existStr
      char* pDP = strchr(existStr,'.');                          // pointer to dp in existStr, returns 0 if not there
      for(int8_t i = 0; i < 16; i++) { result[i] = 0; }          // set result[] entries to \0 just to clear it out
      result[8] = '.';                                           // place dp at proper place
      int len = 0;
      int j = 0;

      // FRACTION PART make sure three digits after dp
      if (pDP == 0) { for(int i = 9; i<12; i++) { result[i] = '0'; } } // if no dp in original str, then place three 0s
      else                                                          // get 3 digits past dp, if not there, add '0'
      {     j = 0;
```

```
        pExist = pDP + 1;                                          //skip dp; each char only 1 byte in memory
        while( (j<3) && (*pExist != 0) ) { result[9 + j++] = *pExist++; } //insert chars until existingStr presents \0
        for(int i = j; i<3; i++) { result[9+i] = '0'; }            //add zeros so x.1 => x.100
    }

    j = 0; //can use j with char pointers

    // INTEGER PART of existStr: make sure 8 digits before dp -- add zeros if not
    if (pDP == 0) { len = strlen(existStr);  }                     // j should count number of zeros to add
    else { len = pDP - existStr ; }                                // need number of digits prior to dp
    pExist = existStr + len - 1;                                   //set pExist pointer to last entry before dp or end

    j = 7;                                                         //save in result[] at left of DP
    while(  ( j >= 0 ) && ( pExist >= existStr  ) ) { result[j--] = *pExist--; } //copy starting at back until pointers equal
    for ( ; j >= 0 ; j--) { result[j] = '0'; }                     //add zeros to front if needed

    //now remove leading zero if needed; even though they might have been added.
    if ( !Flag_T2AddZeros_F2RemoveZeros )
    {
        char* pRes = result;                                       // point to start of result[]
        j = 0;                                                     //use this as a counter
        while(  ( *pRes++ == '0' ) && ( *pRes != '.' )  ) j++;     // j=num of '0' to remove; note ++ checks next char

        uint8_t lenRes = strlen(result);
        for(int i = j; i < lenRes; i++) { result[i-j] = result[i]; } //j = # shifts of result[] to left,
        result[ lenRes - j ] = 0;                                  //terminator
    }
    return result;
}

// ===========================================================================
// ====================== Fout and Fcode Calculations ======================
// ===========================================================================
//Calcs F = R*Fcode/2^32; Fcode does not include 0x
char* freqOutCalc(char* RtrueStr, char* FcodeStr)
{   // typical numbers: R = 50255055.118773, Fcode = 2ABB5050 ; note Fcode does not use 0x
    float64_t F1 = f_strtoF64(RtrueStr);                        // converts Rtrue[] to a float64 number
    float64_t TwoPow16F64 = f_long_to_float64(pow(2,16));       // divide using this twice to get #/2^32

    uint32_t FcUint32 = strtoul(FcodeStr,0,16);                 // HEX Str to UL = UINT32_t
    float64_t Fc = f_long_to_float64(FcUint32);                 // get float64 corresponding to FcUint32

    F1 = f_div(F1,TwoPow16F64);                                 // Gives Rtrue/2^16
    F1 = f_div(F1,TwoPow16F64);                                 // Gives Rtrue/2^32
    F1 = f_mult(F1, Fc);                                        // Gives result of Fcode*Rtrue/2^32
    return f_to_string(F1,15,0);   }

// calculates Rtrue = Kp * Rstored
char* RtrFromRst(char* RstStr, char* KpStr)
{   float64_t RtrF64 = f_strtoF64(RstStr);                      // temporarily store first factor Rstored
    float64_t KpF64 = f_strtoF64(KpStr);                        // second factor of Kp
    RtrF64 = f_mult(RtrF64,KpF64);                              // result is Kp*Rstored
    return f_to_string(RtrF64, 15, 0);  }

// calculates Fcode = Fout (2^32 / R). Rounds Fcode to nearest integer
char* FcodeCalc(char* FoutStr, char* RtrueStr)
{   // example numbers: R = 50255055.118773, Fcode = 2ABB5050; note Fcode does not use 0x
    float64_t TwoPow16F64 = f_long_to_float64(pow(2,16));          // Stores 2^16
```

```
    float64_t Fc1 = TwoPow16F64;
    Fc1 = f_mult(Fc1, TwoPow16F64 );                    // now have 2^32

    float64_t num = f_strtoF64(RtrueStr);               // temporarily store Rtrue
    Fc1 = f_div(Fc1,num);                               // now have ( 2^32 / Rtrue )

    num = f_strtoF64(FoutStr);                          // temporarily store Fout
    Fc1 = f_mult(Fc1,num);                              // now have Fcod = ( 2^32 / Rtrue ) * Fout

    return f_ftoHexIntStr(Fc1, true);
}


// =====================================================
// ==================== uC Setup =======================
// =====================================================
void uC_Init(void)
{
    // configure caps lock and auto lock
    DDR_CAPSLOK |= (1 << PIN_CAPSLOK);                  //LED for Caps lock
    PORT_CAPSLOK &= ~(1 << PIN_CAPSLOK);                //starts off
    DDR_AUTO |= (1<<PIN_AUTO);                          //LED for Auto Send
    PORT_AUTO &= ~(1 << PIN_AUTO);

    //Has the eeprom been initialized with values?
    eeprom_read_block( (void*)&result, (const void*)ADDR_ALL, 4 );
    if ( strcmp("ALL", result) != 0)                                //initialize for first run
        {   eeprom_update_block( (const void*)"ALL", (void*)ADDR_ALL, 4);   //includes ALL and \0
            eeprom_update_block( (const void*)Rstored, (void*)ADDR_RSTORE, 16);
            eeprom_update_block( (const void*)Fstored, (void*)ADDR_FSTORE, 9);
            eeprom_update_block( (const void*)Kp, (void*)ADDR_KP, 14);
            eeprom_update_block( (const void*)Rtrue, (void*)ADDR_RTRUE, 16);
            eeprom_update_block( (const void*)Fcode, (void*)ADDR_FCODE, 9);
            eeprom_update_block( (const void*)Fout, (void*)ADDR_FOUT, 13);
            eeprom_update_word( (uint16_t*)ADDR_BAUD, baud ) ;   }

    else                                                    //not first run, read saved parms
        {  eeprom_read_block( (void*)Rstored, (const void*)ADDR_RSTORE, 16 );
           eeprom_read_block( (void*)Fstored, (const void*)ADDR_FSTORE, 9 );
           eeprom_read_block( (void*)Kp, (const void*)ADDR_KP, 14 );
           eeprom_read_block( (void*)Rtrue, (const void*)ADDR_RTRUE, 16 );
           eeprom_read_block( (void*)Fcode, (const void*)ADDR_FCODE, 9 );
           eeprom_read_block( (void*)Fout, (const void*)ADDR_FOUT, 13 );
           baud = eeprom_read_word( (uint16_t*)ADDR_BAUD ); }
}


void delay_ms(int ms)
{   for(int i = 0; i < ms; i++)
    { _delay_ms(1); } }
```

Section A6.2: ADC328.h

```
#ifndef ADC328_H_
#define ADC328_H_

enum ADC_Ref {AREF=0, AVcc, Rsvd, Intrnl};                    //need order of 0, 1, 2, 3 ...

enum ADC_TrigSrc {FreeRun=0, AnlgComp, ExtIntptReq0, TC0CompMatA, TC0OvrFlw,TC1CompMatB, TC1OvrFlw, TC1CapEvnt};

enum ADC_Prescale {zeroDefaultDiv2=0, Div2, Div4, Div8, Div16, Div32, Div64, Div128};

bool ADCisDone(void);
void ADCsetDoneFlag(bool flagset);
int32_t adc_getResult(void);

void ADC_Config(ADC_Ref aRef, enum ADC_TrigSrc aTrig, enum ADC_Prescale aPresc,
 uint8_t muxChan, bool aAutoTrig, bool aInterruptEnable, bool callFunction, bool FrawTmv );

void ADC_Enable(void);
void ADC_Disable(void);
void ADC_StartConvert(void);

//declared here for access; defined in main module
void ADC_calledFunction(uint16_t adcResult);

#endif /* ADC328_H_ */
```

Section A6.3: ADC328.cpp

```
#include <avr/interrupt.h>
#include "ADC328.h"

volatile bool flag_ADCdone = false;
bool flag_allowCallFunction = false;          // directs ISR to call function in main module
bool flag_FrawTmv = false;                    // if False then return raw, if True then millivolts
bool flag_allowInterrupt = false;             // allow the ISR

uint16_t adcResult = 0;                       // either raw ADC bits, or millivolts

bool ADCisDone(void)                          // external control over flag
{       return flag_ADCdone; }

void ADCsetDoneFlag(bool flagset)             // external control over flag
{ flag_ADCdone = flagset; }


int32_t adc_getResult(void)                   // call to read ADC results
{       return adcResult; }

// uses enums defined in the header for ADC_Ref, ADC_TrigSrc, ADC_Prescale -- refer to ADC in the mega328 data sheet
void ADC_Config(ADC_Ref aRef, enum ADC_TrigSrc aTrig, enum ADC_Prescale aPresc,
        uint8_t muxChan, bool aAutoTrig, bool aInterruptEnable,
        bool callFunction, bool FrawTmv )
{       //shut down ADC for changes
        ADCSRA =0;                            //disable the ADC
```

```
	ADMUX = 0;

	flag_allowCallFunction = callFunction;		// set module flags
	flag_FrawTmv = FrawTmv;
	flag_allowInterrupt = aInterruptEnable;

	ADMUX |= ( ( aRef << REFS0 ) | muxChan ) ;	//select reference and set channel
	ADCSRA |= aPresc;				//set prescalar; the ADC needs 50kHz to 200kHz

	ADCSRB = 0;
	ADCSRB |= aTrig;

	// now need to set ADCSRA: Enable, ADCStart, ADC AutoTrig, intrp enable
	if (aAutoTrig) ADCSRA |= ( 1 << ADATE );
	else ADCSRA &= ~( 1 << ADATE );

	if (aInterruptEnable) { ADCSRA |= ( 1 << ADIE); }
	else { ADCSRA &= ~( 1 << ADIE ); }

	sei();						//enable global interrupts
}

void ADC_Enable(void)
{  ADCSRA |= ( 1 << ADEN );}				//enable requires 12 cpu clock cycles

void ADC_Disable(void)
{  ADCSRA &= ~( 1 << ADEN ); }

void ADC_StartConvert(void)
{  ADCSRA |= 1 << ADSC; }

ISR(ADC_vect)
{	flag_ADCdone = true;
	if(flag_FrawTmv)
		{	adcResult = uint16_t ( ( 1100.0 * ADCW ) /1024.0 ); }
	else
		{	adcResult = ADCW;  }

	if(flag_allowCallFunction) ADC_calledFunction(adcResult);  }
```

Section A6.4: KeyPad4x4.h

```
#ifndef KEYPAD4X4_H_
#define KEYPAD4X4_H_

//macros to define a token string as a C/C++ identifier
//KEYPAD
#define PORT_KEYROW PORTB
#define DDR_KEYROW DDRB			//Already set for display to out
#define PORT_KEYCOL PIND			//input
#define DDR_KEYCOL DDRD

#define PORT_CAPSLOK PORTB
#define DDR_CAPSLOK DDRB
#define PORT_AUTO PORTB
#define DDR_AUTO DDRB
```

```
#define PIN_CAPSLOK 4                    //pin B4 in PORTB for caps lock LED
#define PIN_AUTO 5                       //pin B5  in PORTB for AUTO LED


void keyPad_init(void);                  // initialize the keypad
char keyPad(void);                       // run an entire scan of keypad but return a pressed key
char keyRead(uint8_t row, uint8_t col ); // if key at row col is pressed, return char value
char keyGet(void);                       // wait till any key pressed, return the corresponding char

#endif /* KEYPAD4X4_H_ */
```

Section A6.5: KeyPad4x4.cpp

```
#define F_CPU 16000000UL

#include <avr/io.h>
#include "KeyPad4x4.h"
#include <util/delay.h>

void delayK_ms(int ms);
void keyPad_init(void);

// define the physical pins for the keypad rows and cols
// For rows, the physical pins are B0 B1 B2 B3
static uint8_t pinsRows[4] = {0, 1, 2, 3};         //pin0 addresses key labeled as 1 2 3 A etc
static uint8_t pinsCols[4] = { 2, 3, 4, 5 };       //pin2 addresses the keys labeled as 1 4 7 *

void keyPad_init(void)
{     //define DDR_KEYROW
      for (int i = 0; i<4; i++) {   DDR_KEYROW  |= ( 1 << pinsRows[i] ); }   //make KEYROW (PORTB) an output
      for (int i = 0; i<4; i++) {   DDR_KEYCOL &= ~( 1 << pinsCols[i] ); }   //make KEYCOL (PORTD) an input
}

//make 1 scan of pad but stop when key found press
// voltage pulse applied to a row; column checked for pulse by key closure
char keyPad(void)
{
      uint8_t row = 0;                                  // rows carry the voltage
      uint8_t col = 0;
      char key = 0;

      while (  (row < 4) && (key == 0) )                // for each row
      {     col = 0;
            while ( (col < 4) && (key == 0) )           // check each column until key pressed
            { key = keyRead(row,col); col++; }
      row++;  }
      return key;
}

// Apply voltage to a Row (0 or VCC), Read voltage on a col pin.
// Get determines if single key pressed at Row, Col
char keyRead( uint8_t row, uint8_t col )
{
      // Assigns ASCII values to a keypad row and col
      // dimensioned as two 4x4 arrays key[2][4][4]
      // key[0][i][j] give the caps UNlocked case
```

```
    // key[1][i][j] give the caps locked case
    // keep in mind, static vars only dimensioned once
    static uint8_t keys[2][4][4] = {
        {
            { '1', '2',  '3', 'z' },        // 1  2    3    A='z' => caps Lock
            { '4', '5',  '6', 't' },        // 4  5    6    B='t' => toggle Row
            { '7', '8',  '9', 'a' },        // 7  8    9    C='a' => autoSend
            { '.', '0',  'c',  's' }        // *=.     0    #=<cr>    D='s' => select ..prev menu
        },
        {
            { 'B', 'U', 'E', 'z' },         // 1=Backsp 2=Up   3=ESC    A=lock
            { 'L',  0,   'R', 't' },        // 4=Left    5=na    6=Rght   B= t
            { 'M', 'D',  'C', 'a' },        // 7=Menu 8=dwn  9=Clr    C =auto send
            { '.',  0,     'c', 's' }       // *=.       0=na    #=<cr>   D=s=select ... prev menu
        }
    };
    static uint8_t caps = 0;

    char key = 0;

    uint8_t rowPin = pinsRows[row];         // select a row and col
    uint8_t colPin = pinsCols[col];

    PORT_KEYROW  |=  1 << rowPin;                                   // apply +5V to row

    delayK_ms(1);                                                   // wait for voltage to stabilize
    while( ( PORT_KEYCOL & ( 1 << colPin ) ) == ( 1 << colPin ) )   // if yes then set key
    {     key = keys[caps][row][col];
          delayK_ms(20);  }                                         // make sure bounce stops

    if (key == 'z') {caps ^= 0b01; key = 0;}                        // specially handle caps ... toggle caps but key=0
    if (caps == 0) { PORT_CAPSLOK &= ~(1 << PIN_CAPSLOK);}          // set cap Lock LED
    else { PORT_CAPSLOK  |= (1 << PIN_CAPSLOK);  }

    PORT_KEYROW &=  ~(1 << rowPin);                                 //apply 0V to row
    return key;
}

// waits till a key pressed. Returns character corresponding to the key
char keyGet(void)
{     char key = 0;
      while (key==0) {key=keyPad();}
return key;            }

void delayK_ms(int ms)
{     for(int i = 0; i < ms; i++)
      { _delay_ms(1); }  }
```

Section A6.6: LCD16x2_ST7032.h

```
#ifndef LCD16X2_ST7032_H_
#define LCD16X2_ST7032_H_

//======= LCD
#define PORTCON PORTD                   // compiler definition for control port
#define DDRCON DDRD
#define PORTDAT PORTB                   // directive defines port to handle data
```

```
#define DDRDAT DDRB
#define BIT_CD (1<<6)                    //pin 6; RS, register select, 0=command, 1=data
#define BIT_E (1<<7)                     //enable: falling edge triggered

#define MASK_CTRL (0b011 << 6)           //will select the control bits: 6,7

#define MASK_DATA 0x0F                   //data pins on B0-3
#define MASK_HIGH 0xF0                   //will select upper 4 data bits to send first
#define MASK_LOW 0x0F                    //select lower 4 bits of a register

#define COM_WAKEUP 0x30                  // token to represent wakeup command: '0'=>DDRAM
#define COM_8BIT2LINE5x8 0x38            // 8bit input, 2 lines, 5x8 char size
#define COM_INIT4Bit 0x20                // Initialize 4 bit mode
#define COM_4BIT2LINE5X8 0X28            // Command sets 4bit input, 2 lines, 5x8 char size
#define COM_SETCURS 0b0000010100         // Set cursor to move right, no display shifting
#define COM_DispONCurs 0x0C              // 0C= 1100; disp ON, 0E=incl Curs ON, 0F incl blink
#define COM_ENTRYMODE 0x06
        // Move curs right

// ===== Declare LCD functions
void L_command(char cmd, int post_mS);   // sends command to display
void L_writeChar(char c);                // writes data to display
void L_writeStr(const char* str);        // write a string
void L_erase(uint8_t line, uint8_t start, uint8_t stop);            // erase selected characters
void L_cursorLine(uint8_t lineIndex, bool clrNewLine = true);       // has optional clear new line
void L_cursorLinePos(uint8_t lineIndex, uint8_t posIndex );         // goto line and position
void L_cursorVisBlink(bool vis, bool blink);                        //cursor visible/invisible
void L_enablePulse(void);
void LCD_init(void);                                                // initialize display at power ON

void L_buildChar( uint8_t location,  char* ptr);                    // custom character
void L_clear(void);                                                 // clear display

#endif /* LCD16X2_ST7032_H_ */
```

Section A6.7: LCD16x2_ST7032.cpp

```
#define F_CPU 16000000UL

#include <avr/io.h>
#include <string.h>
#include <util/delay.h>
#include "LCD16x2_ST7032.h"

void L_delay_ms(int ms);                                            // delays specified milliseconds

void LCD_init(void)
{       _delay_ms(500);                                             // allow voltages to stabilize
        DDRDAT |= MASK_DATA;                                        // for commands and writes
        PORTDAT &= ~MASK_DATA;
        DDRCON |= MASK_CTRL;                                        // for commands and writes
        PORTCON &= ~MASK_CTRL;

        PORTCON &= ~BIT_E;                                          // set LCD Enable bit to 0
        L_delay_ms(100);

        PORTDAT |= (COM_WAKEUP>>4) & MASK_DATA;                     // get the attention of the LCD
```

```
        L_delay_ms(30);

        L_enablePulse();                                    // clocks in the command/data
        L_delay_ms(10);
        L_enablePulse();                                    // clocks in the command/data
        L_delay_ms(10);
        L_enablePulse();                                    // clocks in the command/data
        L_delay_ms(10);

        // note, up to this point,  the NHD doesn't know it should work in 4bit mode

        PORTDAT &= ~MASK_DATA;                              // clears previous data
        PORTDAT = (COM_INIT4Bit>>4) & MASK_DATA;            // !!! Sets to 4 bit mode
        L_enablePulse();                                    // clocks in the command/data
        L_delay_ms(1);

        L_command(COM_4BIT2LINE5X8,1);                      //set 2 lines and 5x8 character size
        L_command(COM_SETCURS,1);                           // move right
        L_command(COM_DispONCurs,1);                        // display on
        L_command(COM_ENTRYMODE,1);                         // move right


        // Construct special characters. Probably better not to use ASCII 0 since that is normally the NULL code
        char p[8]  = {0x04, 0x0E, 0x0E, 0x0E, 0x1F, 0x00, 0x04, 0x00}; //Bell
        L_buildChar(0, p);                                  // place 'bell' at ASCII 0
        p[0]=0x00; p[1]=0x1B; p[2]=0x00; p[3]=0x04; p[4]=0x04; p[5]=0x11; p[6]=0x0E; p[7]=0x00; //Smiley
        L_buildChar(1, p);                                  // place 'Smiley at ASCII 1
        p[0]=0x00; p[1]=0x10; p[2]=0x08; p[3]=0x04; p[4]=0x0A; p[5]=0x11; p[6]=0x00; p[7]=0x00; //Lambda
        L_buildChar(2, p);                                  // place 'Lambda' at ASCII 2
        p[0]=0x1C; p[1]=0x19; p[2]=0x12; p[3]=0x04; p[4]=0x09; p[5]=0x12; p[6]=0x04; p[7]=0x00; //Shine
        L_buildChar(3, p);                                  // place 'Shine' at ASCII 3
        p[0]=0x07; p[1]=0x05; p[2]=0x07; p[3]=0x00; p[4]=0x00; p[5]=0x00; p[6]=0x00; p[7]=0x00; //Degrees Right symbol
        L_buildChar(4, p);                                  // place 'degrees right' at ASCII 4
        p[0]=0x1C; p[1]=0x14; p[2]=0x1C; p[3]=0x00; p[4]=0x00; p[5]=0x00; p[6]=0x00; p[7]=0x00; //Degrees Left symbol
        L_buildChar(5, p);                                  // place 'degrees left' at ASCII 5
        p[0]=0x03; p[1]=0x17; p[2]=0x1E; p[3]=0x1C; p[4]=0x1E; p[5]=0x00; p[6]=0x00; p[7]=0x00; //arrow left down symbol
        L_buildChar(6, p);                                  // place 'arrow left down' at ASCII 6
        p[0]=0x0F; p[1]=0x07; p[2]=0x0F; p[3]=0x1D; p[4]=0x18; p[5]=0x00; p[6]=0x00; p[7]=0x00; //arrow right up symbol
        L_buildChar(7, p);                                  // place 'arrow up right' at ASCII 7
        L_cursorLinePos(0,1);                  //must use DDRAM address else the CGRAM is used and more chars added.

        L_clear();   }

void L_delay_ms(int ms)                                     // delay milliseconds
{       for(int i = 0; i < ms; i++)
        { _delay_ms(1); }  }

void L_erase(uint8_t line, uint8_t start, uint8_t stop)     // erase 'line' from position 'start' thru 'stop'
{       if (start > stop) { uint8_t temp = stop; stop = start; start = temp; }
        if (start > 15) {start = 15;}                       // make sure start not out of range
        if (stop > 15) {stop = 15;}                         // make sure stop not out of range
        L_cursorLinePos(line,start);                        // cursor back to start position
        for (int i = start; i <= stop; i++) { L_writeChar(' '); }    }    // clear by writing a space

// sends command cmd to display, upper 4 bits go first; different cmds require different delays post_mS
void L_command(char cmd, int post_mS)
{       PORTDAT &= ~MASK_DATA;                              //clears only data pins
        PORTCON &= ~MASK_CTRL;                              //clears control pins
```

```
        PORTDAT |= (cmd >> 4) & MASK_LOW;                    // upper 4bits on data lines, no change to ctrl lines
        L_enablePulse();                                     // pulse clocks in the command bits

        PORTDAT &= ~MASK_DATA;                               // clears 4 data bits
        PORTDAT |= cmd & MASK_DATA;                          // place lower 4 bits on data lines
        L_enablePulse();                                     // clock in bits and execute command

        L_delay_ms(post_mS);        }                        // delay time for command to execute

// write character at current cursor line and position
void L_writeChar(char c)
{       PORTDAT &= ~MASK_DATA;                               //clears upper 4 bits
        PORTCON &= ~MASK_CTRL;

        PORTCON |= BIT_CD;                                   // Set RS bit to 1 to indicate Data (i.e., char)

        PORTDAT |= (c >> 4) & MASK_DATA;                     // place char upper 4bits on data lines, no CTRL change
        L_enablePulse();                                     // clock in bits

        PORTDAT &= ~MASK_DATA;                               // clear bits to zero
        PORTDAT |= c & MASK_DATA;                            // place lower 4 bits on data line

        L_enablePulse();                                     // clock in bits and executed command
        L_delay_ms(1);      }                                // delay 1mSec for command to complete

// writes a string but no check overflow on line
void L_writeStr(const char* str)
{       int L = strlen(str);
        for (int i = 0; i < L ; i++) { L_writeChar(str[i]); }  }

// Set cursor at start of line 0 or 1 on the display
// set clrNewLine = false to stop new line from clearing
// set clrNewLine = true to clear the new line
void L_cursorLine(uint8_t lineIndex, bool clrNewLine)
{       int ddramAddr = 0b10000000;                          // address line 0
        if (lineIndex != 0) {ddramAddr = 0b11000000;}        // address line 1 = 0b1000000+0xC0
        L_command(ddramAddr,1);                              // send address to display, delay 1mS
        if (clrNewLine)                                      // if clearNewLine true then write spaces over 8 chars
        { for (int i = 0; i <8; i++) { L_writeChar(' '); }   // write 'spaces' to clear
        L_command(ddramAddr,1);}  }                          // set cursor back to start position

// goto line 0,1 and positions 0-15
void L_cursorLinePos(uint8_t lineIndex, uint8_t posIndex )   // set cursor to 'line' and 'Pos'
{       int ddramAddr = 0b10000000;                          // address line 0
        if (lineIndex != 0) {ddramAddr = 0b11000000;}        // address line 1
        if (posIndex > 15) {posIndex=15;}                    // keep position in 0 to 7
        ddramAddr += posIndex;                               // calculate address
        L_command(ddramAddr,1);   }                          // send address and execute

void L_cursorVisBlink(bool vis, bool blink)                  // make cursor visible; make it blink
{  uint8_t Cmd = 0b00001100;                        //1000=cmd base, 0100=display ON; cursor not vis, no blink
        if (vis) Cmd += 0b010;                               //make cursor visible
        if (blink) Cmd += 0b001;                             //cursor blinks
        L_command(Cmd,3); }

void L_enablePulse(void)
{       PORTCON |= BIT_E;                                    // Set enable bit to high
```

```
          L_delay_ms(1);                                  // E pulse width
          PORTCON &= ~BIT_E; }                            // New Haven claim: trigger on falling edge


//Routine places characters in ASCII positions 0x00 up to 0x0F
void L_buildChar( uint8_t location,  char* ptr)
{         if(location<8)
          {         L_command(0x40+(location*8),1);       //set CGRAM address
                    for(int i=0; i<8; i++) {L_writeChar( ptr[i] );} }  }

void L_clear(void)
{         L_command(0x01,3); }
```

Section A6.8 myF64.h

```
#ifndef MYF64_H_
#define MYF64_H_

#include <inttypes.h>
#include <stdlib.h>
#include <string.h>
#include <avr/pgmspace.h>

typedef uint64_t float64_t; // IEEE 754 double precision floating point number
typedef float float32_t; // IEEE 754 single precision floating point number

float64_t f_long_to_float64(long n); // Converts a long to the float64_t representing the same number

// Converts a float64_t x to long by cutting the noninteger fraction of x and returning the integer fraction.
// If x is nonnegative and less then 2^^32,   (unsigned long)f_float64_to_long(x)
// yields the correct value. If x is negative and not less than  -2^^31,  f_float64_to_long(x)
// yields the correct value. If the absolute value of x is 2^^32 or greater, f_float64_to_long(x) returns zero.
// ATTENTION: If -2^^32 < x < -2^^31, f_float64_to_long(x) returns 2^^32 plus the integer fraction of x.
long f_float64_to_long(float64_t x);

float64_t f_sd(float32_t fx);                   // Converts a float32 to the float64 representing the same number.
// Denormalized 32-Bit single precision numbers are handled correctly and result in non denormalized double precision
// numbers.
float32_t f_ds(float64_t fx);   // Converts a float64 to the nearest float32. Although denormalized float64's other
// than zero are not supported, f_ds(x) returns a denormalized float32 if the absolute value of x is small enough
// or zero (zero itself is also denormalized).

float64_t f_add(float64_t a, float64_t b);          // Returns a+b . Special case: -INF + INF = NaN
float64_t f_sub(float64_t a, float64_t b);          // Returns a-b . Special case:  INF - INF = NaN
float64_t f_mult(float64_t fa, float64_t fb);       // Returns a*b . Special case: +/-INF * 0 = NaN
float64_t f_div(float64_t x, float64_t y);          // Returns a/b . Special cases: x/0=NaN , INF/INF = NaN , x/INF = 0 if x!=+/-INF


float64_t f_abs(float64_t x);                       // Returns the absolute value of x
float64_t f_cut_noninteger_fraction(float64_t x);// Returns the integer part of x by cutting the noninteger part.

float64_t f_mod(float64_t x, float64_t y, float64_t *ganz); // Returns the floating-point
// remainder f of x / y such that x = i * y + f, where i is an integer, f has the same sign as x,
// and the absolute value of f is less than the absolute value of y. If ganz!=0 is passed, the
// value i is assigned to *ganz.
```

```
int8_t f_finite(float64_t x);                   // Returns nonzero if x represents a real number and zero otherwise i.e. if
// x is +INF, -INF or NaN.

// returns zero if x is equal to y, positive nonzero if x > y and negative nonzero if x < y.
int8_t f_compare(float64_t x, float64_t y);
// If both x and y represent real numbers
// (or +/-INF if F_ONLY_NAN_NO_INFINITY is not defined)
// If x or y are NaN, f_compare returns zero.


// f_to_decimalExp() converts the float64 to the decimal representation of the number x if x is
// a real number or to the strings "+INF", "-INF", "NaN". If x is real, f_to_decimalExp() generates
// a mantisse-exponent decimal representation of x using anz_dezimal_mantisse decimal digits for
// the mantisse. If MantisseUndExponentGetrennt!=0 is passed f_to_decimalExp() will generate different
// strings for the mantisse and the exponent. If you assign
//     char *str=f_to_decimalExp(x, anz_mts, 1, 0)
// then str points to the mantisse string and str+strlen(str) points to the exponent string.
// If the pointer ExponentBasis10 passed to f_to_decimalExp() is nonzero, the function will
// assign the 10-exponent to *ExponentBasis10 ; e.g. if the decimal representation of x
// is 1.234E58 then the integer 58 is assigned to *ExponentBasis10.

char *f_to_decimalExp(float64_t x, uint8_t anz_dezimal_mantisse, uint8_t MantisseUndExponentGetrennt, int16_t *ExponentBasis10);


char *f_to_string(float64_t x, uint8_t max_nr_chars, uint8_t max_leading_mantisse_zeros);
// f_to_decimalExp() converts the float64 to the decimal representation of the number x if x is
// a real number or to the strings "+INF", "-INF", "NaN". If x is real, f_to_decimalExp() generates
// a decimal representation without or with mantisse-exponent representation depending on
// what in more suitable or possible. If -1 < x < 1 the exponent free representation is chosen if
// there are less than 'max_leading_mantisse_zeros' zeros after the decimal point.
// In most cases f_to_string() will generate a string with maximal 'max_nr_chars' chars. If
// necessary, f_to_string() reduces the number of decimal digits in the mantisse in order to
// get a maximum string width of 'max_nr_chars' chars. If however max_nr_chars is so small that
// even a mantisse of one digit and the corresponding exponent doesn't fit into 'max_nr_chars' chars,
// the string returned will be longer than 'max_nr_chars' chars.


// Converts a decimal representation of a real number
// of "INF", "+INF", "-INF", "NaN" into the float64 representing the same number of the non-real object.
// The string str must be in the usual format with or without 10-exponent, e.g.
//     1.234  ,  -89.32  , .001  ,  1E100   , -10.8432E32
// If a nonzero pointer to char-pointer is passed as endptr, f_strtod() will assign the char* to
// *endptr which points to the char (or to zero) that terminates the scan of the string str,
// i.e. the first char after the decimal number string or zero.
float64_t f_strtod(char *str, char **endptr);

#define f_atof(str) (f_strtod((str), 0))

#define float64_NUMBER_ONE          ((float64_t)0x3ff0000000000000LLU)   // 1.0
#define float64_NUMBER_PLUS_ZERO    ((float64_t)0x0000000000000000LLU)   // 0.0
#define float64_ONE_POSSIBLE_NAN_REPRESENTATION    ((float64_t)0x7fffffffffffffffLLU)          // NaN
#define float64_PLUS_INFINITY       ((float64_t)0x7ff0000000000000LLU)   // +INF
#define float64_MINUS_INFINITY      ((float64_t)0xfff0000000000000LLU)   // -INF

float64_t f_strtoF64(char* str);
char* f_ftoHexIntStr(float64_t x, bool flagAllowRounding );

#endif /* MYF64_H_ */
```

Section A6.9: myF64.cpp

```
/*

// The Copyright notice, by law, should not be deleted


//  ===  ATTENTION ===
//     The functions f_to_decimalExp() and f_to_string() uint32toString (and others)
//     return pointers to static memory containing the decimal representation of the float64 passed to these
//    functions. The string contained in this memory will become invalid if one of the functions uint32toString,
//    f_to_decimalExp(), f_to_string(), f_exp(), f_log(), f_sin(), f_cos(), f_tan(), f_arcsin(), f_arccos(),
//    f_arctan() and others is called as these functions will overwrite the memory.

The library myF64 contains methods from both Internet sources and some written by the present author
(M.A.Parker) for the FE5650A including:
      char* f_ftoHexIntStr(float64_t x, char* result, bool flagAllowRounding);
      float64_t f_strtoF64(char* str, bool* pError);
      char* f_uint32toa(uint32_t value, uint8_t base);
These can be used/distributed subject to the following copyright and license.

The F64.c/h can be found on the Internet through the following links
https://www.avrfreaks.net/comment/596905#comment-596905
The original was written by 'Detlef _. (detlef_a)' on or about '02.12.2007'
It was posted at https://www.mikrocontroller.net/topic/85256 (need to translate from German). There have since
been updates/edits that might appear below by subsequent authors such as Florian Königstein. The routines listed
here do not include the log and trigonometric functions. The present author has corrected several minor errors and
eliminated flash storage  non C++ programming, and +/- infinity notation.

// Copyright:                                                                                            //
//     You may use this program free of charge for any purpose you wish provided you agree to the following   //
//     License:                                                                                          //
//     Warranty of Provenance and Disclaimer of Warranty. Licensor warrants that the copyright           //
//     in and to the Original Work and the patent rights granted herein by Licensor are owned by the     //
//     Licensor or are sublicensed to You under the terms of this License with the permission of the     //
//     contributor(s) of those copyrights and patent rights. Except as expressly stated in the immediately //
//     preceding sentence, the Original Work is provided under this License on an "AS IS" BASIS and WITHOUT //
//     WARRANTY, either express or implied, including, without limitation, the warranties of             //
//     non-infringement, merchantability or fitness for a particular purpose. THE ENTIRE RISK AS TO THE  //
//     QUALITY OF THE ORIGINAL WORK IS WITH YOU. This DISCLAIMER OF WARRANTY constitutes an essential part //
//     of this License. No license to the Original Work is granted by this License except under this     //
//     disclaimer.                                                                                       //
//                                                                                                       //
//     Limitation of Liability. Under no circumstances and under no legal theory, whether in tort        //
//     (including negligence), contract, or otherwise, shall the Licensor be liable to anyone for any    //
//     indirect, special, incidental, or consequential damages of any character arising as a result of   //
//     this License or the use of the Original Work including, without limitation, damages for loss of   //
//     goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages  //
//     or losses. This limitation of liability shall not apply to the extent applicable law prohibits such //
//     limitation.

 Note: Some lines might run beyond the end of the line in the following listing; in such a case, a carriage return (i.e., asci chars
13, 10 should not be included.

 */
```

```
#include "myF64.h"

static char TemporaryMemory[128] = {0};

char* f_uint32toa(uint32_t value, uint8_t base);

static void f_split64(float64_t* x, uint8_t* f_sign, int16_t* f_ex, uint64_t *frac, uint8_t lshift)
{       // Returns * frac = 0 if x is zero or + /-INF.  Returns * frac != 0 if x is NaN.
        // Otherwise,  *frac has the leading 1 bit of the mantissa added, which is not present in *x where an implicit 1 applies
        *frac = (*x) & 0xfffffffffffff;
        if(0==(*f_ex  =  (((*x)>>52) & 2047)))
        *frac=0;
        else if(2047!=(*f_ex))
        *frac |= 0x10000000000000;
        *frac <<= lshift;
        *f_sign = ((*x)>>63) & 1;  }

static void f_split_to_fixpoint(float64_t *x, uint8_t * f_sign, int16_t * f_ex, uint64_t *frac, int16_t point_bitnr)
{       // Returns *frac = 0 if x is 0
        f_split64(x, f_sign, f_ex, frac, 0);
        if(0!=(*f_ex) && 2047!=(*f_ex))
        {       point_bitnr=(1023+52)-(*f_ex)-point_bitnr;
                if(point_bitnr<=-64 || point_bitnr>=64) *frac=0;
                else if(point_bitnr<0) *frac<<=-point_bitnr;
                else *frac>>=point_bitnr;  }  }

static void f_combi_from_fixpoint(float64_t * x, uint8_t f_sign, int16_t f_ex, uint64_t *frac)
{       // Warning: NaN can not be created with this function.
        // If * frac == 0 passed it generates zero if f_ex is <2047, otherwise + /-INF.
        uint8_t round=0;
        uint64_t w=*frac;
        if(0!=w)
        {
                while(0==(w & 0xffffe00000000000))    { w<<=8; f_ex-=8; }
                while(0==(w & 0xfff0000000000000))  { w<<=1; --f_ex;  }
                while(0!=(w & 0xff00000000000000))  { round = 0!=(w&(1<<3)); w>>=4; f_ex+=4; }
                while(0!=(w & 0xffe0000000000000))  { round = 0!=(w&1); w>>=1; ++f_ex; }
                if(round)
                {       ++w;
                        if(0!=(w & 0xffe0000000000000))
                        { w>>=1; ++f_ex; }  }
                if(f_ex<=0)                             // If f_ex == 0: Does not support the unnormalized numbers except zero
                  { f_ex=0; w=0; }                      // +0 oder -0  -0 or 0
        }
        else if(f_ex<2047) f_ex=0;
        if(f_ex>=2047) { f_ex=2047; w=0; }      // +INF oder -INF
        *((uint64_t*)x)=(((uint64_t)f_sign)<<63) | (((uint64_t)f_ex)<<52) | (w & 0xfffffffffffff);
}

static int8_t f_shift_left_until_bit63_set(uint64_t *w)
{       // If * w = 0 if the bit with the number 63 or not
        // Repeatedly left shift of w * can be set, * w = 0, and 64 returned.
        // Otherwise, w * as often shifted to the left until the bit with the number 63
        // Is set. The number of links made shifts is returned.
        register int8_t count=0;
        register uint64_t mask;
```

```
    for(mask=((uint64_t)255LU)<<(63-7); 0==(mask & (*w)) && count<64; count+=8) (*w) <<= 8;
    for(mask=((uint64_t)1LU)<<63; 0==(mask & (*w)) && count<64; ++count) (*w) <<= 1;
    return count;  }

static uint64_t approx_high_uint64_word_of_uint64_mult_uint64(uint64_t *x, uint64_t *y, uint8_t signed_mult)
{   // Computes an approximation of floor (x * y / (2 to 64)). If signed_mult == 0 is passed
    // the value returned is never greater and a maximum of 3 smaller than the actual value.
    // 0 != signed_mult & 1: *x is signed     0 != signed_mult & 2: *y is signed
    uint64_t r=((*x)>>32)*((*y)>>32) + ((((*x)>>32)*((*y)&0xffffffff))>>32) + ((((*y)>>32)*((*x)&0xffffffff))>>32);
    if(0!=(signed_mult & 1) && ((int64_t)(*x))<0)   r -= (*y);
    if(0!=(signed_mult & 2) && ((int64_t)(*y))<0)   r -= (*x);
    return r; }

static uint64_t approx_high_uint64_word_of_uint64_mult_uint64_pbv_y(uint64_t *x, uint64_t y, uint8_t signed_mult)
    {   return approx_high_uint64_word_of_uint64_mult_uint64(x, &y, signed_mult);  }

static uint64_t approx_inverse_of_fixpoint_uint64(uint64_t *y)
{   // It must be 0 != *y & 0x8000000000000000!
    // Under this condition (2 ^ 126) / * y is calculated and, if necessary, rounded to the next integer.
    uint64_t zL=0, dH=*y, dL=0, aH=0x4000000000000000, aL=0;
    uint8_t i=65;
    while(1)
    {
        dL >>= 1;
        if(dH & 1) dL |= ((uint64_t)1LU) << 63;
        dH >>= 1;
        if(0==--i) break;
        zL<<=1;
        if(aH>dH || (aH==dH && aL>=dL))
          {  zL |= 1;      aH -= dH;
             if(dL > aL)  --aH;
             aL -= dL;  }
    }
    if(aL>=dL) ++zL;
    return zL;
}

// convert uint32_t (aka long) to float61
float64_t f_long_to_float64(long n)
{   float64_t r;
    uint64_t w=((uint64_t)(n<0 ? -n : n))<<20;
    f_combi_from_fixpoint(&r, n<0 ? 1 : 0, 1023+32, &w);
    return r;   }

float64_t f_abs(float64_t x)
{   return x & 0x7fffffffffffffff;     }

//converts a float32 to float64
/***********************************************************/
float64_t f_sd(float32_t fx)
/***********************************************************/
{   uint32_t* i;
    uint8_t  f_sign;
    int16_t f_ex;
    uint64_t  w;
    float64_t f64;

    //For (uint32_t)(*(uint32_t*) &x), the (uint32_t*) x is a cast to treat the expression x as if it were a unit32_t
```

```
    //The * at the beginning (dereferencing operator) accesses memory through a pointer.

    i = (uint32_t*)&fx;                                   //original had i=&fx;
    w = ((*i) & 0x7fffffl);
    f_ex  =  (*i>>23)&0xff;
    f_sign = (*i>>31)&1;

    if(0==f_ex && 0!=w) f_ex+=29+0x3ff-0x7e;              // Denormalized float (32 bits) Number
    else if(255==f_ex)                                    // +/-INF oder NaN
      return 0==w ? (f_sign ? float64_MINUS_INFINITY : float64_PLUS_INFINITY) :
float64_ONE_POSSIBLE_NAN_REPRESENTATION;
    else
    {    w |= 0x800000;     // If NO denormalized float number (32 bits) is present, the implicit leading 1 bit is added to w
         if(f_ex) f_ex += 29+0x3ff-0x7f;  }   // For ==> FLOAT (32 bits) <== true: NOT ALL FLOAT denormalized numbers
                                              // are interpreted as zero.
    f_combi_from_fixpoint(&f64, f_sign, f_ex, &w);
    return(f64);  }


/**********************************************************/
float32_t f_ds(float64_t fx)
/**********************************************************/
{   // converts float64 to float32 when possible
    uint8_t  f_sign;
    int16_t f_ex;
    uint32_t ui32;
    float32_t f32;
    uint64_t w;

    f_split64(&fx, &f_sign, &f_ex, &w, 0);   //f_split64(float64_t* x, uint8_t* f_sign, int16_t* f_ex, uint64_t *frac, uint8_t lshift)

    if(f_ex >= 1023-149)
    {    if(f_ex>1023+127) // +/-INF oder NaN  +/- INF or NaN
         {    if(f_ex==2047 && 0!=w) ui32=0xfffff;// NaN
              else ui32=0;
              f_ex=255;   }
         else
         {    ui32=(w>>(52-23)) & 0x7fffff;       // Es ist 0x10000000000000 | w == w  It is 0x10000000000000 | w == w
              if(f_ex<1023-126)                   // It creates a denormalized float number (32 bits).
              {    ui32 = (ui32 | 0x800000) >> (1023-126-f_ex);  f_ex=0;   }
              else  f_ex=(f_ex-0x3ff+0x7f) & 0xff;  }
    }
    else ui32=0;                                  // All denormalized float64 numbers are interpreted as zero.

    ui32 |= ((uint32_t)f_sign<<31)|((uint32_t)f_ex<<23);
    f32= *((float32_t *)&ui32);

    return(f32);  }

/**********************************************************/
static void f_addsub2(float64_t* x, float64_t a, float64_t b, uint8_t flagadd, uint8_t* flagexd )
/**********************************************************/
{ // add positive doubles
    uint8_t sig;
    int16_t aex,bex;
    uint64_t wa, wb;

    f_split64(&a,&sig,&aex,&wa, 10);
    f_split64(&b,&sig,&bex,&wb, 10);
```

```
    *flagexd=0;

    //if only NAN
    if(2047==aex || 2047==bex)
    {    *x=float64_ONE_POSSIBLE_NAN_REPRESENTATION; return;  }

         if(!aex || aex+64<=bex) {*x=b; *flagexd=1; return;} // All denormalized numbers are interpreted as zero.
         if(!bex || bex+64<=aex) {*x=a; return;} // All denormalized numbers are interpreted as zero.

         if(flagadd)
         { if(aex>=bex) wa+=wb>>(aex-bex); // aex-bex <64 has already been through above test if (bex! || bex+64 <= aex) excluded
           else
          {wa=wb+(wa>>(bex-aex)); aex=bex;  }  } // aex-bex <64 has already been through above test if (bex! || bex+64 <= aex) excluded
         else
         { if(aex>bex || (aex==bex && wa>=wb)) wa-=wb>>(aex-bex); //  aex-bex <64 has already …  test if (bex! || bex+64 <= aex) excluded
              else
              {    wa=wb-(wa>>(bex-aex)); *flagexd=1;aex=bex;}  } // aex-bex <64 has already …  test if (bex! | | bex+64 <= aex) excluded
         f_combi_from_fixpoint(x, 0, aex-10, &wa);   }

/***********************************************************/
static void f_setsign(float64_t* x, int8_t sign)
/***********************************************************/
{    if(sign) *x |= 0x8000000000000000;
     else *x &= 0x7fffffffffffffff;  }

/***********************************************************/
static uint8_t f_getsign(float64_t x)
/***********************************************************/
{ uint64_t* px = &x;
     return ((uint8_t)(((*px)>>63)&1));  }

/***********************************************************/
float64_t f_add(float64_t a, float64_t b)
/***********************************************************/
{    uint8_t signa,signb,signerg;
     uint8_t   flagexd;
     uint64_t  i64;
     float64_t* x = &i64;

     signa= f_getsign(a);
     signb= f_getsign(b);
     if(signa^signb)                                          // if different else equal signs
      {  f_addsub2(x,a,b,0,&flagexd); signerg= ((flagexd^signa))&1; }  // calc a-b
     else { f_addsub2(x,a,b,1,&flagexd); signerg= signa; }             // calc a+b

     f_setsign(x,signerg);

     return(i64);  }

/***********************************************************/
float64_t f_sub(float64_t a, float64_t b)
/***********************************************************/
{    uint8_t signb;
     float64_t  bloc=b;
     uint64_t  i64;

     signb= f_getsign(bloc);
     signb ^= 1;
```

```
    f_setsign(&bloc,signb);
    i64=f_add(a,bloc);
    return(i64);  }

/**********************************************************/
float64_t f_mult(float64_t fa, float64_t fb)
/**********************************************************/
{ //multiply doubles
    uint8_t  asig,bsig;
    int16_t aex,bex;
    uint64_t am, bm;

    f_split64(&fa,&asig,&aex,&am, 11);
    f_split64(&fb,&bsig,&bex,&bm, 11);

    if(2047==aex || 2047==bex)  return float64_ONE_POSSIBLE_NAN_REPRESENTATION;

    else if(!aex || !bex)  // All denormalized numbers are interpreted as zero.
        { return float64_NUMBER_PLUS_ZERO;  }
    else { aex=aex+bex-(0x3ff+10); am=approx_high_uint64_word_of_uint64_mult_uint64(&am, &bm, 0); }
    asig ^= bsig;
    f_combi_from_fixpoint(&fa, asig, aex, &am);        // This must be subtracted from aex  so
                                                       //aex == 2047 (INF, NaN) is not distorted
    return fa;   }

/**********************************************************/
float64_t f_div(float64_t x, float64_t y)
/**********************************************************/
{   // Divides x/y
    uint8_t  xsig, ysig;
    int16_t xex, yex;
    uint64_t xm, ym, i64;

    f_split64(&x,&xsig,&xex,&xm, 11);
    f_split64(&y,&ysig,&yex,&ym, 11);

    if(2047==xex || 2047==yex || 0==yex) return float64_ONE_POSSIBLE_NAN_REPRESENTATION;
    else
    {     i64=approx_high_uint64_word_of_uint64_mult_uint64_pbv_y(&xm, approx_inverse_of_fixpoint_uint64(&ym), 0);
        xex += 1023-yex;  }
    f_combi_from_fixpoint(&x, xsig^ysig, xex-10, &i64);
    return x;   }

float64_t f_cut_noninteger_fraction(float64_t x)
{     int16_t f_ex=(((x)>>52) & 2047);
    if(0==f_ex || f_ex>=1023+52) return x;          // All denormalized numbers interpreted as zero. includes the cases x = NaN, x = + -INF.
    if(f_ex<1023) return float64_NUMBER_PLUS_ZERO;
    return x & (0xffffffffffffffff << ((1023+52)-f_ex));  }

long f_float64_to_long(float64_t x)
{     // When f_abs(x) is greater than 1LU << 31, zero is returned.
    uint8_t f_sign;
    int16_t f_ex;
    uint64_t w;
    f_split_to_fixpoint(&x, &f_sign, &f_ex, &w, 0);
    //for next line: x= +/-INF and x = NaN covered (delivery of 0). In the case of x = 0, w = 0
    return (f_ex>=1023+32) ? 0 : (f_sign ? -((long)w) : ((long)w)); }
```

```
int8_t f_isnan(float64_t x)
{	// Returns nonzero if x is an IEEE 754 "Not A Number" and otherwise zero.
	return 0x7ff0000000000000 == (((uint64_t)x) & 0x7ff0000000000000) && 0!=(((uint64_t)x) & 0xfffffffffffff);  }

int8_t f_compare(float64_t x, float64_t y)
{	// If both x and y represent real numbers
	// (or +/-INF if F_ONLY_NAN_NO_INFINITY is not defined) f_compare returns
	// zero if x is equal to y, positive nonzero if x > y and negative nonzero if x < y.
	// If x or y are NaN, f_compare returns zero.
	uint8_t  asig,bsig;
	int16_t xex,yex;
	uint64_t wx, wy;
	f_split64(&x,&asig,&xex,&wx, 0);
	f_split64(&y,&bsig,&yex,&wy, 0);

	if(2047==xex || 2047==yex) return 0;

	if(0==xex) return (0==yex && 0==wy) ? 0 : (bsig ? 1 : -1);  // All denomalised numbers will be interpreted as 0

	if(0==yex || asig!=bsig || xex>yex) return asig ? -1 : 1;  // denomalised numbers will be interpreted as 0
	if(xex<yex) return asig ? 1 : -1;
	return wx==wy ? 0 : ((wx>wy && !asig) || (wx<wy && asig) ? 1 : -1);   }

static float64_t f_mod_intern(float64_t x, uint8_t ysig, int16_t yex, uint64_t *ymts, float64_t *ganz)
{	uint8_t xsig, count;
	int16_t xex, zex;
	uint64_t xm;
	float64_t g;
	uint64_t q;

	f_split64(&x,&xsig,&xex,&xm, 11);
	if(ganz) *ganz=float64_NUMBER_PLUS_ZERO;
	if(0==xex)	return float64_NUMBER_PLUS_ZERO;
	if(2047==xex || 2047==yex || 0==(yex|(*ymts)))
	{	if(ganz) *ganz=float64_ONE_POSSIBLE_NAN_REPRESENTATION;
		return float64_ONE_POSSIBLE_NAN_REPRESENTATION; }

	for(count=10 ; 0!=(xex|xm) && (xex>yex || (xex==yex && xm>=(*ymts))) && 0!=--count ; )
	{	// In many cases, this loop is executed zero or one time.
		q=approx_high_uint64_word_of_uint64_mult_uint64_pbv_y(&xm, (approx_inverse_of_fixpoint_uint64(ymts)-5)<<1, 0)-5;
		// approx_inverse_of_fixpoint_uint64(&ym)-5 is not greater than the actual value (without rounding error).
		// Außerdem ist (2 hoch 62) <= approx_inverse_of_fixpoint_uint64(&ym)-5 < (2 hoch 63).
		// Moreover, (2 ^ 62) <= approx_inverse_of_fixpoint_uint64 (&ym) -5 < (2 ^ 63).
		//  Rounding error and q corresponds to the number 1 for following line:
		if(xex==yex && 0==(0x8000000000000000 & q)) q=0x8000000000000000;
		zex=xex-yex;
		if(zex<=63) q &= (0xffffffffffffffff << (63-zex));
		xm -= approx_high_uint64_word_of_uint64_mult_uint64(&q, ymts, 0)<<1;
		if(ganz)
		{	f_combi_from_fixpoint(&g, xsig^ysig, (1023-11)+zex, &q);  *ganz=f_add(*ganz, g);  }

		xex-=f_shift_left_until_bit63_set(&xm);	}

	f_combi_from_fixpoint(&x, xsig, xex-11, &xm);
	*ymts=xm;
	// No exact calculation takes place. Make sure return value is from 0 up to but not y:
	if(0==count) return float64_NUMBER_PLUS_ZERO;
	return x;  }
```

```
float64_t f_mod(float64_t x, float64_t y, float64_t *ganz)
{       // The fmod function calculates the floating-point remainder f of x / y such that x = i * y + f,
        // where i is an integer, f has the same sign as x, and the absolute value of f is less than the absolute value of y.
        uint8_t ysig;
        int16_t yex;
        uint64_t ym;
        f_split64(&y,&ysig,&yex,&ym, 11);
        return f_mod_intern(x, ysig, yex, &ym, ganz);  }

static int16_t f_10HochN(int64_t n, uint64_t *res)
{       uint64_t pot=((uint64_t)10)<<60;
        int16_t exp2=0, pot_exp2=3;
        uint8_t neg=0;
        *res=((uint64_t)1)<<63;
        if(n<0) { neg=1; n=-n; }
        while(0 != n)
        {       if(0 != (n & 1))
                {       *res = approx_high_uint64_word_of_uint64_mult_uint64(res, &pot, 0);
                        exp2+=pot_exp2+1-f_shift_left_until_bit63_set(res);  }
                pot = approx_high_uint64_word_of_uint64_mult_uint64(&pot, &pot, 0);
                pot_exp2=(pot_exp2<<1)+1-f_shift_left_until_bit63_set(&pot);
                n >>= 1;   }
        if(neg)
        {       *res = approx_inverse_of_fixpoint_uint64(res);  exp2=-exp2-f_shift_left_until_bit63_set(res);  }

        return exp2;  }

// f_to_decimalExp() converts the float64 to the decimal representation of the number x
char* f_to_decimalExp(float64_t x, uint8_t anz_dezimal_mantisse, uint8_t MantisseUndExponentGetrennt, int16_t *ExponentBasis10)
{
        // if x is a real number or to the strings "+INF", "-INF", "NaN". If x is real, f_to_decimalExp() generates
        // a mantisse-exponent decimal representation of x using anz_dezimal_mantisse decimal digits for
        // the mantisse. If MantisseUndExponentGetrennt!=0 is passed f_to_decimalExp() will generate different
        // strings for the mantisse and the exponent. If you assign   char *str=f_to_decimalExp(x, anz_mts, 1, 0)
        // then str points to the mantisse string and str+strlen(str) points to the exponent string.  If the pointer
        // ExponentBasis10 passed to f_to_decimalExp() is nonzero, the function will assign the 10-exponent to
        // *ExponentBasis10 ; e.g. if the decimal representation of x is 1.234E58 then the integer 58 is assigned to
        // *ExponentBasis10.

        uint8_t f_sign;
        uint8_t len, posm, i;
        int16_t f_ex;
        uint64_t w, w2;
        int16_t Exp10;

        if(anz_dezimal_mantisse>17) anz_dezimal_mantisse=17;
        if(anz_dezimal_mantisse<1)   anz_dezimal_mantisse=1;
        f_split64(&x, &f_sign, &f_ex, &w, 11);
        // All denormalised numbers are treated as 0:
        if(0==f_ex) { TemporaryMemory[0]='0'; TemporaryMemory[1]=0; return TemporaryMemory; }
        if(2047==f_ex) { strcpy(TemporaryMemory, "NaN"); return TemporaryMemory; }

        f_ex-=1023;                                          // After the test  0 == f_ex and  2047 ==  f_ex !
        len=0;
        if(f_sign) TemporaryMemory[len++]='-';

        if(f_ex >= 0) Exp10=(uint16_t)((((uint16_t)f_ex)*10+31)>>5);
        else Exp10=(int16_t)(-((((uint16_t)(-f_ex))*9)>>5));
```

```
    f_ex+=f_10HochN(-Exp10, &w2);
    w=approx_high_uint64_word_of_uint64_mult_uint64(&w, &w2, 0);
    f_ex+=1-f_shift_left_until_bit63_set(&w);

    while(f_ex<0)
    {   w=approx_high_uint64_word_of_uint64_mult_uint64_pbv_y(&w, ((uint64_t)10)<<60, 0);
        f_ex+=4-f_shift_left_until_bit63_set(&w);
        Exp10--;  }

    while(f_ex>=4 || (f_ex==3 && (w & 0xf000000000000000)>=0xa000000000000000))
    {   w=approx_high_uint64_word_of_uint64_mult_uint64_pbv_y(&w, 0xcccccccccccccccc, 0);
        f_ex-=3+f_shift_left_until_bit63_set(&w);
        Exp10++;  }

    posm=len;
    ++len;
    while(0!=(anz_dezimal_mantisse--))
    {   TemporaryMemory[len]='0';
        if(f_ex>=0)
        {   TemporaryMemory[len] += (w>>(63-f_ex));  w <<= 1+f_ex; f_ex=-1;  }
        ++len;
        w=approx_high_uint64_word_of_uint64_mult_uint64_pbv_y(&w, ((uint64_t)10)<<60, 0);
        f_ex+=4-f_shift_left_until_bit63_set(&w);   }

    if(f_ex>=0 && (w>>(63-f_ex)) >= 5)
    {   for( i=len ; --i>posm ; )
        if(TemporaryMemory[i]=='9') TemporaryMemory[i]='0';
        else { ++TemporaryMemory[i]; break;  }
        if(i==posm)
            {   ++Exp10;
                TemporaryMemory[++i]='1';
                while(++i<len) TemporaryMemory[i]='0';  }    }

    TemporaryMemory[posm]=TemporaryMemory[posm+1];
    TemporaryMemory[posm+1]='.';
    if(MantisseUndExponentGetrennt) TemporaryMemory[len++]=0;
    TemporaryMemory[len++]='E';
    if(Exp10>0) TemporaryMemory[len++]='+';
    itoa(Exp10, &TemporaryMemory[len], 10);
    if(0!=ExponentBasis10)   *ExponentBasis10=Exp10;
    return TemporaryMemory;   }

//converts float64 number to string with number chars = max_nr_chars. max_leading_mantissa_zeros = 0 works ok
char* f_to_string(float64_t x, uint8_t max_nr_chars, uint8_t max_leading_mantisse_zeros)
{   int16_t exp10;
    // the set number to be calculated decimal mantissa:
    int8_t nrd=(0!=(x & 0x8000000000000000)) ? (max_nr_chars-1) : max_nr_chars;
    int8_t nrd_vor=nrd+1;
    char* r=0;
    uint8_t j, k;
    for(j=0; j<3 && nrd!=nrd_vor; j++)
    {   if(nrd>nrd_vor) { r=f_to_decimalExp(x, nrd_vor, 1, &exp10); break; }          // Calculate Only Exp10
        r = f_to_decimalExp(x, nrd, 1, &exp10); // Calculate Only Exp10
        if(((x>>52)&2047)==2047 || 0==(x & 0x7ff0000000000000))  return r;
        if(exp10<-max_leading_mantisse_zeros-1) break;
        nrd_vor=nrd;

        nrd=f_getsign(x) ? max_nr_chars-1 : max_nr_chars;
```

```
        // These  variables must be initialized both here and above before the start of the loop.
        if(exp10<0) nrd += exp10-1;                    // Exp10-1 character consumed by 0:00 .....
        else if(exp10+1<nrd) --nrd;  }                 // && exp10>=0 .
        // A sign by a decimal point consumes.
        // Example for the case Exp10 == max_nr_chars-2: It is max_nr_chars = 4, Exp10 = 2, Number = 683.79426
        // ==> It is shown 684
        // (3 instead of 4 characters is because would occupy 683.8 - too many characters).

    //Representation without 10-exponent (E):
    if(0!=(x & 0x7ff0000000000000) && exp10>=-max_leading_mantisse_zeros && nrd>exp10)
    {   if(f_getsign(x)) ++r;                          // sign
        if(exp10<0)
        {   for(j=strlen(r) ; (j--)>0 && '0'==r[j] ; )     r[j]=0;
            r[1]=r[0];
            for(++j ; j>0 ; j--) r[j-exp10]=r[j];
            r[0]='0';
            r[1]='.';
            for(j=2; ++exp10<0; ) r[j++]='0';  }
        else
        {   for(j=1 ; j<=exp10 ; j++)
            if(0==(r[j]=r[j+1]))
                {   while(j<=exp10) r[j++]='0';
                    r[j]=0;  break;  }

            if(j+1<max_nr_chars && 0!=r[j])
                {   r[j]='.';
                    for(j=strlen(r) ; (j--)>0 && '0'==r[j] ; ) r[j]=0;
                    if('.'==r[j]) r[j]=0;  }
            else  r[j]=0; // j+1>=max_nr_chars  Necessary for the case j +1> = max_nr_chars
        }
        if(f_getsign(x)) --r;
    }
    else  // Representation with 10-exponent (E)
    {   j=max_nr_chars-4;
        if(f_getsign(x))  --j;
        exp10=(x>>52)&2047; // Exp10 here is for receiving the binary exponent "abused"
        if(exp10<1023) --j;
        if(exp10>1023+34 || exp10<1023-34) --j;
        if(j<1) j=1;

        while((r=f_to_decimalExp(x, j, 0, 0)), strlen(r)>(uint8_t)max_nr_chars)
            if(--j<1) break;
        for(j=2; 0!=r[j] && 'E'!=r[j] && 'e'!=r[j] ; j++) ;
        k=j;
        while((--j>=4 || (!f_getsign(x) && j>=3)) && '0'==r[j]) ;
        while(0!=(r[++j]=r[k++])) ;  }
    return r;   }

//converts float to hex string. If allowed to round, it adds 0.5 for rounding purposes.
// The routine cuts off the fraction.
char* f_ftoHexIntStr(float64_t x, bool flagAllowRounding)
{   if (flagAllowRounding)
    {float64_t FiveTenthsF64 = f_long_to_float64(5);
        float64_t TenF64 = f_long_to_float64(10);
        FiveTenthsF64 = f_div(FiveTenthsF64,TenF64);
    x = f_add(x, FiveTenthsF64); }                           //add 0.5 for rounding purposes in next step

    uint32_t xU32 = f_float64_to_long(x);                    // cuts off fraction
```

```
return f_uint32toa( xU32, 16 );    }

// string to Float64. Returns pointer to True if wrong number digits
// max number digits = 15 including dp
// use instead: float64_t f_strtod(char *str, char **endptr);
float64_t f_strtoF64(char* str) //, bool* pError)
{       //examples: 50255055.123456, 50255055. , 50255055 , 0.123456 , .123456, 50255055.
        //must check for max number digits

        #define MaxNumDigits 15
        int len = strlen(str);
        //if (len == 0 || len > MaxNumDigits) { *pError = true; return 0; }

        float64_t valueF64H = 0;
        float64_t valueF64L = 0;

        char Dig[MaxNumDigits+1] = {0};         //add 1 to hold string terminator of 0; holds 50255055+0
        char* pD = Dig;                         //set pointer to starting location for next digit
        char* ps = str;                         // pointer at start of string
        char* pDecPnt = strchr(str,'.');        //returns pointer for dp and zero if no dp

        if (pDecPnt == 0)                                   // first get the integer part
        { while( *ps != 0 )  { *pD++ = *ps++; } }           //populate H; ps stops at end
        else
        { if (pDecPnt == ps) { Dig[0] = '0'; }              // covers case of ".123 "; ps stops at dp
        else {  while ( *ps != '.' )  { *pD++ = *ps++; } } }  //populate Dig; ps stops at dp
        *pD = 0; //adds terminator to the string

        // need some float64 values to multiply into the digits dig[]
        float64_t tenF64 = f_long_to_float64(10);           // float64 for 10 (ten)
        float64_t OneF64 = f_long_to_float64(1);            // float64 for 1 (one)
        float64_t OneTenthF64 = f_div(OneF64,tenF64);  // float64 for 0.1 (one tenth)

        float64_t digValF64;                                // temporary float64 value of one of the digits in Dig[]

        // Calculate f64 for the integer part: sequentially multiply the value in value64H
        // and add next digValF64: cycles as 5 => 50 + 0 = 50 => 500+2 => 502 => 5020 + 5 ...
        len = strlen(Dig);
        for( int i = 0; i < len; i++ )
        {       digValF64 = f_long_to_float64( Dig[i] - 0x30 );        // convert character to number between 0 and 9
                valueF64H = f_mult( valueF64H, tenF64 );               // shift value left by factor of 10
        valueF64H = f_add( valueF64H, digValF64 );  }                  // add the value of the digit

        // reset the various arrays and values
        for(int i = 0; i < MaxNumDigits+1; i++) Dig[i] = 0;            // place str terminates in receptacle array
        pD = Dig;                                                      //reset pointer

        //now get fraction part
        if (pDecPnt == 0)  {;}                              //no dp: do nothing since only needed the int part; ps still at end
        else                                                //ps pointer will be pointing to dp
        {       ps++;                                       //skip over dp
                while( *ps != 0 )  { *pD++ = *ps++; }    }  // populate Dig[] with fraction part, no dp
                *pD = 0; // add terminator

                // Calculate f64 for the fract part
                len = strlen(Dig);
                for( int i = 0; i < len; i++ )
```

```
        {     digValF64 = f_long_to_float64( Dig[len-1-i] - 0x30 );    // convert character to number between 0 and 9
              valueF64L = f_add( valueF64L, digValF64 );                // add the value of the digit
        valueF64L = f_mult( valueF64L, OneTenthF64 );   }               // shift value right by factor of 10
    return f_add(valueF64H,valueF64L);   }

/* converts uint32_t to string.
    value: input uint32_t, *result: output char array
    resStartIndex: index of starting char in result array
    base: base 2 through 32
*/
char* f_uint32toa(uint32_t value, uint8_t base)
{ // check that the base is valid
 if (base < 2 || base > 36) { *TemporaryMemory = '\0'; return TemporaryMemory; }

 char* ptr = TemporaryMemory;
 char* ptr1 = TemporaryMemory;
 char tmp_char;
 int32_t tmp_value;

 do
{ tmp_value = value;
  value /= (uint32_t)base ;
 *ptr++ = "ZYXWVUTSRQPONMLKJIHGFEDCBA9876543210123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"[35 + (tmp_value - value * base)];
} while ( value );

    *ptr-- = '\0';
    while (ptr1 < ptr)
    {     tmp_char = *ptr;
          *ptr--= *ptr1;
          *ptr1++ = tmp_char;  }

    return TemporaryMemory;
}
```

Section A6.10: StrNum.h

```
#ifndef STRNUM_H_
#define STRNUM_H_

char* uint32toa(uint32_t value, char* result, uint8_t base);
char* uint64toa(uint64_t value, char* result, uint8_t base);

#endif /* STRNUM_H_ */
```

Section A6.11: StrNum.cpp

```
#include <avr/io.h>
#include "StrNum.h"

char* uint64toa(uint64_t value, char* result, uint8_t base)
{     // check that the base is valid
if (base < 2 || base > 36) { *result = '\0'; return result; }

char* ptr = result;
char* ptr1 = result;
char tmp_char;
```

```
int64_t tmp_value;

do
{ tmp_value = value;
 value /= base ;
*ptr++ = "ZYXWVUTSRQPONMLKJIHGFEDCBA9876543210123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"[35 + (tmp_value - value * base)];
} while ( value );

*ptr-- = '\0';
while (ptr1 < ptr)
{       tmp_char = *ptr;
        *ptr--= *ptr1;
        *ptr1++ = tmp_char;  }

return result;
}

/* converts uint32_t to string.
        value: input uint32_t, *result: output char array
        resStartIndex: index of starting char in result array
        base: base 2 through 32
*/
char* uint32toa(uint32_t value, char* result, uint8_t base)
{       // check that the base is valid
if (base < 2 || base > 36) { *result = '\0'; return result; }

char* ptr = result;
char* ptr1 = result;
char tmp_char;
int32_t tmp_value;

do
{ tmp_value = value;
value /= (uint32_t) base ;
*ptr++ = "ZYXWVUTSRQPONMLKJIHGFEDCBA9876543210123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"[35 + (tmp_value - value * base)];
} while ( value );

*ptr-- = '\0';
while (ptr1 < ptr)
{       tmp_char = *ptr;
        *ptr--= *ptr1;
        *ptr1++ = tmp_char;  }
        return result;
}
```

Section A6.12: TC16.h

```
#ifndef TC16_H_
#define TC16_H_


/*
NOTE: full functionality has not been verified except as used
                        by main.cpp in the present program

keep mS < 4000 without using a counter
mS = number milliseconds for timer. callFunction = true if interrupt
calls the function "TC16_calledFunction".
if mS > 4000, then enable the internal incremental Counting by setting
mSCounterIncr to a non-zero value. For times to the nearest 1 second,
use 1000 (1 sec increments), and for 4 sec increm use 4000. The counter
```

```
calls interrupt every msCounterIncr and increments count by 1 till
maxCount. The msCounterInc>0 case can be 10% long.
*/

void TC16_config(double mS, double fcpu, bool callFunction, bool setFlag, bool stopInISR, uint16_t msCounterIncr);

void TC16_Start(void);
void TC16_Stop(void);

bool TC16_isTimeExpired(void);
void TC16_resetExpiredFlag(bool flag_TimExpired);

void TC16_calledFunction(void);


#endif /* TC16_H_ */
```

Section A6.13: TC16.cpp

```
#include <avr/interrupt.h>
#include "TC16.h"

volatile bool flag_TimeExpired = false; //need this because of time ISR
volatile uint16_t count = 0;
uint16_t maxCount = 0;

bool flag_callFunction = false;
bool flag_setFlag = false;
bool flag_stopInISR = false;



/*
NOTE: full functionality has not been verified except as used
by main.cpp in the present program

keep mS < 4000 without using a counter
mS = number milliseconds for timer. callFunction = true if interrupt
calls the function "TC16_calledFunction".
if mS > 4000, then enable the internal incremental Counting by setting
mSCounterIncr to a non-zero value. For times to the nearest 1 second,
use 1000 (1 sec increments), and for 4 sec increm use 4000. The counter
calls interrupt every msCounterIncr and increments count by 1 till
maxCount. Setting TforOCB_FforOCA=T compares timer with OCR1B (for ADC trigger)
rather than OCR1A (for USART) and does not make use of msCounter.
*/

void TC16_config(double mS, double fcpu, bool callFunction, bool setFlag, bool stopInISR, uint16_t msCounterIncr)
{
	TCCR1B &= ~ ( (1 << CS12) | (1 << CS11) | (1 << CS10) );		// stop timer
	TIMSK1 &= ~(1 << OCIE1B);						// disable interrupt

	flag_callFunction = callFunction;					// Save booleans
	flag_setFlag = setFlag;
	flag_stopInISR = stopInISR;
```

```
    TCCR1A &= ~( 1<<COM1A1 | 1<<COM1A0 | 1<<COM1B1 | 1<<COM1B0 ); //normal port ops
    TCCR1A |= (0<<WGM11) | (0<<WGM10);                  //Needed as part of CTC Mode for top=OCR1A and ICR1
    //TCCR1B |= (0<<WGM13) | (1<<WGM12);                //needed as part of CTC mode for top=OCR1A
    TCCR1B |= (1<<WGM13) | (1<<WGM12);                  //needed as part of CTC mode for top=ICR1

    if (msCounterIncr < 1)                              //triggers every mS millisecs
        {  maxCount = 0;
           ICR1 = (uint16_t) (  (mS/1000.0) * (fcpu/1024.0) - 1  );  }
    else                                   //will trigger every msCounterIncr and adds 1 to count until MaxCount
        {    maxCount = (uint16_t) ( ( (float)mS + 0.1) / (float)msCounterIncr);
           ICR1 = (uint16_t) (  (msCounterIncr/1000.0) * (fcpu/1024.0) - 1  );  }

    TIMSK1 |= (1 << OCIE1B);
    sei();                                              // allow interrupts
}

//Starts timer but can also be used to reset timing since it has TCNT1=0
void TC16_Start(void)
{   flag_TimeExpired = false;
    count = 0;                                          //used for times longer than 4000mSec
    TCNT1 = 0;                                          //Start timer 1 at 0
  TCCR1B |= (1 << CS12) | (0 << CS11) | (1 << CS10); }

void TC16_Stop(void)
{   TCCR1B &= ~ ( (1 << CS12) | (1 << CS11) | (1 << CS10) );       //stop timer
    count = 0; }

bool TC16_isTimeExpired(void)
{ return flag_TimeExpired;  }

void TC16_resetExpiredFlag(bool flag_TimExpired)
{ flag_TimeExpired = flag_TimExpired; }

ISR ( _VECTOR(12)  )                          // can write TIMER1_COMPB_vect instead of _VECTOR(12)
{   count++; //PORTB ^= (1<<4);
    if (count > maxCount)
    {   if(flag_stopInISR) TC16_Stop();
        if(flag_setFlag) flag_TimeExpired = true;
        if(flag_callFunction) TC16_calledFunction();
        count = 0; }
}
```

Section A6.14: USART.h

```
#ifndef USART_H_
#define USART_H_

void USART_SendChar( uint8_t data );
bool USART_isBuffEmpty(void);
void USART_config( double Fcpu, double baud);
void USART_Send( uint8_t data );
void USART_SendStr(const char* str);
char USART_ReadBuffChar(void);
void USART_ClearBuffer(void);
bool USART_bufchr(const char c);

#endif /* USART_H_ */
```

Section A6.15: USART.cpp

```
#include <avr/io.h>
#include <math.h>
#include <string.h>
#include <avr/interrupt.h>
#include "USART.h"

#include "LCD16x2_ST7032.h"
#include "StrNum.h"

#define USART_BUFFLEN 50

// The word 'static' can be removed from next 4 lines but not 'volatile'
volatile static uint8_t USART_BufferLen = USART_BUFFLEN;
volatile char USART_Buffer[USART_BUFFLEN] = {0};

volatile static uint8_t USART_Next_Write_Loc = 0;
volatile static uint8_t USART_Next_Read_Loc = 0;

//initialize or reset baud
void USART_config( double Fcpu, double baud)
{
	//disable R and T
	UCSR0B &= ~( (1<<RXEN0)|(1<<TXEN0) ); //disable receiver and transmitter
	UCSR0B &= ~(1<<RXCIE0); //disable interrupt

	// tx=out, rx=in
	DDRD &= ~0b001;				//RX is input
	DDRD |= 0b010;				//TX is output

	// Set baud rate
	UBRR0 = floor( (uint16_t) ( ( Fcpu / ( 16 * baud ) ) - 1 ));

	UCSR0C |= (0b011<<UCSZ00);		// set 8data 1stop (already set for one stop bits)
	UCSR0B |= (1<<RXEN0)|(1<<TXEN0);	//Enable receiver and transmitter
	UCSR0B |= (1<<RXCIE0);			//enable interrupt

	sei();				//set global interrupt for USART; need: #include <avr/interrupt.h>

	USART_Next_Read_Loc = 0;	//usart rcv triggers interupt and saves in a circular buffer
	USART_Next_Write_Loc = 0;

	USART_ClearBuffer();
}

//returns true if char c in usart rcv buffer
bool USART_bufchr(const char c)
{	bool flag_itsThere = true;
	if( strchr(  (const char*)USART_Buffer, c) == 0  ) { flag_itsThere = false; }
	return flag_itsThere;  }

//sends byte (not necessarily a character)
void USART_SendChar( uint8_t data )
{	while ( !( UCSR0A & (1<<UDRE0)) );	// Wait for empty transmit register
	UDR0 = data; }				// Put data into register, sends the data
```

```
//FE5650A must include separate <cr>
void USART_SendStr(const char* str)
{           int L = strlen(str);
            for (int i = 0; i < L ; i++) { USART_SendChar(str[i]); }  }

//remember to add 0 terminator to end of string if needed
char USART_ReadBuffChar(void)
{           char Ch = 0;
            if(USART_Next_Read_Loc != USART_Next_Write_Loc)
            {           Ch = USART_Buffer[USART_Next_Read_Loc++];
                        if(USART_Next_Read_Loc == USART_BufferLen) {USART_Next_Read_Loc=0;}  }
            return Ch;  }

bool USART_isBuffEmpty(void)
{           return( USART_Next_Read_Loc == USART_Next_Write_Loc ); }          //returns true when not equal

void USART_ClearBuffer(void)
{  USART_Next_Read_Loc = USART_Next_Write_Loc = 0;                            //set all entries as string terminators
for(int i = 0; i < USART_BufferLen; i++) USART_Buffer[i] = 0; }

ISR(USART_RX_vect)
{           // the interrupt signals when a byte is received by the USART
            // store the data into the buffer
            USART_Buffer[USART_Next_Write_Loc++] = UDR0;
            if(USART_Next_Write_Loc == USART_BufferLen) {USART_Next_Write_Loc=0;}  }
```

Appendix 7: Allan Deviation Software

The appendix contains the essential Microsoft Visual Studio (MVS) demonstration code [A7.4] for calculating and plotting the Allan Deviation and Variance. The software allows the user to implement simple uniformly and normally distributed random numbers (either continuous or discrete) and view the results for the Allan Deviation on a plot. The software is meant for demonstration purposes and could provide the basis for the reader's own software endeavors. Here, we provide the basic overview of using the software and the codes listings in case installable software is not available. The listings can be copied into the Visual Studio (2017). For this purpose, the code for both Form1.vb.designer (i.e., the Graphical User Interface GUI) and Form1.vb (the code that animates the GUI) has been listed along with instruction for populating them. As discussed in Chapter 4 (see the references), a number of well tested programs with extensive functions can be found on the internet for free to little cost and would be the preferable go-to professional software for Allan Deviation calculation and visualization.

Section A7.1: Graphical User Interface (GUI) for AllanDev

This book includes access to a variety of software primarily meant to be modified by the reader. The included free Allan Deviation software provides some simulation capability and it can read text files with frequency points separated by CR and LF characters. The reader would be best advised to download some of the well-tested software listed in the Chapter 4 references [4.31-35]. We start by showing the user interface and basic functionality for the included software.

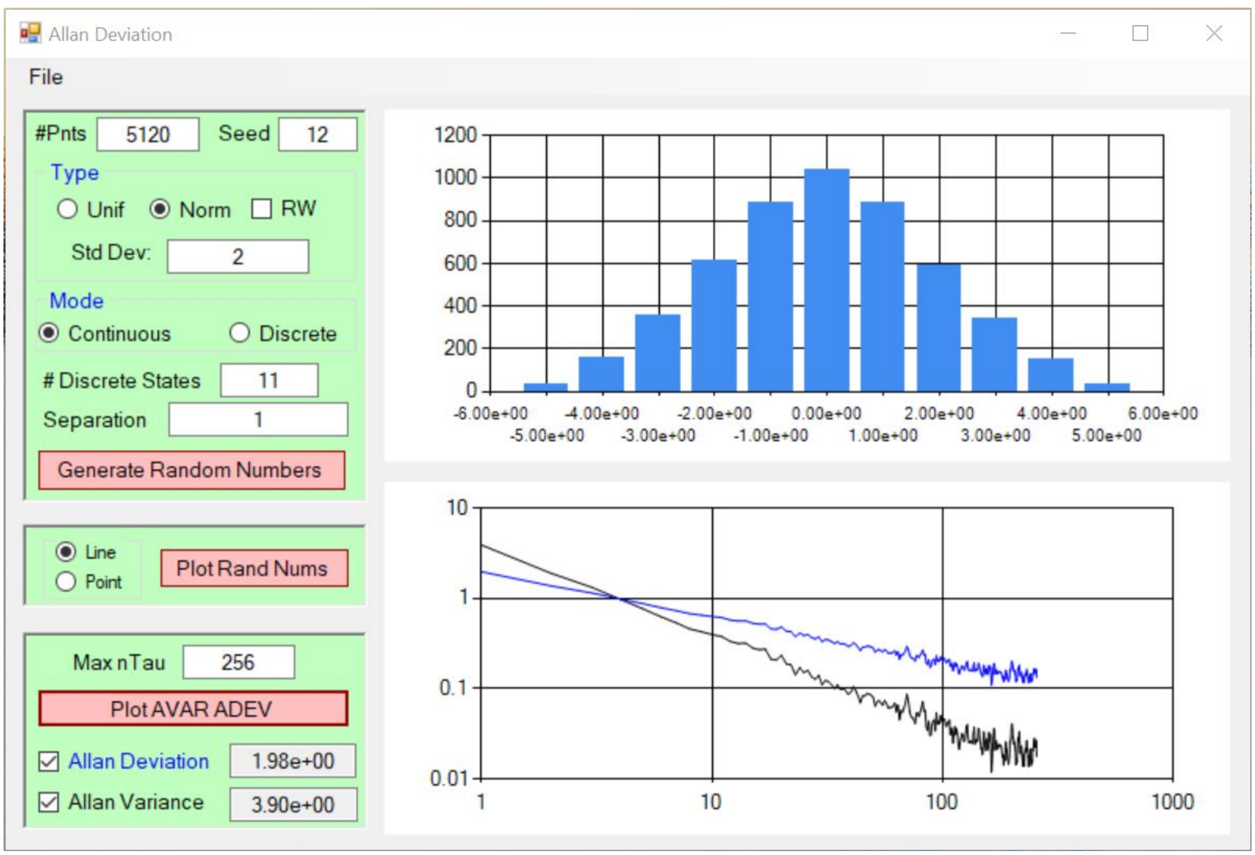


Figure A7.1: Graphical User Interface (GUI) for demo Allan Deviation software

The Allan Deviation software plots the Allan Deviation and Variance for various statistical distributions and conditions. Figure A7.1 shows the Graphical User Interface (GUI). The basic idea consists of generating a random process using either a uniform or a Normal/Gaussian distribution. A checkbox allows the program to form a random walk whereby each newly generated random number is added to the sum of all previously generated random numbers. The top graph renders either the distribution (such as the Normal distribution shown) upon clicking the button labelled 'Generate Random Numbers', or it renders the actual sequence of random numbers, which looks like noise, upon clicking the button labelled 'Plot Rand Nums'. The bottom graph shows the time-parameterized Allan Variance (black) or Allan Deviation (blue) by clicking the 'Plot AVAR/ADEV' button. The software can read a text file (.txt) containing a custom set of numbers (i.e., entered by hand) using the 'File' menu item. It is also possible for the software to save the software-generated random numbers to a '.txt' file using the 'File' menu item. Keep in mind, the software is only for demonstration. Well tested software should be used for 'the stuff that matters.'

Examine the three individual panels on the left side. Starting with the bottom panel, the button simply causes the program to use the generated random numbers to calculate the (time-parameterized) Allan Variance (black) and the Allan Deviation (blue) and to overwrite any existing plot in the bottom chart. The code associated with the bottom panel is the work-horse for the Allan computations. The middle panel plots the generated random numbers/process in the top chart. To use either of the bottom two panels, it's only necessary to generate the random numbers using the top panel or use the file menu to read numbers from a file. The bottom two panels do not depend on each other but they both depend on the numbers generated in the top panel or read from the external file.

The top panel links to the code for generating the random numbers. At present, the software can generate a uniform or Normal distribution of random numbers. Many good references can be easily found in a library or online. For a few typical examples, see references [A7.1-3].

The number of samples can be set in the text box labelled as '#Pnts'. It is possible to enter a seed for generating the random numbers; each set of numbers will be the same for the same seed. This makes it possible to repeat a set of numbers when 'something looks interesting'.

The inputs for 'Std. Dev.', '# Discrete States', and 'Separation' require some explanations.

1. The standard deviation (stdDev) refers to the standard deviation of either the normal or uniform distributions. One standard deviation on either side of the average (zero in these cases) encompass 68% of the values for the normal distribution and 58% for the uniform distribution.
2. The number of discrete states refers to both the discrete distributions and the histogram data visualization for the top chart. For the discrete distribution, the '# Discrete States' refers to the number of bins which the random number generator can select among. For example, for two discrete states with a separation of 1, the midpoints of the states/bins would be located at

   x=-0.5 and x=0.5

   Each random number generated would either be -0.5 or +0.5. For 3 discrete states separated by 1, the random number generator would produce one of the set

   -1, 0, 1

   For 4 states separated by 1, the random number generator would produce one of the set

   -1.5, -0.5, +0.5, +1.5. And so on.

3. The Separation Distance W=stateWidth is used to change the separation between the discrete states discussed in #2 above. For the 4 state case, for example, the states would be
   -1.5W, -0.5W, 0.5W, 1.5W
4. Now to make the plot easier, the bins of the plot correspond to the same numbers set by the 'number discrete states' and the 'state separation'. On the other hand, the continuous distributions do not necessarily produce these discrete numbers – in fact, never to seldom. So the software artificially divides the continuous range into bins/states for the purpose of the histogram. So for the continuous distribution, the 'number of states' and the 'state separation' controls the number of bins/states and their separation. Try it with the software, it's easier.
5. Another point worth mentioning concerns the Mode Continuous and Discrete selection. The distribution can generate either a continuous range of numbers or one divided up into discrete but adjacent bins. The textbox for the number of discrete states refers to the number of vertical states (i.e., y states) and the states appear as bins across the horizontal axis of the histogram. An even number of bins skips zero. An odd number includes zero. The 'separation' textbox refers to the (vertical) separation of the discrete states and hence also the horizontal separation of histogram bins.

Section A7.2: Form1.designer.vb for AllanDev

The present section contains the design code for the Allan Deviation software. As previously mentioned, the software is written in Visual Basic .NET (VB.NET) rather than C#. The VB does not need the single characters such as '{' and '}' and ';'. These single tokens can be easily missed while copying. The VB.NET can be easily translated to C# if desired by using free online translators. The program listed here uses the older .net 'Forms' rather than the Universal or Windows Presentation Foundation. As with all Microsoft Visual Studio projects/solutions, the programmer has the option of either dragging/dropping controls/tools onto a given form so as to construct the Graphical User Interface (GUI), or else developing both the visual and functional aspects entirely within the code behind the displayed form. The drag-and-drop method is cumbersome to list in a book but is probably the better method for learning. Placing the Design and the Function code into the same Visual Studio Form can also be cumbersome due to the extensive size and thereby making it more difficult to focus on the code to be developed/maintained. So we have divided the project code into the Form1.Designer.vb and Form1.vb pages. The Designer.vb code must be put in place prior to the functional code. Alternatively, the GUI of Visual Studio can be used by dragging and dropping the desired tool from the 'tool box' and then altering the property parameters at the right hand side of the Visual Studio according to the parameters in the Designer code (below). As a note, we use the Microsoft Visual Studio Professional 2017. However, the Microsoft website does offer a free version.

Topic A7.2.1: Start a new project

First, a new project/solution needs to be implemented in the Microsoft Visual Studio (MVS). Open the MVS and select the menu sequence

File > New > Project

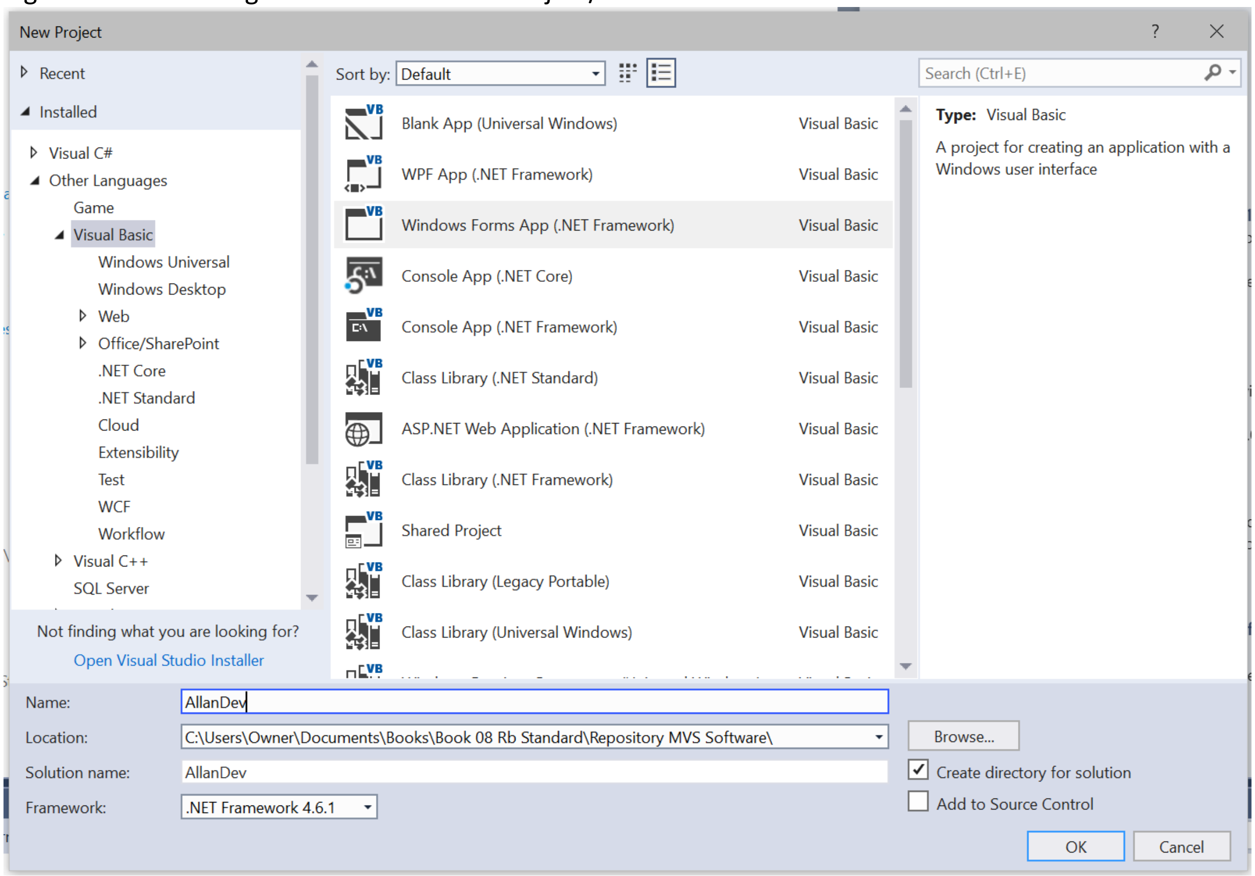
With reference to Figure A7.2, select from the left menu in the New Project dialogue:

Visual Basic > Windows Desktop

Single click on the right hand list item:

Windows Forms App (.Net Framework)

Figure A7.2: Initiating a new Visual Studio Project/Solution



At the bottom, select

.NET Framework 4.6.1

or a subsequent version. Place a checkmark in the box for making a directory, select a reasonable Location for the Solution/Project, and finally enter a name such as

AllanDev

Click the OK button. Probably now is a good time to save the project by either clicking the double-diskette in the menu bar at the top or using the menu sequence File > Save All. MVS should show 'Form1.vb'. If MVS suggests resizing the form to 100%, then agree to it so the form will appear as it would on the PC display.

Topic A7.2.2: Initial Preparation of Form1

Normally a person first designs the software Graphical User Interface (sGUI) using the MVS GUI (mGUI) by dragging and dropping 'Tools' from the 'Tool Box' on the left hand side of the mGUI. However alternatively sometimes the developer will simply type the code required for creating the various controls on the sGUI and thereby circumvent the drag-and-drop procedure while only requiring a single page of code (instead of the designer and functional code on separate pages). However the extra code for the controls becomes mixed with the functional code needed for the control events/animations as well as the other functional methods; this mixing can make it more difficult to focus on those portions of the code that need to be developed or maintained. In the present situation, we place much of the sGUI information into the Form1.Designer.vb file where it would automatically be placed during the drag-drop procedure. Unfortunately, for the charts used for the data visualization (i.e., plots), it is necessary (? Maybe the correct word is 'easier') to drag-and-drop two charts onto form1 prior to populating the form1.vb.designer page in order to circumvent a fatal error. It should be pointed out that all of the sGUI can be reconstructed here by dragging-and-dropping the controls listed for the form1.vb.designer and then setting the properties as also given by the designer content.

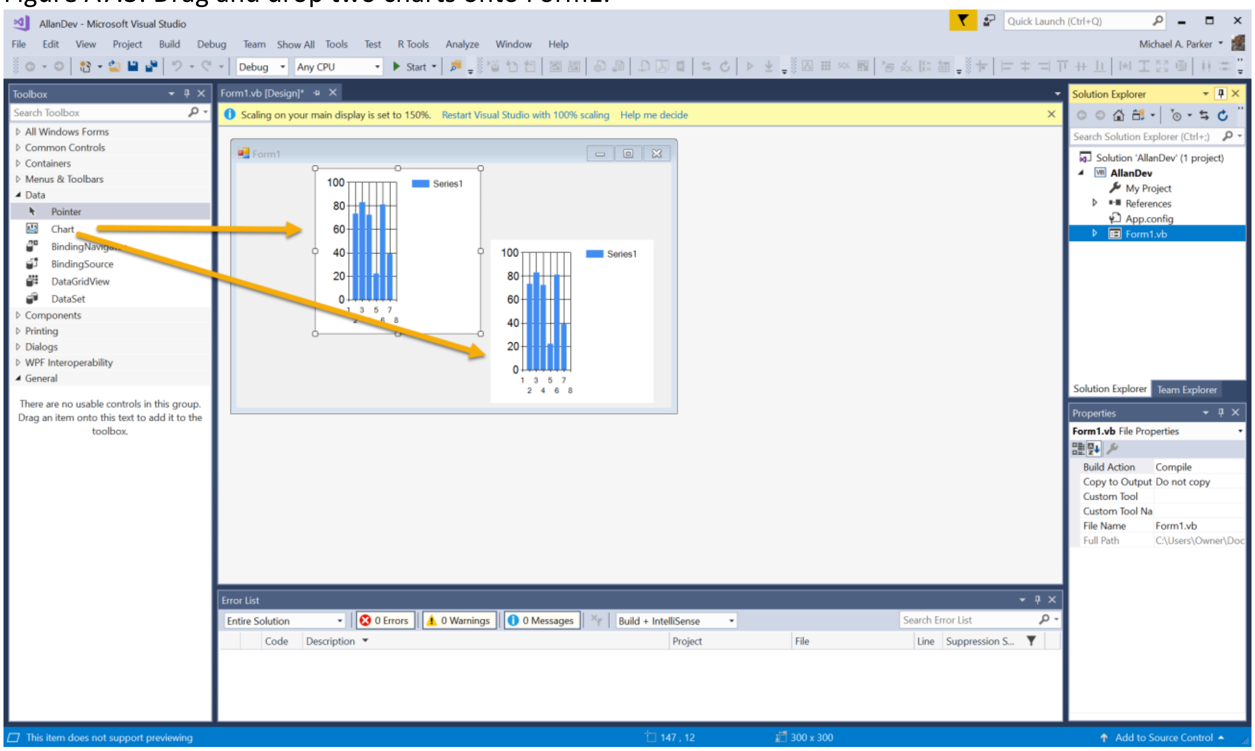
To start, make sure the Toolbox appears to the left of Form1 (left side), and the Solution Explorer on the right side and the Properties pane on the right side. If they are not visible, click the following sequences from the mGUI main menu:

View > Toolbox and View > Solution Explorer

Make sure 'Chart' can be found in the Toolbox under the 'Data' section. Some of the older versions of MVS require the 'Chart' to be downloaded to the Toolbox. If this is the case, to download it, select the menu sequence Tools>Choose Toolbox Items. Under the tab for the '.NET Framework Components', check the box next to 'Chart'.

Drag and drop two chart controls onto the Form1 from the Toolbox as shown in Figure A7.3. Do not worry about the size or the exact placement. Do not change any names. Make sure to Save All using either the double-diskette icon on the MVS menu bar or use File > Save All.

Figure A7.3: Drag and drop two charts onto Form1.



Topic A7.2.3: Access Form1.Designer.vb

Next, the Form1.Designer.vb code (listed below) needs to be added to the Form1.Designer.vb page. There are a couple of methods to access the Designer.vb page.

The first method of accessing the designer page consists of using the toolbar appearing at the top of **Solution Explorer** as shown at the upper right portion of Figure A7.3 and the zoomed-in view in Figure A7.4. Click on the fourth icon which looks like a folder with a couple of arrows. It might be necessary to click the arrows until the files similar to those shown in Figure A7.4 appear – the files should include Form1.Designer.vb. Double Click the **Form1.Designer.vb** entry in the list in the Solution Explorer.

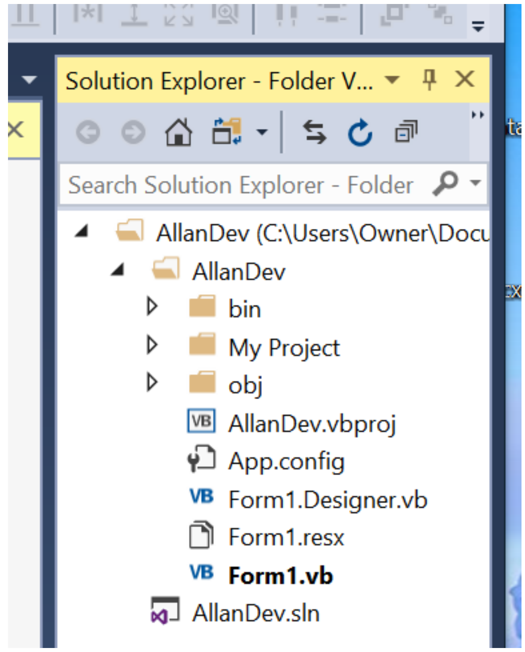


Figure A7.4: Fourth icon from the left should provide access to the Form1.Designer.vb page.

For the second double click Chart1 on Form1. In the functional coding window, find ‘Chart1.Click‘ in the handler. Right click ‘Chart1’ and select ‘Go to definition’. The Form1.Designer.vb page will appear. Go back to the functional coding window and delete any code related to the subroutine including ‘sub’ and ‘end sub’.

Now select all of the content in the Form1.Designer.vb page and delete it. Copy the code listed in Topic A7.2.4 into the Form1.Designer.vb. Once completed, the GUI for Form1 should appear similar to that shown in Figure A7.1. Save All. ***Please note that some lines wrap to the next line in the listing below. None of these ‘wrapped around’ lines have a carriage return in the middle of the line. For example, the line***

```
Dim ChartArea3 As System.Windows.Forms.DataVisualization.Charting.ChartArea = New
System.Windows.Forms.DataVisualization.Charting.ChartArea()
```

should really be entered as a single line of code:

```
Dim ChartArea3 As System.Windows.Forms.DataVisualization.Charting.ChartArea = New System.Windows.Forms.DataVisualization.Charting.ChartArea()
```

Each complete line of code has a typical Carriage Return CR, Line Feed LF at the end as normally caused by the ‘Enter’ key on a typical keyboard. If you are simply cutting and pasting the code from a Word document (etc) then you don’t need to worry about the wrapped lines.

Before leaving the issue of the designer code, a few comments should be made on the nature of the code.

1. Visual Studio places the lines of code in the designer because of the “Private Sub InitializeComponent()” and normally, using the GUI of Visual Studio, the programmer doesn’t

need to write any of them. Most of these lines can be easily understood based on the English language.

2. "MyBase" allows the code to call a member (such as a function) in the base class (sometimes called 'parent class') in order to perform a function for the derived class (sometimes called child class). Form1 is derived from the Form base class. Many methods for Form1 can be accessed through the base class using 'mybase'.
3. Notice the various components/controls are instantiated such as

   Me.Label7 = New System.Windows.Forms.Label()

   The 'Me' refers to Form1, Label7 is the name of the label given by the programmer in the Visual Studio GUI. Further down in the listing, the various properties of the label are defined such as

   Me.Label7.ForeColor = System.Drawing.Color.Black
   Me.Label7.Location = New System.Drawing.Point(30, 78)

   which defines the text color and the location of the upper left hand corner, respectively.
4. As previously mentioned, the Visual Studio enters all of the parameter information such as that in #3 above into the Properties window on the lower right side of the Visual Studio window. The programmer does the same when using the drag and drop method of constructing the GUI. As a note, the same lines of code can instead be entered into the Functional Code of Form1 if desired.
5. Notice the statements similar to

   Me.gbxDist.Controls.Add(Me.Label7)

   Here, gbxDist is the name of a group box simply used to visually divide up an area on the Form1 but in this case, it also contains a variety of tools/controls, one of which is an informational label named 'Label7'. The important point is that controls such as buttons are added to a collection of controls for the group box (etc.)
6. Sometimes components/controls have the 'WithEvents' keyword such as

   Friend WithEvents btnGen As Button

   The 'WithEvents' keyword indicates the button will produce an event when the button is clicked. The block of code known as the event handler on the functional code page (Form1.vb) will do something in response such as produce random numbers.

After having copied the listing in Topic A7.2.4 into the Form1.Designer.vb page, proceed to Appendix Section A7.3 for the Form1 functional code. Again, note some code wraps around to the next line – do not place a carriage return where the wrap occurs – the code should be all one coding line in the program.

Topic A7.2.4: Listing for Form1.designer.vb

Delete all code in the Form1.designer.vb and then cut and paste all of the following code into the Form1.designer.vb. Save All when finished.

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()>
Partial Class Form1
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()>
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
```

```
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()>
    Private Sub InitializeComponent()
        Dim ChartArea3 As System.Windows.Forms.DataVisualization.Charting.ChartArea = New
System.Windows.Forms.DataVisualization.Charting.ChartArea()
        Dim Legend3 As System.Windows.Forms.DataVisualization.Charting.Legend = New
System.Windows.Forms.DataVisualization.Charting.Legend()
        Dim Series4 As System.Windows.Forms.DataVisualization.Charting.Series = New
System.Windows.Forms.DataVisualization.Charting.Series()
        Dim Series5 As System.Windows.Forms.DataVisualization.Charting.Series = New
System.Windows.Forms.DataVisualization.Charting.Series()
        Dim ChartArea4 As System.Windows.Forms.DataVisualization.Charting.ChartArea = New
System.Windows.Forms.DataVisualization.Charting.ChartArea()
        Dim Legend4 As System.Windows.Forms.DataVisualization.Charting.Legend = New
System.Windows.Forms.DataVisualization.Charting.Legend()
        Dim Series6 As System.Windows.Forms.DataVisualization.Charting.Series = New
System.Windows.Forms.DataVisualization.Charting.Series()
        Me.gbxDist = New System.Windows.Forms.GroupBox()
        Me.Label7 = New System.Windows.Forms.Label()
        Me.tbxStdDev = New System.Windows.Forms.TextBox()
        Me.cbxRW = New System.Windows.Forms.CheckBox()
        Me.rbNormal = New System.Windows.Forms.RadioButton()
        Me.rbUnif = New System.Windows.Forms.RadioButton()
        Me.sfd01 = New System.Windows.Forms.SaveFileDialog()
        Me.tbxSeed = New System.Windows.Forms.TextBox()
        Me.Label2 = New System.Windows.Forms.Label()
        Me.tbxNumDisSts = New System.Windows.Forms.TextBox()
        Me.Label3 = New System.Windows.Forms.Label()
        Me.MenuStrip1 = New System.Windows.Forms.MenuStrip()
        Me.mnuFile = New System.Windows.Forms.ToolStripMenuItem()
        Me.mnuFopenSamp = New System.Windows.Forms.ToolStripMenuItem()
        Me.mnuFsaveSamp = New System.Windows.Forms.ToolStripMenuItem()
        Me.ofd01 = New System.Windows.Forms.OpenFileDialog()
        Me.Chart2 = New System.Windows.Forms.DataVisualization.Charting.Chart()
        Me.Panel1 = New System.Windows.Forms.Panel()
        Me.btnGen = New System.Windows.Forms.Button()
        Me.Label8 = New System.Windows.Forms.Label()
        Me.tbxDiscrSep = New System.Windows.Forms.TextBox()
        Me.Label1 = New System.Windows.Forms.Label()
        Me.tbxNumPnts = New System.Windows.Forms.TextBox()
        Me.gbxMode = New System.Windows.Forms.GroupBox()
        Me.rbCont = New System.Windows.Forms.RadioButton()
        Me.rbDiscr = New System.Windows.Forms.RadioButton()
        Me.btnPlotRand = New System.Windows.Forms.Button()
```

```
        Me.GroupBox1 = New System.Windows.Forms.GroupBox()
        Me.rbPnt = New System.Windows.Forms.RadioButton()
        Me.rbLine = New System.Windows.Forms.RadioButton()
        Me.Panel2 = New System.Windows.Forms.Panel()
        Me.cbxPlotVar = New System.Windows.Forms.CheckBox()
        Me.cbxPlotDev = New System.Windows.Forms.CheckBox()
        Me.tbxAdev = New System.Windows.Forms.TextBox()
        Me.tbxAvar = New System.Windows.Forms.TextBox()
        Me.tbxNTAUmax = New System.Windows.Forms.TextBox()
        Me.Label4 = New System.Windows.Forms.Label()
        Me.btnAVAR = New System.Windows.Forms.Button()
        Me.Panel3 = New System.Windows.Forms.Panel()
        Me.Chart1 = New System.Windows.Forms.DataVisualization.Charting.Chart()
        Me.gbxDist.SuspendLayout()
        Me.MenuStrip1.SuspendLayout()
        CType(Me.Chart2, System.ComponentModel.ISupportInitialize).BeginInit()
        Me.Panel1.SuspendLayout()
        Me.gbxMode.SuspendLayout()
        Me.GroupBox1.SuspendLayout()
        Me.Panel2.SuspendLayout()
        Me.Panel3.SuspendLayout()
        CType(Me.Chart1, System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'gbxDist
        '
        Me.gbxDist.Controls.Add(Me.Label7)
        Me.gbxDist.Controls.Add(Me.tbxStdDev)
        Me.gbxDist.Controls.Add(Me.cbxRW)
        Me.gbxDist.Controls.Add(Me.rbNormal)
        Me.gbxDist.Controls.Add(Me.rbUnif)
        Me.gbxDist.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.gbxDist.ForeColor = System.Drawing.Color.Blue
        Me.gbxDist.Location = New System.Drawing.Point(9, 46)
        Me.gbxDist.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.gbxDist.Name = "gbxDist"
        Me.gbxDist.Padding = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.gbxDist.Size = New System.Drawing.Size(308, 122)
        Me.gbxDist.TabIndex = 2
        Me.gbxDist.TabStop = False
        Me.gbxDist.Text = "Type"
        '
        'Label7
        '
        Me.Label7.AutoSize = True
        Me.Label7.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label7.ForeColor = System.Drawing.Color.Black
        Me.Label7.Location = New System.Drawing.Point(30, 78)
        Me.Label7.Margin = New System.Windows.Forms.Padding(4, 0, 4, 0)
        Me.Label7.Name = "Label7"
        Me.Label7.Size = New System.Drawing.Size(88, 25)
        Me.Label7.TabIndex = 12
        Me.Label7.Text = "Std Dev:"
        '
        'tbxStdDev
        '
```

```
        Me.tbxStdDev.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.tbxStdDev.Location = New System.Drawing.Point(126, 78)
        Me.tbxStdDev.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.tbxStdDev.Name = "tbxStdDev"
        Me.tbxStdDev.Size = New System.Drawing.Size(134, 30)
        Me.tbxStdDev.TabIndex = 13
        Me.tbxStdDev.Text = "1"
        Me.tbxStdDev.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'cbxRW
        '
        Me.cbxRW.AutoSize = True
        Me.cbxRW.ForeColor = System.Drawing.Color.Black
        Me.cbxRW.Location = New System.Drawing.Point(207, 34)
        Me.cbxRW.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.cbxRW.Name = "cbxRW"
        Me.cbxRW.Size = New System.Drawing.Size(71, 29)
        Me.cbxRW.TabIndex = 11
        Me.cbxRW.Text = "RW"
        Me.cbxRW.UseVisualStyleBackColor = True
        '
        'rbNormal
        '
        Me.rbNormal.AutoSize = True
        Me.rbNormal.ForeColor = System.Drawing.Color.Black
        Me.rbNormal.Location = New System.Drawing.Point(110, 34)
        Me.rbNormal.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.rbNormal.Name = "rbNormal"
        Me.rbNormal.Size = New System.Drawing.Size(84, 29)
        Me.rbNormal.TabIndex = 10
        Me.rbNormal.Text = "Norm"
        Me.rbNormal.UseVisualStyleBackColor = True
        '
        'rbUnif
        '
        Me.rbUnif.AutoSize = True
        Me.rbUnif.Checked = True
        Me.rbUnif.ForeColor = System.Drawing.Color.Black
        Me.rbUnif.Location = New System.Drawing.Point(21, 34)
        Me.rbUnif.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.rbUnif.Name = "rbUnif"
        Me.rbUnif.Size = New System.Drawing.Size(71, 29)
        Me.rbUnif.TabIndex = 9
        Me.rbUnif.TabStop = True
        Me.rbUnif.Text = "Unif"
        Me.rbUnif.UseVisualStyleBackColor = True
        '
        'tbxSeed
        '
        Me.tbxSeed.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.tbxSeed.Location = New System.Drawing.Point(242, 5)
        Me.tbxSeed.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.tbxSeed.Name = "tbxSeed"
        Me.tbxSeed.Size = New System.Drawing.Size(74, 30)
        Me.tbxSeed.TabIndex = 6
        Me.tbxSeed.Text = "12"
```

```
        Me.tbxSeed.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Label2
        '
        Me.Label2.AutoSize = True
        Me.Label2.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label2.Location = New System.Drawing.Point(180, 8)
        Me.Label2.Margin = New System.Windows.Forms.Padding(4, 0, 4, 0)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(59, 25)
        Me.Label2.TabIndex = 5
        Me.Label2.Text = "Seed"
        '
        'tbxNumDisSts
        '
        Me.tbxNumDisSts.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.tbxNumDisSts.Location = New System.Drawing.Point(186, 246)
        Me.tbxNumDisSts.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.tbxNumDisSts.Name = "tbxNumDisSts"
        Me.tbxNumDisSts.Size = New System.Drawing.Size(92, 30)
        Me.tbxNumDisSts.TabIndex = 9
        Me.tbxNumDisSts.Text = "10"
        Me.tbxNumDisSts.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Label3
        '
        Me.Label3.AutoSize = True
        Me.Label3.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label3.Location = New System.Drawing.Point(12, 251)
        Me.Label3.Margin = New System.Windows.Forms.Padding(4, 0, 4, 0)
        Me.Label3.Name = "Label3"
        Me.Label3.Size = New System.Drawing.Size(160, 25)
        Me.Label3.TabIndex = 8
        Me.Label3.Text = "# Discrete States"
        Me.Label3.UseMnemonic = False
        Me.Label3.UseWaitCursor = True
        '
        'MenuStrip1
        '
        Me.MenuStrip1.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.MenuStrip1.ImageScalingSize = New System.Drawing.Size(24, 24)
        Me.MenuStrip1.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.mnuFile})
        Me.MenuStrip1.Location = New System.Drawing.Point(0, 0)
        Me.MenuStrip1.Name = "MenuStrip1"
        Me.MenuStrip1.Padding = New System.Windows.Forms.Padding(9, 3, 0, 3)
        Me.MenuStrip1.Size = New System.Drawing.Size(1192, 35)
        Me.MenuStrip1.TabIndex = 10
        Me.MenuStrip1.Text = "MenuStrip1"
        '
        'mnuFile
        '
        Me.mnuFile.DropDownItems.AddRange(New System.Windows.Forms.ToolStripItem() {Me.mnuFopenSamp,
Me.mnuFsaveSamp})
        Me.mnuFile.Name = "mnuFile"
```

```
        Me.mnuFile.Size = New System.Drawing.Size(55, 29)
        Me.mnuFile.Text = "File"
        '
        'mnuFopenSamp
        '
        Me.mnuFopenSamp.Name = "mnuFopenSamp"
        Me.mnuFopenSamp.Size = New System.Drawing.Size(227, 30)
        Me.mnuFopenSamp.Text = "Open Samples"
        '
        'mnuFsaveSamp
        '
        Me.mnuFsaveSamp.Name = "mnuFsaveSamp"
        Me.mnuFsaveSamp.Size = New System.Drawing.Size(227, 30)
        Me.mnuFsaveSamp.Text = "Save Samples"
        '
        'ofd01
        '
        Me.ofd01.FileName = "OpenFileDialog1"
        '
        'Chart2
        '
        ChartArea3.Name = "ChartArea1"
        Me.Chart2.ChartAreas.Add(ChartArea3)
        Legend3.Enabled = False
        Legend3.Name = "Legend1"
        Me.Chart2.Legends.Add(Legend3)
        Me.Chart2.Location = New System.Drawing.Point(363, 417)
        Me.Chart2.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.Chart2.Name = "Chart2"
        Series4.ChartArea = "ChartArea1"
        Series4.ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Point
        Series4.Legend = "Legend1"
        Series4.Name = "Series1"
        Series5.ChartArea = "ChartArea1"
        Series5.Color = System.Drawing.Color.Blue
        Series5.Legend = "Legend1"
        Series5.Name = "Series2"
        Me.Chart2.Series.Add(Series4)
        Me.Chart2.Series.Add(Series5)
        Me.Chart2.Size = New System.Drawing.Size(810, 346)
        Me.Chart2.TabIndex = 11
        Me.Chart2.Text = "Chart2"
        '
        'Panel1
        '
        Me.Panel1.BackColor = System.Drawing.Color.FromArgb(CType(CType(192, Byte), Integer), CType(CType(255, Byte),
Integer), CType(CType(192, Byte), Integer))
        Me.Panel1.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.Panel1.Controls.Add(Me.btnGen)
        Me.Panel1.Controls.Add(Me.Label8)
        Me.Panel1.Controls.Add(Me.tbxDiscrSep)
        Me.Panel1.Controls.Add(Me.Label1)
        Me.Panel1.Controls.Add(Me.tbxNumPnts)
        Me.Panel1.Controls.Add(Me.gbxMode)
        Me.Panel1.Controls.Add(Me.Label2)
        Me.Panel1.Controls.Add(Me.gbxDist)
        Me.Panel1.Controls.Add(Me.Label3)
        Me.Panel1.Controls.Add(Me.tbxSeed)
```

```
        Me.Panel1.Controls.Add(Me.tbxNumDisSts)
        Me.Panel1.Location = New System.Drawing.Point(16, 51)
        Me.Panel1.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.Panel1.Name = "Panel1"
        Me.Panel1.Size = New System.Drawing.Size(328, 384)
        Me.Panel1.TabIndex = 15
        '
        'btnGen
        '
        Me.btnGen.BackColor = System.Drawing.Color.FromArgb(CType(CType(255, Byte), Integer), CType(CType(192, Byte),
Integer), CType(CType(192, Byte), Integer))
        Me.btnGen.FlatStyle = System.Windows.Forms.FlatStyle.Popup
        Me.btnGen.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btnGen.Location = New System.Drawing.Point(12, 332)
        Me.btnGen.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.btnGen.Name = "btnGen"
        Me.btnGen.Size = New System.Drawing.Size(294, 38)
        Me.btnGen.TabIndex = 20
        Me.btnGen.Text = "Generate Random Numbers"
        Me.btnGen.UseVisualStyleBackColor = False
        '
        'Label8
        '
        Me.Label8.AutoSize = True
        Me.Label8.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label8.Location = New System.Drawing.Point(12, 289)
        Me.Label8.Margin = New System.Windows.Forms.Padding(4, 0, 4, 0)
        Me.Label8.Name = "Label8"
        Me.Label8.Size = New System.Drawing.Size(107, 25)
        Me.Label8.TabIndex = 22
        Me.Label8.Text = "Separation"
        Me.Label8.UseMnemonic = False
        Me.Label8.UseWaitCursor = True
        '
        'tbxDiscrSep
        '
        Me.tbxDiscrSep.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.tbxDiscrSep.Location = New System.Drawing.Point(136, 285)
        Me.tbxDiscrSep.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.tbxDiscrSep.Name = "tbxDiscrSep"
        Me.tbxDiscrSep.Size = New System.Drawing.Size(170, 30)
        Me.tbxDiscrSep.TabIndex = 23
        Me.tbxDiscrSep.Text = "0.1"
        Me.tbxDiscrSep.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Label1
        '
        Me.Label1.AutoSize = True
        Me.Label1.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label1.Location = New System.Drawing.Point(4, 8)
        Me.Label1.Margin = New System.Windows.Forms.Padding(4, 0, 4, 0)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(62, 25)
        Me.Label1.TabIndex = 11
```

```
        Me.Label1.Text = "#Pnts"
        '
        'tbxNumPnts
        '
        Me.tbxNumPnts.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.tbxNumPnts.Location = New System.Drawing.Point(68, 5)
        Me.tbxNumPnts.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.tbxNumPnts.Name = "tbxNumPnts"
        Me.tbxNumPnts.Size = New System.Drawing.Size(97, 30)
        Me.tbxNumPnts.TabIndex = 12
        Me.tbxNumPnts.Text = "512"
        Me.tbxNumPnts.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'gbxMode
        '
        Me.gbxMode.Controls.Add(Me.rbCont)
        Me.gbxMode.Controls.Add(Me.rbDiscr)
        Me.gbxMode.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.gbxMode.ForeColor = System.Drawing.Color.Blue
        Me.gbxMode.Location = New System.Drawing.Point(9, 172)
        Me.gbxMode.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.gbxMode.Name = "gbxMode"
        Me.gbxMode.Padding = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.gbxMode.Size = New System.Drawing.Size(308, 65)
        Me.gbxMode.TabIndex = 10
        Me.gbxMode.TabStop = False
        Me.gbxMode.Text = "Mode"
        '
        'rbCont
        '
        Me.rbCont.AutoSize = True
        Me.rbCont.Checked = True
        Me.rbCont.ForeColor = System.Drawing.Color.Black
        Me.rbCont.Location = New System.Drawing.Point(3, 29)
        Me.rbCont.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.rbCont.Name = "rbCont"
        Me.rbCont.Size = New System.Drawing.Size(137, 29)
        Me.rbCont.TabIndex = 4
        Me.rbCont.TabStop = True
        Me.rbCont.Text = "Continuous"
        Me.rbCont.UseVisualStyleBackColor = True
        '
        'rbDiscr
        '
        Me.rbDiscr.AutoSize = True
        Me.rbDiscr.ForeColor = System.Drawing.Color.Black
        Me.rbDiscr.Location = New System.Drawing.Point(186, 29)
        Me.rbDiscr.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.rbDiscr.Name = "rbDiscr"
        Me.rbDiscr.Size = New System.Drawing.Size(108, 29)
        Me.rbDiscr.TabIndex = 3
        Me.rbDiscr.Text = "Discrete"
        Me.rbDiscr.UseVisualStyleBackColor = True
        '
        'btnPlotRand
        '
```

```
        Me.btnPlotRand.BackColor = System.Drawing.Color.FromArgb(CType(CType(255, Byte), Integer), CType(CType(192, Byte), Integer), CType(CType(192, Byte), Integer))
        Me.btnPlotRand.FlatStyle = System.Windows.Forms.FlatStyle.Popup
        Me.btnPlotRand.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btnPlotRand.Location = New System.Drawing.Point(129, 22)
        Me.btnPlotRand.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.btnPlotRand.Name = "btnPlotRand"
        Me.btnPlotRand.Size = New System.Drawing.Size(180, 37)
        Me.btnPlotRand.TabIndex = 23
        Me.btnPlotRand.Text = "Plot Rand Nums"
        Me.btnPlotRand.UseVisualStyleBackColor = False
        '
        'GroupBox1
        '
        Me.GroupBox1.Controls.Add(Me.rbPnt)
        Me.GroupBox1.Controls.Add(Me.rbLine)
        Me.GroupBox1.Location = New System.Drawing.Point(16, 3)
        Me.GroupBox1.Name = "GroupBox1"
        Me.GroupBox1.Size = New System.Drawing.Size(94, 69)
        Me.GroupBox1.TabIndex = 21
        Me.GroupBox1.TabStop = False
        '
        'rbPnt
        '
        Me.rbPnt.AutoSize = True
        Me.rbPnt.Location = New System.Drawing.Point(12, 37)
        Me.rbPnt.Name = "rbPnt"
        Me.rbPnt.Size = New System.Drawing.Size(70, 24)
        Me.rbPnt.TabIndex = 1
        Me.rbPnt.Text = "Point"
        Me.rbPnt.UseVisualStyleBackColor = True
        '
        'rbLine
        '
        Me.rbLine.AutoSize = True
        Me.rbLine.Checked = True
        Me.rbLine.Location = New System.Drawing.Point(12, 8)
        Me.rbLine.Name = "rbLine"
        Me.rbLine.Size = New System.Drawing.Size(64, 24)
        Me.rbLine.TabIndex = 0
        Me.rbLine.TabStop = True
        Me.rbLine.Text = "Line"
        Me.rbLine.UseVisualStyleBackColor = True
        '
        'Panel2
        '
        Me.Panel2.BackColor = System.Drawing.Color.FromArgb(CType(CType(192, Byte), Integer), CType(CType(255, Byte), Integer), CType(CType(192, Byte), Integer))
        Me.Panel2.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.Panel2.Controls.Add(Me.cbxPlotVar)
        Me.Panel2.Controls.Add(Me.cbxPlotDev)
        Me.Panel2.Controls.Add(Me.tbxAdev)
        Me.Panel2.Controls.Add(Me.tbxAvar)
        Me.Panel2.Controls.Add(Me.tbxNTAUmax)
        Me.Panel2.Controls.Add(Me.Label4)
        Me.Panel2.Controls.Add(Me.btnAVAR)
```

```
        Me.Panel2.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Panel2.Location = New System.Drawing.Point(16, 565)
        Me.Panel2.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.Panel2.Name = "Panel2"
        Me.Panel2.Size = New System.Drawing.Size(328, 192)
        Me.Panel2.TabIndex = 16
        '
        'cbxPlotVar
        '
        Me.cbxPlotVar.AutoSize = True
        Me.cbxPlotVar.ForeColor = System.Drawing.Color.Black
        Me.cbxPlotVar.Location = New System.Drawing.Point(12, 149)
        Me.cbxPlotVar.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.cbxPlotVar.Name = "cbxPlotVar"
        Me.cbxPlotVar.Size = New System.Drawing.Size(165, 29)
        Me.cbxPlotVar.TabIndex = 28
        Me.cbxPlotVar.Text = "Allan Variance"
        Me.cbxPlotVar.UseVisualStyleBackColor = True
        '
        'cbxPlotDev
        '
        Me.cbxPlotDev.AutoSize = True
        Me.cbxPlotDev.Checked = True
        Me.cbxPlotDev.CheckState = System.Windows.Forms.CheckState.Checked
        Me.cbxPlotDev.ForeColor = System.Drawing.Color.Blue
        Me.cbxPlotDev.Location = New System.Drawing.Point(12, 109)
        Me.cbxPlotDev.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.cbxPlotDev.Name = "cbxPlotDev"
        Me.cbxPlotDev.Size = New System.Drawing.Size(168, 29)
        Me.cbxPlotDev.TabIndex = 27
        Me.cbxPlotDev.Text = "Allan Deviation"
        Me.cbxPlotDev.UseVisualStyleBackColor = True
        '
        'tbxAdev
        '
        Me.tbxAdev.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.tbxAdev.ForeColor = System.Drawing.Color.Blue
        Me.tbxAdev.Location = New System.Drawing.Point(195, 106)
        Me.tbxAdev.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.tbxAdev.Name = "tbxAdev"
        Me.tbxAdev.ReadOnly = True
        Me.tbxAdev.Size = New System.Drawing.Size(120, 30)
        Me.tbxAdev.TabIndex = 26
        Me.tbxAdev.Text = "0"
        Me.tbxAdev.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'tbxAvar
        '
        Me.tbxAvar.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.tbxAvar.Location = New System.Drawing.Point(195, 149)
        Me.tbxAvar.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.tbxAvar.Name = "tbxAvar"
        Me.tbxAvar.ReadOnly = True
        Me.tbxAvar.Size = New System.Drawing.Size(121, 30)
        Me.tbxAvar.TabIndex = 22
```

```
        Me.tbxAvar.Text = "0"
        Me.tbxAvar.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'tbxNTAUmax
        '
        Me.tbxNTAUmax.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.tbxNTAUmax.Location = New System.Drawing.Point(150, 9)
        Me.tbxNTAUmax.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.tbxNTAUmax.Name = "tbxNTAUmax"
        Me.tbxNTAUmax.Size = New System.Drawing.Size(106, 30)
        Me.tbxNTAUmax.TabIndex = 18
        Me.tbxNTAUmax.Text = "256"
        Me.tbxNTAUmax.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Label4
        '
        Me.Label4.AutoSize = True
        Me.Label4.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label4.Location = New System.Drawing.Point(40, 14)
        Me.Label4.Margin = New System.Windows.Forms.Padding(4, 0, 4, 0)
        Me.Label4.Name = "Label4"
        Me.Label4.Size = New System.Drawing.Size(101, 25)
        Me.Label4.TabIndex = 17
        Me.Label4.Text = "Max nTau"
        '
        'btnAVAR
        '
        Me.btnAVAR.BackColor = System.Drawing.Color.FromArgb(CType(CType(255, Byte), Integer), CType(CType(192, Byte),
Integer), CType(CType(192, Byte), Integer))
        Me.btnAVAR.FlatStyle = System.Windows.Forms.FlatStyle.Popup
        Me.btnAVAR.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.btnAVAR.Location = New System.Drawing.Point(12, 54)
        Me.btnAVAR.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.btnAVAR.Name = "btnAVAR"
        Me.btnAVAR.Size = New System.Drawing.Size(297, 35)
        Me.btnAVAR.TabIndex = 16
        Me.btnAVAR.Text = "Plot AVAR ADEV"
        Me.btnAVAR.UseVisualStyleBackColor = False
        '
        'Panel3
        '
        Me.Panel3.BackColor = System.Drawing.Color.FromArgb(CType(CType(192, Byte), Integer), CType(CType(255, Byte),
Integer), CType(CType(192, Byte), Integer))
        Me.Panel3.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.Panel3.Controls.Add(Me.btnPlotRand)
        Me.Panel3.Controls.Add(Me.GroupBox1)
        Me.Panel3.Location = New System.Drawing.Point(16, 458)
        Me.Panel3.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
        Me.Panel3.Name = "Panel3"
        Me.Panel3.Size = New System.Drawing.Size(328, 79)
        Me.Panel3.TabIndex = 22
        '
        'Chart1
        '
        ChartArea4.Name = "ChartArea1"
```

```
    Me.Chart1.ChartAreas.Add(ChartArea4)
    Legend4.Enabled = False
    Legend4.Name = "Legend1"
    Me.Chart1.Legends.Add(Legend4)
    Me.Chart1.Location = New System.Drawing.Point(363, 51)
    Me.Chart1.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
    Me.Chart1.Name = "Chart1"
    Series6.ChartArea = "ChartArea1"
    Series6.Legend = "Legend1"
    Series6.Name = "Series1"
    Me.Chart1.Series.Add(Series6)
    Me.Chart1.Size = New System.Drawing.Size(810, 346)
    Me.Chart1.TabIndex = 23
    Me.Chart1.Text = "Chart1"
    '
    'Form1
    '
    Me.AutoScaleDimensions = New System.Drawing.SizeF(9.0!, 20.0!)
    Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
    Me.ClientSize = New System.Drawing.Size(1192, 778)
    Me.Controls.Add(Me.Chart1)
    Me.Controls.Add(Me.Panel3)
    Me.Controls.Add(Me.Panel2)
    Me.Controls.Add(Me.Panel1)
    Me.Controls.Add(Me.Chart2)
    Me.Controls.Add(Me.MenuStrip1)
    Me.MainMenuStrip = Me.MenuStrip1
    Me.Margin = New System.Windows.Forms.Padding(4, 5, 4, 5)
    Me.Name = "Form1"
    Me.Text = "Allan Deviation"
    Me.gbxDist.ResumeLayout(False)
    Me.gbxDist.PerformLayout()
    Me.MenuStrip1.ResumeLayout(False)
    Me.MenuStrip1.PerformLayout()
    CType(Me.Chart2, System.ComponentModel.ISupportInitialize).EndInit()
    Me.Panel1.ResumeLayout(False)
    Me.Panel1.PerformLayout()
    Me.gbxMode.ResumeLayout(False)
    Me.gbxMode.PerformLayout()
    Me.GroupBox1.ResumeLayout(False)
    Me.GroupBox1.PerformLayout()
    Me.Panel2.ResumeLayout(False)
    Me.Panel2.PerformLayout()
    Me.Panel3.ResumeLayout(False)
    CType(Me.Chart1, System.ComponentModel.ISupportInitialize).EndInit()
    Me.ResumeLayout(False)
    Me.PerformLayout()

End Sub
Friend WithEvents gbxDist As GroupBox
Friend WithEvents sfd01 As SaveFileDialog
Friend WithEvents tbxSeed As TextBox
Friend WithEvents Label2 As Label
Friend WithEvents tbxNumDisSts As TextBox
Friend WithEvents Label3 As Label
Friend WithEvents MenuStrip1 As MenuStrip
Friend WithEvents mnuFile As ToolStripMenuItem
Friend WithEvents mnuFopenSamp As ToolStripMenuItem
```

```
    Friend WithEvents mnuFsaveSamp As ToolStripMenuItem
    Friend WithEvents ofd01 As OpenFileDialog
    Friend WithEvents Chart2 As DataVisualization.Charting.Chart
    Friend WithEvents Panel1 As Panel
    Friend WithEvents Panel2 As Panel
    Friend WithEvents tbxNTAUmax As TextBox
    Friend WithEvents Label4 As Label
    Friend WithEvents btnAVAR As Button
    Friend WithEvents gbxMode As GroupBox
    Friend WithEvents rbCont As RadioButton
    Friend WithEvents rbDiscr As RadioButton
    Friend WithEvents Label1 As Label
    Friend WithEvents tbxNumPnts As TextBox
    Friend WithEvents Label7 As Label
    Friend WithEvents tbxStdDev As TextBox
    Friend WithEvents cbxRW As CheckBox
    Friend WithEvents rbNormal As RadioButton
    Friend WithEvents rbUnif As RadioButton
    Friend WithEvents GroupBox1 As GroupBox
    Friend WithEvents rbPnt As RadioButton
    Friend WithEvents rbLine As RadioButton
    Friend WithEvents btnGen As Button
    Friend WithEvents tbxAvar As TextBox
    Friend WithEvents btnPlotRand As Button
    Friend WithEvents Label8 As Label
    Friend WithEvents tbxDiscrSep As TextBox
    Friend WithEvents Panel3 As Panel
    Friend WithEvents Chart1 As DataVisualization.Charting.Chart
    Friend WithEvents tbxAdev As TextBox
    Friend WithEvents cbxPlotVar As CheckBox
    Friend WithEvents cbxPlotDev As CheckBox
End Class
```

Section A7.3: Form1 Source Code for AllanDev

The previous section provided the code listing for the Graphical User Interface GUI of the Allan Deviation software (AllanDev). Now the various controls/components need to be made functional by placing code on the Form1.vb page. At this point, the AllanDev GUI should be visible in the Visual Studio. As previously discussed, the GUI design could have been made by simply dragging-dropping the various tools/controls into the Visual Studio GUI design area and using the parameters listed in the previous section to fill-in the properties box for each component. Having the source code in this appendix makes it easy to modify and improve the software as the programmer sees fit for the application. Some comments on the source code can be found in Topic A7.3.3 after the source code listing.

Topic A7.3.1: Notes on Form1.vb and the EXE file

The AllanDev Interface GUI should be visible in Visual Studio. In the Solution Explorer window, *right* click Form1.vb under the folder named the same as your software (AllanDev). Then select 'View Code'. The functional code window should be displayed. If it didn't work, double click one of the components of the GUI. In either case, delete any and all code on the page. The code listed below should be copied directly into the page. Once completed, make sure any errors have been resolved. Don't worry about the 3 or 4 'messages' that might appear; the messengers don't understand the current application and don't give relevant messages – ignore them. Then type CTRL F5, or else click the 'Start' on the tool bar just below the main menu.

As a note, once the program has successfully run, the EXE file can be added to the desktop and run by double clicking on the icon. To access the EXE file, open the directory containing the AllanDev software using Windows Explorer. The directory will contain AllanDev.sln and also a folder labeled as AllanDev. Open this second folder and then open the 'bin' folder and then copy the .exe file. It should be pointed out that Visual Studio can make an installer (MSI) although it involves some extra steps. The .exe file by itself does not install in the Windows registry.

Topic A7.3.2: Form1 Source Code

Open the function code page, namely Form1.vb, and delete any and all code that might be there, and then copy the code listed below into the Form1.vb. The comments can be omitted as desired.

```
' variable types in vb.net
' https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/data-types/
' using charts
' https://www.i-programmer.info/programming/uiux/2756-getting-started-with-net-charts.html
' good
' https://docs.microsoft.com/en-us/previous-versions/dd456632(v=vs.140)?redirectedfrom=MSDN
'format string
' https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualbasic.strings.format?view=netframework-4.8
' free software for probability distributions (and more
' http://accord-framework.net/
' http://accord-framework.net/docs/html/T_Accord_Statistics_Distributions_Univariate_NormalDistribution.htm
' other normal
' https://stackoverflow.com/questions/75677/converting-a-uniform-distribution-to-a-normal-distribution?rq=1
' inverse of ERF and defs of ERT
' https://stackoverflow.com/questions/27229371/inverse-error-function-in-c
' chart x-axis label format
' https://stackoverflow.com/questions/18025263/formatting-chart-axis-labels/18026219
' https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-numeric-format-strings?redirectedfrom=MSDN
' interface
' https://stackoverflow.com/questions/14085784/vb-net-what-is-the-purpose-of-a-class-or-module

'Basic User Interface
' Form1.vb[design] is divided into 4 basic regions.
' Right side has the plots. Top: Distribution and random Numbers. Bottom: Allan Dev and Var
' Left side has 3 sections:
'   Top: Random numbers Mode, Type, parameters; generate and plot distribution
'   Mid: plots random numbers
'   Bot: plots Allan Variance and Allan Deviation vs. index (not the time)
' Top Section:
'   Enter number of points to generate and a seed for any of the distributions
'   Select the distribution Type of normal or uniform and select whether or not Random Walk
'   Select the mode of either continuous or discrete. Continuous produces any real number
'       in a range. Discrete produces specific numbers (where W = State width = state separation
'       such as for example, numStates=5 would give one of -2W, -W, 0, W, 2W for each generation
'       Event whereas, for example, numStates=4 would give one of -1.5W, -0.5W, +0.5W, 1.5W (no
'       zero), where W = stateWidth. The values are quantized according to the number of discrete
'       values (numStates) - the separation between the states is W=stateWidth. Click the Generate
'       Random Number button to generate random numbers (# = numPnts) And place them In the Samples
```

```
'     list and plot the distribution as in number in a bin versus discrete/quantized number
'     (i.e. frequency plot).
'
'Basic software operations:
'  The class RandomNumGenerator inherits the Random class built into the Microsoft Framework and
'  adds extra methods to generate uniformly and normally distributed numbers either of the
'  continuous or discrete variety. The objects 'Samples' instantiated on the SamplesLists class
'  store the random numbers generated by the object 'Generator' instantiated on the class
'  RandomNumGenerator. The 'Samples' object also has methods to produce the frequency
'  histogram. The Random Walk is handled in 'Samples' by reading in the generated random number
'  and adding the value to the sum of previous random numbers and storing that number in the
'  internal List. The list in 'Samples' can be accessed to calculate the Allan Deviation and
'  produce the plot.  The 'File' menu item makes it possible to either save the 'Samples' or
'  else load the samples from a file. The files consist of text entries separated by CR and LF.

Imports Microsoft.VisualBasic.VBMath
Imports Microsoft.VisualBasic.FileIO
Imports Microsoft.VisualBasic.Strings
Imports System.Windows.Forms.DataVisualization.Charting

Public Class Form1

  Friend Enum DistrTypes
    Normal
    Uniform
    Triangle
  End Enum

  Friend Enum DistrModes
    Continuous
    Discrete
  End Enum

  Friend NumPnts As Integer                   'number of random number samples to generate
  Friend NumStates As Integer                 'number of states for discrete random generator
  Friend StateWidth As Double                  'separation between states

  Friend seed As Integer = 0                  'seed for random number generator
  Friend DistrType As DistrTypes              'Uniform or normal
  Friend DistrMode As DistrModes               'continuous over applicable range, or discrete
  Friend StndDev As Double                    'standard deviation for the distribution

  Friend flagIsRandWalk As Boolean             'boolean to indicate if random walk or not

  Friend Generator As RandomNumGenerator
  Friend Samples As SamplesLists
  Friend AVAR() As Double                     'array for calculating Allan Variance and Deviation

  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' === set up the charts
    Chart1.Legends("Legend1").Enabled = False
    Chart2.Legends("Legend1").Enabled = False
    Chart1.Series("Series1").ChartType = SeriesChartType.Line
    Chart2.Series("Series1").ChartType = SeriesChartType.Line
    Chart2.Series("Series1").Color = Color.Black
    Chart2.Series("Series2").ChartType = SeriesChartType.Line
    Chart2.Series("Series2").Color = Color.Blue
```

```
    Chart1.ChartAreas("ChartArea1").AxisX.LabelStyle.Format = "{0.00e+00}"
    Chart2.ChartAreas("ChartArea1").AxisX.IsLogarithmic = False              'cannot start in log mode
    Chart2.ChartAreas("ChartArea1").AxisY.IsStartedFromZero = True           'looks better at startup

    Chart1.Series("Series1").Points.AddY(3)              'Add a point so that the chart will appear
    Chart2.Series("Series1").Points.AddY(3)              'Add a point so that the chart will appear

    Chart1.Visible = True
    Chart1.Show()
    Chart2.Visible = True
    Chart2.Show()

    GetRandomParms()                                     'read parameters from User Interface: Form1
    Generator = New RandomNumGenerator(seed, StndDev, NumStates, StateWidth)
    GC.Collect()                                         'can remove this line
    Samples = New SamplesLists(NumStates, StateWidth)    'object to store random numbers
End Sub

''' <summary>
''' Generates random numbers using 'Generator' and stores in 'Samples'. The parameters
''' such as seed, StndDev, NumStates, StateWidth are obtained from GetRandomParm().
''' </summary>
Private Sub BtnGen_Click(sender As Object, e As EventArgs) Handles btnGen.Click
    Try
        GetRandomParms()                                 ' Gets any updates to the parameters for Generator
        Generator = New RandomNumGenerator(seed, StndDev, NumStates, StateWidth)
        GC.Collect()

        Samples = New SamplesLists(NumStates, StateWidth, flagIsRandWalk)   ' random num repository

        ' Generate and store random numbers acccording to the mode and type
        If DistrType = DistrTypes.Normal And DistrMode = DistrModes.Continuous Then
            For i As Integer = 0 To NumPnts - 1 : Samples.Y = Generator.NextNormalContinuous : Next i
        ElseIf DistrType = DistrTypes.Normal And DistrMode = DistrModes.Discrete Then
            For i As Integer = 0 To NumPnts - 1 : Samples.Y = Generator.NextNormalDiscrete : Next i
        ElseIf DistrType = DistrTypes.Uniform And DistrMode = DistrModes.Continuous Then
            For i As Integer = 0 To NumPnts - 1 : Samples.Y = Generator.NextUniformContinuous : Next i
        ElseIf DistrType = DistrTypes.Uniform And DistrMode = DistrModes.Discrete Then
            For i As Integer = 0 To NumPnts - 1 : Samples.Y = Generator.NextUniformDiscrete : Next i
        End If

        PlotDistribution(Samples)                        ' plots distribution: the frequency plot/histogram

    Catch ex As Exception
        MsgBox("Generate Random Numbers; btn" + vbCrLf + ex.ToString, , "Error")
    End Try

End Sub

''' <summary>
''' gets parameters required to generate random numbers including mode, type
''' seed, numPnts, stateWidth, numStates
''' </summary>
Friend Sub GetRandomParms()
    Try
        seed = Integer.Parse(tbxSeed.Text)               ' need to seed random number gen.

        If rbNormal.Checked Then
```

```
            DistrType = DistrTypes.Normal
        ElseIf rbUnif.Checked Then
            DistrType = DistrTypes.Uniform
        End If

        If rbCont.Checked Then
            DistrMode = DistrModes.Continuous
        ElseIf rbDiscr.Checked Then
            DistrMode = DistrModes.Discrete
        End If

        NumPnts = Integer.Parse(tbxNumPnts.Text)            ' number of random numbers to be generated
        If NumPnts < 1 Then
            NumPnts = 1
            tbxNumPnts.Text = "1"
        End If

        NumStates = Math.Abs(Integer.Parse(tbxNumDisSts.Text))  ' number states, number Histogram bins
        If NumStates < 1 Then
            tbxNumDisSts.Text = "1"
            NumStates = 1
            MsgBox("Error", , "Number states changed to the minimum of 1")
        End If

        StndDev = Math.Abs(Double.Parse(tbxStdDev.Text))              ' standard deviation for any distribution

        StateWidth = Math.Abs(Double.Parse(tbxDiscrSep.Text))         ' get state width = state separation

        flagIsRandWalk = cbxRW.Checked                                ' is it a random walk?

    Catch ex As Exception
        MsgBox("Read parameters" + vbCrLf + ex.ToString,, "Error")
    End Try

End Sub

''' <summary>
''' Plots the frequency histogram in top chart using
''' sample data from 'samples'
''' </summary>
Private Sub PlotDistribution(Samps As SamplesLists)

    Chart1.Series("Series1").Points.Clear()                   ' remove any existing samples
    Chart1.Series("Series1").ChartType = SeriesChartType.Column

    Dim Hist As PointF() = Samps.Histogram                    ' histogram x,y values for distribution

    For i As Integer = 0 To Hist.Length - 1
        Chart1.Series("Series1").Points.AddXY(Hist(i).X, Hist(i).Y)
    Next i

    Chart1.Update()
End Sub

Private Sub BtnPlotRand_Click(sender As Object, e As EventArgs) Handles btnPlotRand.Click
    Try
        PlotRand(Samples)                                     'plots the random numbers in top chart
    Catch ex As Exception
```

```
        MsgBox("Plot Random Numbers: PlotRand" + vbCrLf + ex.ToString, , "Error")
    End Try
End Sub

''' <summary>
''' plot random numbers on top chart
''' </summary>
Private Sub PlotRand(Samps As SamplesLists)

    Select Case True                                'choose either lines or points for plot
        Case rbLine.Checked
            Chart1.Series("Series1").ChartType = SeriesChartType.Line
        Case rbPnt.Checked
            Chart1.Series("Series1").ChartType = SeriesChartType.Point
    End Select

    Chart1.Series("Series1").Points.Clear()         'clears any existing points

    For i As Integer = 0 To Samps.Length - 1
        Chart1.Series("Series1").Points.Add(Samps.Yget(i)) 'could use real point of (x(i),y(i)) for real scatter plot
    Next i
    Application.DoEvents()

    Chart1.Update()
End Sub

''' <summary>
''' Click button to calculate Allan Dev/Var and plot
''' </summary>
Private Sub BtnAVAR_Click(sender As Object, e As EventArgs) Handles btnAVAR.Click
    Try
        RunAVAR(Samples)
        Call Calc2PointAVAR(Samples) 'gives quick visual of AllanDev and AllanVar
    Catch ex As Exception
        MsgBox("AVAR ADEV: RunAVAR" + vbCrLf + ex.ToString, , "Error")
    End Try
End Sub

''' <summary>
''' calculate and show two-point Allan Deviation/Variance in bottom panel
''' </summary>
Private Sub Calc2PointAVAR(Samps As SamplesLists)
    Try
        Dim NTAUmax As Integer = Integer.Parse(tbxNTAUmax.Text)
        Dim AVARtemp As Double = CalcAVAR(Samps, 1)   'nTau=1; after plotting, mult horiz axis by gate time
        tbxAvar.Text = AVARtemp.ToString("0.00e+00")
        tbxAdev.Text = Format(Math.Sqrt(AVARtemp), "0.00e+00")
    Catch ex As Exception
        MsgBox(ex.ToString,, "Error")
    End Try
End Sub

''' <summary>
''' Calculate Allan Var using CalcAVAR
''' </summary>
''' <param name="Samps"></param>
Private Sub RunAVAR(Samps As SamplesLists)
```

```
        Dim nTauMax As Integer = Math.Min(Integer.Parse(tbxNTAUmax.Text), Math.Floor(Samps.Length / 2)) 'Samps.length/2 is
max
        ReDim AVAR(nTauMax - 1)
        For k As Integer = 1 To nTauMax
            AVAR(k - 1) = CalcAVAR(Samps, k)                'calculate Allan Variance
        Next k
        ReDim Preserve AVAR(nTauMax - 1)                    'AVAR holds results; number entries = nTauMax
        Call PlotAllan(AVAR, cbxPlotDev.Checked, cbxPlotVar.Checked)
    End Sub

    ''' <summary>
    ''' Plots Allan Deviation and Variance
    ''' </summary>
    ''' <param name="AVAR"></param>
    ''' <param name="FlagPlotDev"> Set 'true' to plot Deviation </param>
    ''' <param name="FlagPlotVar"> Set 'true' to plot Variance </param>
    Private Sub PlotAllan(ByRef AVAR() As Double, FlagPlotDev As Boolean, FlagPlotVar As Boolean)

        Dim FlagZero As Boolean = True
        Chart2.Series("Series1").Points.Clear()
        Chart2.Series("Series2").Points.Clear()

        For i As Integer = 0 To AVAR.Length - 1
            If AVAR(i) = 0 Then
                AVAR(i) = AVAR(i - 1)
                If FlagZero Then MsgBox("NOTE: zero found, so previous nonzero substituted")
                FlagZero = False    'prevents the msg box from showing again
            End If
            If FlagPlotDev Then
                Chart2.Series("Series1").Points.AddXY(CDbl(i + 1), AVAR(i)) 'could use real point of (x(i),y(i)) for real scatter plot
            End If
            If FlagPlotVar Then
                Chart2.Series("Series2").Points.AddXY(CDbl(i + 1), Math.Sqrt(AVAR(i))) 'could use real point of (x(i),y(i)) for real scatter
plot
            End If
            Application.DoEvents()
        Next i

        Chart2.ChartAreas("ChartArea1").AxisY.IsStartedFromZero = False
        Chart2.ChartAreas("ChartArea1").AxisX.IsLogarithmic = True
        Chart2.ChartAreas("ChartArea1").AxisY.IsLogarithmic = True

        Chart2.Update()
    End Sub

    ''' <summary>
    ''' Saves file as text with each entry separated by CR LF
    ''' Saves Random value Y and no x. X can be reconstructed as the
    ''' integer index starting at 0.
    ''' </summary>
    Private Sub MnuFsaveSamp_Click(sender As Object, e As EventArgs) Handles mnuFsaveSamp.Click

        If Samples Is Nothing Then Exit Sub

        Try
            Dim FNameSave As String = ""
            sfd01.DefaultExt = "txt"
            sfd01.AddExtension = True
```

```
            If sfd01.ShowDialog = DialogResult.OK Then
                FNameSave = sfd01.FileName

                If FNameSave <> "" Then
                    Dim objWriter As New System.IO.StreamWriter(FNameSave)
                    For i As Integer = 0 To Samples.Length - 1
                        objWriter.WriteLine(Samples.Yget(i).ToString)           'use plain write for no vbcrlf
                    Next i
                    objWriter.Close()
                    objWriter.Dispose()
                End If
            End If

        Catch ex As Exception

            MsgBox(ex.ToString,, "Error")

        End Try

    End Sub

    Private Sub MnuFopenSamp_Click(sender As Object, e As EventArgs) Handles mnuFopenSamp.Click

        Try

            GetRandomParms()
            If Samples Is Nothing Then

                Samples = New SamplesLists(NumStates, StateWidth, False) 'note need to make sure no random walk
            Else
                Samples.Clear()
            End If

            Dim FNameOpen As String = ""
            sfd01.DefaultExt = "txt"
            sfd01.AddExtension = True

            If ofd01.ShowDialog = DialogResult.OK Then
                FNameOpen = ofd01.FileName

                If FNameOpen <> "" Then
                    Dim objReader As New System.IO.StreamReader(FNameOpen)

                    While (Not objReader.EndOfStream)
                        Samples.Y = objReader.ReadLine()
                    End While

                    NumPnts = Samples.Length
                    tbxNumPnts.Text = NumPnts.ToString
                    objReader.Close()
                    objReader.Dispose()

                    PlotRand(Samples)

                    If NumPnts >= 4 Then
                        Call Calc2PointAVAR(Samples)
                    End If
```

```
            End If
        End If

      Catch ex As Exception
        MsgBox(ex.ToString,, "Error")
      End Try

    End Sub
End Class


' =================== MODULE/CLASSES ====================
' The module can be placed on a separate page. It will be necessary
' to include: Imports Microsoft.VisualBasic.VBMath
' Consider moving the two enums to the top of the module between
' Module1 and Public Class SamplesList

Module Module1

    Public Class SamplesLists
      Private _yList As New List(Of Double)                    ' contains random numbers
      Private _xList As New List(Of Double)                    ' not used

      Private _numStates As Integer       ' number of discrete states that can be selected by random num generator
      Private _widthState As Double       ' Separation between states, which can be identified as the state/bin width

      Private _IsRandomWalk As Boolean = False
      Private _RandWalkAccumulator As Double = 0  'holds sum of previous random numbers

      ''' <summary>
      ''' Parameterless NEW for class where parameters will be added later.
      ''' Includes dummy values for some params.
      ''' </summary>
      Friend Sub New()
        _numStates = 10                                        'place holder
        _widthState = 0.1                                      'place holder
        _IsRandomWalk = False
        _RandWalkAccumulator = 0
      End Sub

      ''' <summary>
      ''' Instantiate new data collection. Can use for continuous and discrete
      ''' Required parameters for histogram are included
      ''' </summary>
      ''' <param name="NumStates"> For discrete distributions </param>
      ''' <param name="WidthState"> For discrete distributions </param>
      Friend Sub New(NumStates As Integer, WidthState As Double, Optional IsRandomWalk As Boolean = False)
        _numStates = NumStates
        _widthState = WidthState
        _IsRandomWalk = IsRandomWalk
        _RandWalkAccumulator = 0
      End Sub

      ''' <summary>
      ''' resets stored variables
      ''' </summary>
      Friend Sub Clear()
```

```
    _yList.Clear()
    _xList.Clear()
    _RandWalkAccumulator = 0
End Sub

''' <summary>
''' clears the accumulation of previous steps for a random walk
''' </summary>
Friend Sub ClearRandWalkAccum()
    _RandWalkAccumulator = 0
End Sub

''' <summary>
''' manually set boolean to indicate the random number
''' should be added to the accumulated value
''' </summary>
Friend Property IsRandomWalk As Boolean
    Set(value As Boolean)
        _IsRandomWalk = value
    End Set
    Get
        Return _IsRandomWalk
    End Get
End Property

''' <summary>
''' adds a double-type number to the list of samples
''' </summary>
Friend WriteOnly Property Y As Double
    Set(value As Double)
        If _IsRandomWalk Then
            _RandWalkAccumulator += value
            _yList.Add(_RandWalkAccumulator)
        Else
            _yList.Add(value)
        End If
    End Set
End Property

''' <summary>
''' returns a random number from the list
''' </summary>
''' <param name="i"> index into the list </param>
Friend Function Yget(ByVal i As Integer) As Double
    Return _yList(i)
End Function

''' <summary>
''' number of discrete states for a discrete distribution
''' </summary>
''' <returns></returns>
Friend Property NumStates As Integer
    Set(value As Integer)
        _numStates = value
    End Set
    Get
        Return _numStates
    End Get
```

```
End Property

''' <summary>
''' Width of discrete states = separation between them
''' States considered like bins adjacent to each other
''' </summary>
Friend Property WidthState As Double
   Set(value As Double)
      _widthState = value
   End Set
   Get
      Return _widthState
   End Get
End Property

''' <summary>
''' Histogram: frequency plot for random numbers
''' </summary>
Friend Function Histogram() As PointF()
   Return Histogram(_numStates, _widthState)
End Function

''' <summary>
''' length is the same as count for the list = number of entries
''' </summary>
Friend Function Length() As Integer
   Return _yList.Count
End Function

''' <summary>
''' Number of 'random numbers' in each range/interval/bin
''' </summary>
''' <param name="numBins"></param>
''' <param name="widthBins"></param>
''' <returns></returns>
Friend Function Histogram(ByVal numBins As Integer, ByVal widthBins As Double) As PointF()

   Dim i As Integer = 0                                  ' i =  0, 1, ... _numStates-1,
   Dim Left(numBins - 1) As Double                        ' left endpoints of the bin,
   Dim Right(numBins - 1) As Double                      ' right endpoints of the bin
   Dim MIDPNT(numBins - 1) As Double                      ' midpoint of bin
   Dim HalfWidth As Double = widthBins / 2.0             ' half-width of bin
   Dim Histgrm(numBins - 1) As PointF                    ' array of (x,y) for histogram
   Dim flagIntrvlFound As Boolean = False

   '
   For j As Integer = 0 To numBins - 1
      MIDPNT(j) = _widthState * CDbl(2 * j - numBins + 1) / 2.0
      Left(j) = MIDPNT(j) - HalfWidth
      Right(j) = MIDPNT(j) + HalfWidth
   Next j

   'load bin midpoints for the horizontal axis
   For j = 0 To numBins - 1 : Histgrm(j).X = CSng(MIDPNT(j)) : Next j

   For j As Integer = 0 To _yList.Count - 1              ' loop for random numbers: _yList
      flagIntrvlFound = False
      i = 0                              ' index for bins
```

```
            Do While (flagIntrvlFound = False And i < numBins)  ' loop for bins
                If (Left(i) <= _yList(j) And _yList(j) < Right(i)) Then
                    flagIntrvlFound = True
                    Histgrm(i).Y += 1                                   ' increment number in bin
                Else
                    i += 1
                End If
            Loop    'loop through bins
        Next j      'loop through all random numbers

        Return Histgrm
    End Function

End Class

''' <summary>
''' Generate uniform/Gaussian random numbers for continuous or discrete
''' for discrete need to set numStates and stateWidth
''' Note: inherits Random so can also use new(), new(seed) but the
''' stdDev, numStates and stateWidth will not be properly set for
''' the various non-"Random" methods - must use properties to set
''' </summary>
Public Class RandomNumGenerator

    Inherits Random

    Private _StdDev As Double = 1                           'for uniform and normal
    Private _numStates As Integer = 10                      'for discrete cases
    Private _StateWidth As Double = 1                       'for discrete cases

    ' VARIABLES TO REDUCE COMPUTATION LOAD
    Private HalfWidth As Double = 0.5                       'HalfWidth = stateWidth/2
    Private Left(_numStates - 1) As Double                  'left endpoints of the bars,
    Private Right(_numStates - 1) As Double                 'right endpoints of the bars
    Private MIDPNT(_numStates - 1) As Double                'midpoint of (left,right)

    ''' <summary> instantiates a traditional random number generator "Random" </summary>
    Friend Sub New()
        MyBase.New()
    End Sub

    ''' <summary> instantiates a traditional random number generator "Random" </summary>
    ''' <param name="seed"></param>
    Friend Sub New(seed As Integer)
        MyBase.New(seed)
    End Sub

    ''' <summary>
    ''' NEW instantiates all distributions (continuous and discrete, uniform and normal)
    ''' For discrete, enter numDiscreteStates and StateWidth.
    ''' All distributions center on zero.
    ''' </summary>
    ''' <param name="seed"></param>
    ''' <param name="StdDev"></param>
    ''' <param name="numDiscreteStates"></param>
    ''' <param name="StateWidth"></param>
    Public Sub New(seed As Integer, StdDev As Double, Optional numDiscreteStates As Integer = 21, Optional StateWidth As
Double = 0.1)
```

```
    MyBase.New(seed)
    _StdDev = StdDev
    _numStates = numDiscreteStates
    _StateWidth = StateWidth
    ReduceWorkLoadVariables()                    'Variables to reduce workload for NormalDiscrete
End Sub

''' <summary>
''' Standard Deviation to be used for any of the distributions for scaling.
''' </summary>
''' <returns></returns>
Friend Property StdDev() As Double
    Set(value As Double)
        _StdDev = value
    End Set
    Get
        Return _StdDev
    End Get
End Property

''' <summary>
''' number of discrete states
''' </summary>
''' <returns></returns>
Friend Property NumStates() As Integer
    Set(value As Integer)
        _numStates = value
        ReduceWorkLoadVariables()                'update those variables repeatedly used in NormalDiscrete
    End Set
    Get
        Return _numStates
    End Get
End Property

''' <summary>
''' bin width
''' </summary>
''' <returns></returns>
Friend Property StateWidth() As Double
    Set(value As Double)
        _StateWidth = value
        ReduceWorkLoadVariables()                'update those variables repeatedly used in NormalDiscrete
    End Set
    Get
        Return _StateWidth
    End Get
End Property

''' <summary>
''' initializes variables to reduce workload for
''' discretizing/quantizing the Normal Distribution
''' interval = (left,right)
''' </summary>
Private Sub ReduceWorkLoadVariables()

    HalfWidth = _StateWidth / 2.0

    ReDim Left(_numStates - 1)                   'left endpoints of the bars,
```

```
    ReDim Right(_numStates - 1)                          'right endpoints of the bars
    ReDim MIDPNT(_numStates - 1)                         'midpoint of (left,right)

    For j As Integer = 0 To _numStates - 1
        MIDPNT(j) = _StateWidth * CDbl(2 * j - _numStates + 1) / 2.0
        Left(j) = MIDPNT(j) - HalfWidth
        Right(j) = MIDPNT(j) + HalfWidth
    Next j

End Sub

''' <summary>
''' generates a uniformly distributed random number in the range (-1,1)
''' </summary>
''' <returns></returns>
Private Function NextUniformN1P1() As Double

    Dim randNum As Double = 0.000000000000000
    Do                                                    'will want (-1,1)
        'randNum = Generator.NextDouble()                 'produces [0 to 1)
        randNum = Me.NextDouble()                        'produces [0 to 1) but need (0 to 1)
    Loop While (randNum = 0.000000000000000)

    randNum = (randNum - 0.5) * 2.0  'want (-1,1) to feed to normal distr calculation
    Return randNum
End Function

''' <summary>
''' generrates the next random value conforming to a
''' continuous normal distribution
''' </summary>
Friend Function NextNormalContinuous() As Double
    Dim randNum As Double = NextUniformN1P1()
    randNum = Math.Sqrt(2) * _StdDev * ErfInv(randNum) '
    Return randNum
End Function

''' <summary>
''' generates the next random number corresponding to a
''' uniform continuous distribution
''' </summary>
''' <returns></returns>
Friend Function NextUniformContinuous() As Double
    Return (Math.Sqrt(3) * _StdDev * NextUniformN1P1())
End Function

''' <summary>
''' Generates: for 1 state: only zero; 2state: -0.5, +0.5; 3state: -1, 0, +1 ...
''' </summary>
''' <returns></returns>
Private Function NextUniformHalfInt() As Double
    'numSt = 2 => 0,1 ;  numSt = 3 => 0, 1, 2
    Dim rndNum As Double = CType(MyBase.Next(_numStates), Double) 'this will be in the interval [0, a=_numStates-1]
    Dim MidNumber As Double = CType(_numStates - 1, Double) / 2
    rndNum -= MidNumber 'rndNum in ( -a/2 , a/2 ) still need to scale
    Return rndNum
End Function
```

```
    ''' <summary>
    ''' Generates: for 1 state: 0; 2state: -0.5W, +0.5W; 3state: -W, 0, +W ...
    ''' must set statewidth, numStates, stdDev before using
    ''' </summary>
    Friend Function NextUniformDiscrete() As Double
        Return NextUniformHalfInt() * _StateWidth              'scales by _StateWidth
    End Function

    ''' <summary>
    ''' generates next random number consistent with
    ''' a normal distribution with discrete values
    ''' </summary>
    Friend Function NextNormalDiscrete() As Double
        Dim randNum As Double = 0                          'includes standard deviation
        Dim i As Integer = 0
        Dim flagIntrvlFound As Boolean = False

        'place a random number continuously distributed into a bin
        While (Not flagIntrvlFound)
            randNum = NextNormalContinuous()               ' includes standard deviation
            i = 0                                       ' bin index
            While ((Not flagIntrvlFound) And (i < _numStates))      ' want to return the midpoint of the found interval
                If ((Left(i) <= randNum) And (randNum < Right(i))) Then
                    flagIntrvlFound = True
                Else
                    i += 1
                End If
            End While 'loop through intervals or until found
        End While 'get another random number then loop through intervals again

        Return MIDPNT(i)
    End Function

    ''' <summary>
    ''' converts a uniformaly distribute x on (-1,1) to a normal with average zero
    ''' </summary>
    Private Function ErfInv(ByVal x As Double) As Double
        ' algorithm from https://stackoverflow.com/questions/27229371/inverse-error-function-in-c
        ' meaning of sqrtf at https://en.cppreference.com/w/c/numeric/math/sqrt means float argument
        ' meaning of logf at https://en.cppreference.com/w/c/numeric/math/log means float argument

        Dim tt1, tt2, lnx, sgn As Double
        sgn = If((x < 0), -1.0F, 1.0F)
        x = (1 - x) * (1 + x)
        lnx = Math.Log(x) 'was logf(x)
        tt1 = 2 / (3.1415 * 0.147) + 0.5F * lnx 'math.pi
        tt2 = 1 / (0.147) * lnx
        Return (sgn * Math.Sqrt(-tt1 + Math.Sqrt(tt1 * tt1 - tt2))) 'was sqrtf
    End Function

End Class

''' <summary>
''' Calculates the Allan Variance
''' </summary>
Friend Function CalcAVAR(Dat As SamplesLists, ByVal nTau As Integer) As Double

    Dim i As Integer = 0
```

```
    Dim j As Integer = 0

    Dim N As Integer = Math.Floor(Dat.Length / nTau)

    Dim Ave1 As Double = 0
    Dim Ave2 As Double = 0
    Dim Ave As Double = 0

    For i = 0 To N - 2

      Ave1 = 0
      Ave2 = 0
      For j = 0 To nTau - 1
        Ave1 += Dat.Yget(nTau * i + j)
        Ave2 += Dat.Yget(nTau * (i + 1) + j)
      Next j
      'DIV BY NTAU, SUBTR AND SQUARE  BEFORE ADDING TO AVE
      Ave1 /= nTau
      Ave2 /= nTau
      Ave += (Ave2 - Ave1) ^ 2
      Application.DoEvents()
    Next i

    Return (Ave / (2 * (N - 1)))
  End Function

End Module
```

Topic A7.3.3: Comments on the Form1 Source Code

There are several aspects to the programming: (i) Design of the GUI as in Section A7.2. (ii) VB.net is fully object oriented (as are other .net languages) and so the program must define and instantiate the objects of interest. (iii) Given that VB.net is event driven, most of the computation will take place in response to events. (iv) The program can be prepped for an installer MSI or for use as a stand-alone EXE file. The EXE file doesn't require any additional effort.

The basic operation of the GUI was discussed in Section A7.1. Now briefly consider the code on Form1 which has the two main sections consisting of the Form1 functional code and a Module toward the bottom of the page. The module contains the two primary classes of 'SamplesLists' and 'RandomNumGenerator'. The SamplesLists provide an internal List for storing the random numbers produced by the Random Number Generator and also histogram methods for plotting the distribution of the stored random numbers. Additionally, it contains the code needed for the random walk. The class RandomNumberGenerator inherits from the Microsoft Random class and therefore allows use of the members of the Random class plus provide extensions to continuous, discrete, normal, and uniform distributions. The Form1 code primarily supports the buttons and charts and interfaces with module1.

FORM1 CODE:

1. The top of Form1 provides some global declarations. Enum DistrTypes refer to the normal and uniform type distribution while Enun DistrModes refers to either the continuous or discrete versions. The continuous mode generates random numbers in a continuous range of real values. The discrete mode generates specific random numbers in one of the sets
   { … -1.5W, -0.5W, 0.5W, 1.5W … } or { … -2W, -W, 0, W, 2W … }

depending on whether the number of states, denoted by NumStates, is even or odd, respectively. W is the state width, denoted StateWidth, and represents the distance between the possible values of the random numbers in the sets. For the histograms, the StateWidth and NumStates give the width of the bins and the number of bins respectively, although the histogram parameters can be separately designated if so needed. The Samples object uses a List to store the random numbers which has an 'add' property and doesn't need to keep explicit track of the number of elements in the list. The variable NumPnts represents the number of samples to be generated by the random number generator.

2. The Form1_Load routine executes when the software loads. This routine initializes the two charts to plot the histogram , random numbers, Allan Deviation and Allan Variance. The routine also instantiates Generator as a RandomNumGenerator and Samples as a SamplesLists. The parameters for Generator and Samples come from the subroutine GetRandomParms which simply scans the values in the GUI controls and then stores them in their respective variables.
3. The BtnGen subroutine is the event handler for the GUI button labelled 'Generate Random Numbers'. Clicking the button causes the subroutine to scan the GUI parameters (using the GetRandomParms subroutine) and defines the corresponding Generator. Notice that the seed for the random number generator can be manually entered on the GUI; the same seed can be reused to generate the same random numbers in case something looks interesting. The BtnGen event handler stores the random numbers using the property 'Samples.Y'. The random numbers are generated by the appropriate function such as 'Generator.NextNormalContinuous' which returns a random number consistent with a continuous normal distribution. The event handler calls the subroutine PlotDistribution on the Form1 page.
4. Recall the top chart on the GUI, named Chart1, can plot either the distribution (i.e., histogram) or the sequence of random numbers stored in Samples. The subroutine 'PlotDistribution' configures Chart1 and then dimensions an array Hist to receive the bins from the Samples Histogram, which does all the required bin calculations. While the List proper 'count' could be used for the total number of random numbers stored in Samples, the class provides a property 'length' to give the same information. The histogram is then transferred to the chart using the chart 'addXY' subroutine. Notice the X values of Hist are the midpoints of the bins while the Y values provide the number of random numbers falling in a particular bin. By the way, notice the floating points pointF(x,y) form the elements of the Hist array.
5. The event handler BtnPlotRand processes the click of the GUI 'Plot Rand Nums' by calling the subroutine 'PlotRand'
6. The subroutine PlotRand configures Chart1 to either plot points or lines which connect the points. The sequence of points come from the Samples object by using the function Yget(i) which returns element i of the internal list.
7. The event handler BtnAVAR calls the subroutine RunAVAR to plot the Allan Deviation and Variance and the event handler also calls the Calc2PointAVAR to show the numbers in the textboxes on the GUI next to the blue Allan Deviation and black Allan Variance. By the way, the statement 'Dim NTAUmax…' is not needed for the Calc2PointAVAR subroutine and can be deleted.
8. The PlotAllan subroutine plots either or both the Allan Deviation and the Allan Variance depending on the which GUI checkboxes have been checked. One idiosyncrasy concerns the problem of plotting zero on a log plot – it doesn't plot. If a y-value should happen to be zero, the PlotAllan subroutine substitutes the previous non-zero y-value. The flag is used to indicate the substitution the first time that it happens. If you don't care, eliminate all statements with the 'FlagZero'. Chart2 must be changed to Logarithmic at the end to prevent errors.

9. The mnuFsaveSamp and mnuFopenSamp event handlers save data to a file and read data from a file, respectively. The files have the .txt extension and consist of ASCII text and the text numbers are separated by CR (LF). Only the y are saved and only y values can be read to a Samples object. As a note, the handler MnuFopenSamp has the line

```
Samples = New SamplesLists(NumStates, StateWidth, False)
```

Notice the `False` refers to whether or not the Samples object should treat the read-in data as a random the walk. It is set to 'no' because otherwise the Samples object were perform random walk operations on the data. The data is already either saved a random walk or its not and should not be modified.

MODULE1 CODE: SamplesLists Class

10. At the top of the SamplesLists class, private variables a declared for internal use but they can be accessed by the properties and routines defined further down. The _yList is a List(of double) used to store the random numbers. The _xList is not used and can be deleted. The _numStates refers to the number of discrete states for discrete distributions whereas _widthState refers to the distance between the adjacent discrete states. Sometimes the _widthState is represented by W or by stateWidth. The discrete states are centered about zero as discussed above in #1. The variable _RandWalkAccumulator stores the sum of previous y values (i.e., steps) for the random walk. For example, consider a person walking in the manner of a random walk. The first step might be +1 then the accumulator will be 0+1. The next step might again be +1 so the person is at y=2 which comes by adding together the present +1 and the accumulator value of +1. The present +1 is added to the accumulator to get +2 in preparation for the next step (and so on).
11. The class has two different NEW subroutines to instantiate new SamplesLists objects. The top NEW assigns values to some variables as place holders until the properties can be used. Probably this top NEW can be deleted for the purposes of the software. The second NEW includes some parameters for the main program.
12. The SamplesLists class then shows some properties that can be used. Notice the Property Y adds an incoming random number to the internal List named _yList. Notice how it implements the random walk as discussed in item #10 above. The function Yget returns element #i from the List. The function Histogram returns an array of PointF which is an ordered pair (x,y) of floating point numbers; the array is produced by the function Histogram found further down. The length function returns the number of items in the list _yList.
13. The Histogram(numBins, widthBins) sorts the samples into bins and returns an array of PointF points (x,y) where x,y are floating point numbers. Here x will be the midpoint of the histogram bin and y will be the number of items in the bin. The bins might also be called intervals because the left and right endpoints of the bin form the an interval ($Left(i)$, $Right(i)$) for bin #i. The midpoint of the interval is labeled as MIDPNT(i). So for each sample j in _yList, the function must find the interval/bin that contains sample _yList(j). First to reduce the work load on the search algorithm, the subroutine configures the intervals/bins and the midpoints:

```
For j As Integer = 0 To numBins - 1
 MIDPNT(j) = _widthState * CDbl(2 * j - numBins + 1) / 2.0
 Left(j) = MIDPNT(j) - HalfWidth
 Right(j) = MIDPNT(j) + HalfWidth
Next j
```

Next a search algorithm must find the interval corresponding to a sample in _yList:

```
For j As Integer = 0  To _yList.Count - 1              ' loop for random numbers: _yList
```

```
            flagIntrvlFound = False
            i = 0                                   ' index for bins
            Do While (flagIntrvlFound = False And i < numBins)  ' loop for bins
              If (Left(i)  <= _yList(j)  And _yList(j)  <  Right(i))  Then
                flagIntrvlFound  =  True
                Histgrm(i).Y += 1                   ' increment number in bin
              Else
                i += 1
              End If
            Loop                                    'loop through bins
    Next j                                          'loop through all random numbers
```

The code above iterates through the sample list _yList using index j starting with interval i=0. If the sample #j is found to be contained in interval #i using

```
If (Left(i)  <= _yList(j)  And _yList(j)  <  Right(i))  Then
```

then the number in the bin is incremented by 1 using Histgrm(i) += 1, and a flag is set to exit the Do While loop. The process continues until all of the samples have been tested using the For j loop. On the other hand, if a given sample does not fall within any of the bin intervals, the condition i<numBins will cause the Do While loop to terminate and the next i will be considered. For this reason, one cannot be 100% sure the histogram includes all of the points in _yList.

MODULE1 CODE: RandomNumGenerator Class

14. Notice that the class inherits the Microsoft Random class even though there aren't that many methods in the Random class. However, it does mean that, with the RandomNumberGenerator class can be viewed as an extension of the Random class capable of handling Normal distributions. For this reason, three NEW subroutines are available for instantiating objects such as 'Generator' used in the Form1 code. The first NEW simply passes the instantiation to the parent class Random and the Generator will use a seed based on the system time keeping. The second NEW also passes the instantiation to the parent Random class but specifies a Seed. The parent class methods are then directly available such as Generator.next etc. The third NEW once again appeals to the Ranom NEW but now includes those parameters needed for discrete, continuous, Normal and Uniform distributions.
15. The class defines a slew of private variables with meanings similar to that in the SamplesLists class except here, they refer to the discrete states for the probability distributions and not specifically for the historgram previously discussed. However, it should be pointed out that when the two sets of variables have the same value, life becomes much easier.
16. A number of properties, subroutines and functions are included to access the private variables. Of special interest, notice some of them have a call to the subroutine "ReduceWorkLoadVariables". For discrete distributions, the generator should produce a number that can be one of several fixed values. The section previously discussed, for example, that numStates=5 and statedWidth=W for a discrete distribution should produce one of the following numbers

$$-2W, -W, 0, +W, +2W$$

To do this, a random number produced by a continuous distribution must be matched to an interval with left-hand and right-hand end points and a midpoint. The subroutine uses the following to calculate the left, right and mid-points for interval #j.

```
For j As Integer = 0 To _numStates - 1
```

```
        MIDPNT(j) = _StateWidth * CDbl(2 * j - _numStates + 1) / 2.0
        Left(j) = MIDPNT(j) - HalfWidth
        Right(j) = MIDPNT(j) + HalfWidth
    Next j
```

These arrays are defined now and only once so that repeated random number generation won't need to do it each time the discrete function is invoked.

17. The function NextUniformN1P1 generates a random number in the range -1 to +1 corresponding to the uniform distribution.
18. Two functions should be combined: NextUniformHalfInt and NextUniformDiscrete rather than keep the separate. The second only applies a scaling factor. After combining, delete the HalfInt one.
19. The function NextNormalDiscrete searches through the intervals and returns the midpoint of the interval which corresponds to the random number produced by the continuous Normal distribution. The procedure is very similar to #13 above.

Section A7.4 References

[A7.1] Review of the Uniform Distributions
https://www.statisticshowto.datasciencecentral.com/uniform-distribution/

[A7.2] Another Review of Uniform Distributions
https://www.ucd.ie/msc/t4media/Uniform%20Distribution.pdf

[A7.3] Review of Normal Distributions
https://statisticsbyjim.com/basics/normal-distribution/

[A7.4] Source code, listings, downloads, flash drives, preprogrammed microcontroller:
(A) Chapter 10 and Appendix 6 provide the source code for the C/C++ program
(B) Appendices 7-9 provide the PC software
(C) Check EBay.com in connection with the title of this book: software and preprogrammed MEGA328P
(D) check www.AngstromLogic.com for current offerings, or write to Sales@AngstromLogic.com
Keep in mind that Angstrom Logic, LLC is not setup/able to answer questions on the book or software.
(E) Check Amazon in connection with the title of this book.
(F) Note: The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the words) and the contents are copyrighted with all rights reserved (also see front copyright page).
(G) The Original Works are provided under this License on an "AS IS" BASIS and WITHOUT WARRANTY, either express or implied, including, without limitation, the warranties of non-infringement, merchantability or fitness for a particular purpose. THE ENTIRE RISK AS TO THE QUALITY OF THE ORIGINAL WORK IS WITH YOU. This DISCLAIMER OF WARRANTY constitutes an essential part of this License. No license to the Original Works is granted by this License except under this disclaimer. Limitation of Liability: Under no circumstances and under no legal theory, whether in tort (including negligence), contract, or otherwise, shall the Licensor be liable to anyone for any indirect, special, incidental, or consequential damages of any character arising as a result of this License or the use of the Original Work including, without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses.

Appendix 8: FA-2 File Conversion Utility

The present appendix lists the software for the utility that converts the text output file from the FA-2 frequency counter and Termite terminal software into a text file or multiple text files suitable for Allan Deviation plotters, Excel or other plotting routines. The appendix first reviews the Graphical User Interface (GUI) discussed in Chapter 9. It then provides the listing for the Form1.designer.vb and the Form1 functional code. As discussed in Appendix A7, there are a number of different procedures for reconstructing the software in Microsoft Visual Studio (MVS) but the listings for both the Designer and Form1 enable a person to copy it directly, or drag-drop control using the designer parameters. The software should also be available with the installer .MSI file or already compiled in the .exe file [A8.1]. We first repeat the information from Section 9.9 for completeness.

Section A8.1: Graphical User Interface (GUI) for FA2Convert

As mentioned in Section 9.1, the combination of the FA-2 frequency counter and the Termite software produces a text file with entries of the form

hh:mm:ss: * F:0010000023.123456789 (A8.1)

The "FA-2 File Converter" software separates the time and frequency parts and then opens new text files with the time in seconds and the frequency as the raw frequency (given by 'F:' in A8.1 above) or as the fractional frequency or as the error frequency. The text data can be written in a single file or in multiple files as will be discussed. Probably it would be much more convenient to add a converter to the FA-2 unit or maybe to software that directly plots the Allan Deviation. Apparently the software Lady Heather has been modified to accept data from the FA-2 Frequency Counter. In any event, the 'FA-2 File Converter' can be used to create files that can be read by a variety of other software including Excel, the EZL plotting routines and the ADEV software discussed in a previous chapter. The software serves its purpose but the reader familiar with programming can improve it as needed.

The first version of the GUI for the 'FA-2 File Converter' appears in Figure A8.1 although other versions might become available that includes graphs, RS232 and ADEV options. The basic layout of the user interface (UI) consists of the various sections:

1. The 'File Structure' group determines the number of output text files and the order of the data as either Time,Freq or Freq,Time when all the data occupies a single file.
2. The 'Frequency Format' sets the format for the frequency in the output file as Fractional Frequency, Error Frequency or Raw (refer to Chapter 4 for the definitions).
3. The textbox 'Nominal Frequency' needs to be set to the expected frequency output from the device connected to the FA-2 front input (10MHz for the RFS). The textbox 'Gate Set' needs to be set to the Gate Time of the FA-2. The textboxes for the average frequency and Cycle time show the averaged frequency and the Cycle Time (i.e., gate time plus dead time), respectively, deduced from the data file. No effort was made to tailor the number of decimal places.
4. The button named "SELECT FA2 TXT" selects a text file originating from the FA-2 and Termite software. The top, right side textbox shows the selected filename and the bottom, right side textbox shows the raw content of the file. Note the 'raw' frequency appears after the 'F:' in the lower textbox for the figure panel (a).

5. The 'RUN' button modifies the Time and Frequency content as shown in the lower right textbox in the figure panel (b). The time offset of 2min 21sec (i.e., starting time in panel a) is subtracted from each time entry and so the time sequence would start at zero. Given the first frequency entry occurs after the gate time of 10 seconds, the gate time is added to each member which places the first time at 10 seconds. The frequency shown is the fractional frequency of that in (a) according to

   (Freq Raw – Nominal) / Nominal

   The error frequency would have the form (Freq Raw – Nominal).
6. The 'Save File' button saves the modified data shown in the lower textbox of panel (b) according to the selected radio button in the 'File Structure' group box.

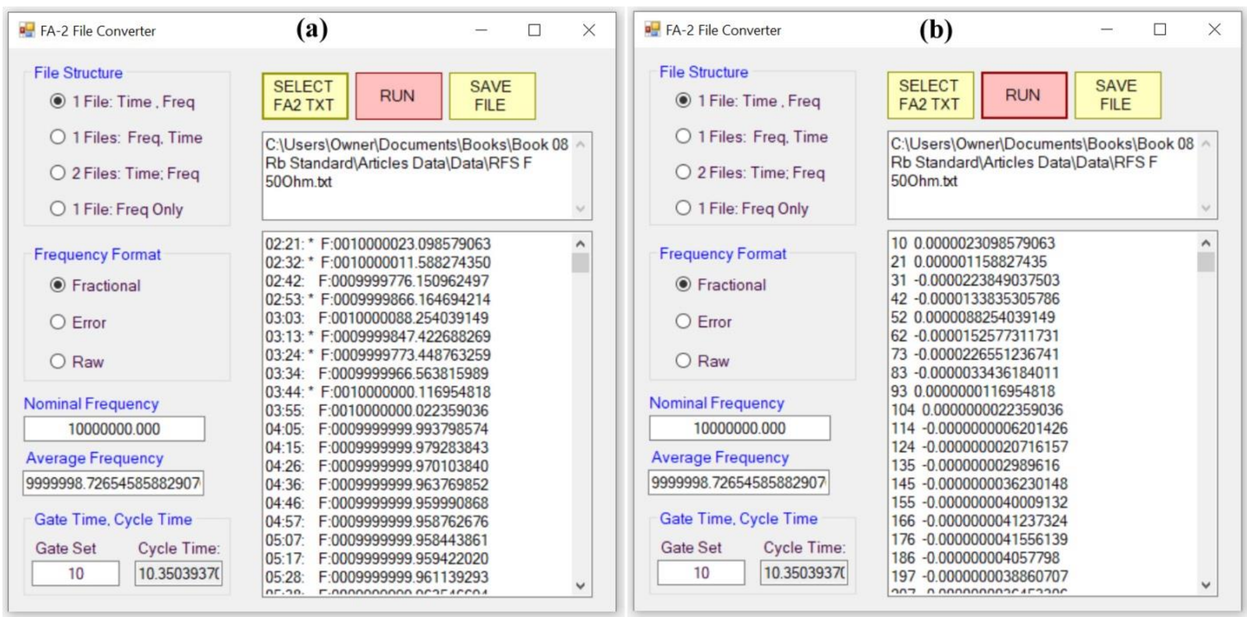
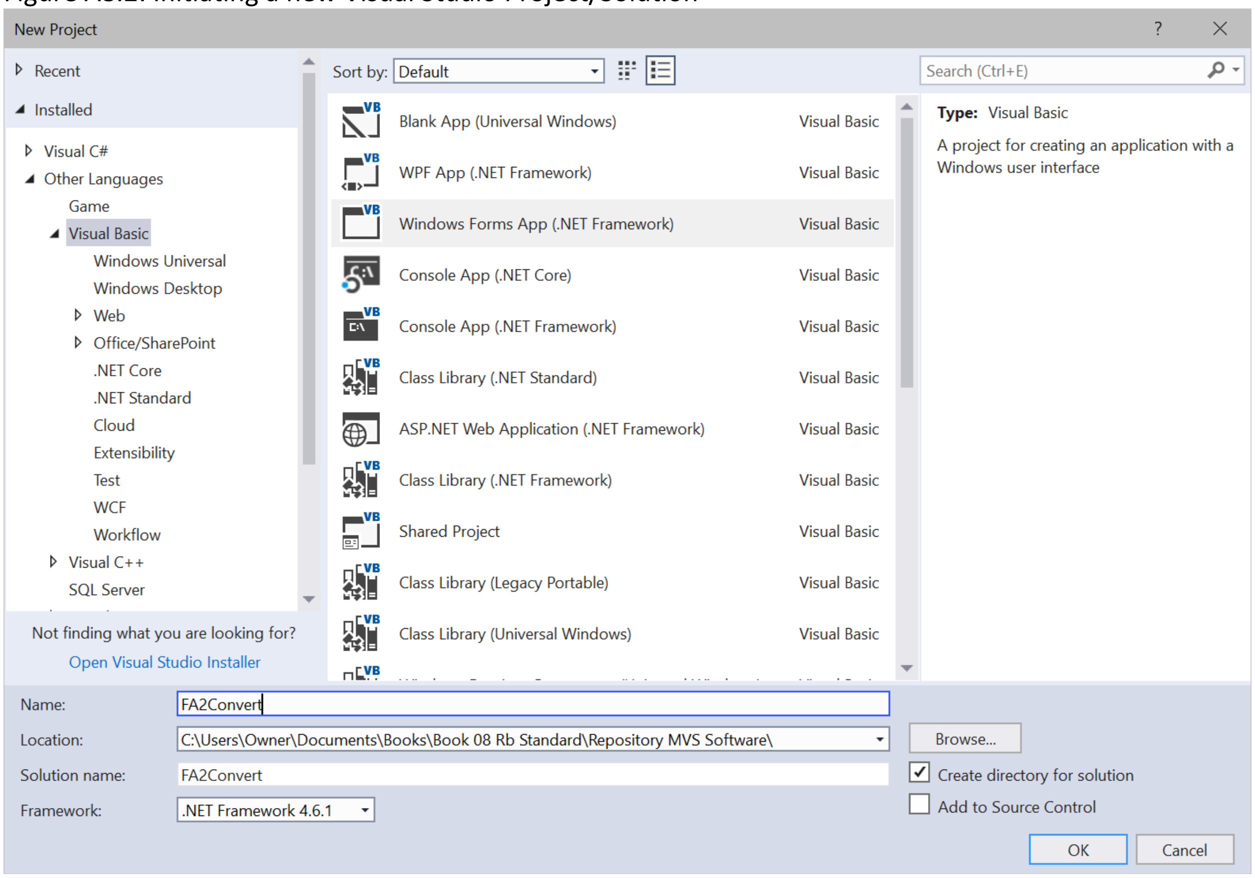


Figure A8.1: (a) The user interface after selecting the file 'RFS F 50Ohm.txt' shown in the upper textbox. The raw content appears in the lower right textbox with time on the left and frequency on the right. (b) The user interface after clicking the RUN button. The offset of 2min 21sec has been subtracted from the time and then the 10sec gate time added.

Section A8.2: Form1.Designer.vb for FA2Convert

The present section contains the design code for the FA2Convert software; it applies the same procedures to FA2Convert as for AllanDev in the previous appendix except for the charts. As previously mentioned, the software is written in Visual Basic .NET (VB.NET) rather than C#. The VB does not need the single characters such as '{' and '}' and ';'. These single tokens can be easily missed while copying. The VB.NET can be easily translated to C# if desired by using free online translators. The program listed here uses the older .net 'Forms' rather than the Universal or Windows Presentation Foundation. As with all Microsoft Visual Studio projects/solutions, the programmer has the option of either dragging/dropping controls/tools onto a given form so as to construct the Graphical User Interface (GUI), or else developing both the visual and functional aspects entirely within the code behind the displayed form. The drag-and-drop method is cumbersome to list in a book but is probably the better method for learning. Placing the Design and the Function code into the same Visual Studio Form can also be cumbersome due to the extensive size and thereby making it more difficult to focus on the code to be developed/maintained. So we have divided the project code into the Form1.Designer.vb and Form1.vb pages. The Designer.vb code must be put in place prior to the functional code. Alternatively, the GUI of Visual Studio can be used by dragging and dropping the desired tool from the 'tool box' and then altering the property parameters at the right hand side of the Visual Studio according to the parameters in the Designer code (below). As a note, we use the Microsoft Visual Studio Professional 2017. However, the Microsoft website does offer a free version.

Figure A8.2: Initiating a new Visual Studio Project/Solution



Topic A8.2.1: Start a New Project

First, a new project/solution needs to be implemented in the Microsoft Visual Studio (MVS). Open the MVS and select the menu sequence

File > New > Project

With reference to Figure A8.2, select from the left menu in the New Project dialogue:

Visual Basic > Windows Desktop

Single click on the right hand list item:

Windows Forms App (.Net Framework)

At the bottom, select

.NET Framework 4.6.1

or a subsequent version. Place a checkmark in the box for making a directory, select a reasonable Location for the Solution/Project, and finally enter a name such as

FA2Convert

Click the OK button. Probably now is a good time to save the project by either clicking the double-diskette in the menu bar at the top or using the menu sequence File > Save All. MVS should show 'Form1.vb'. If MVS suggests resizing the form to 100%, then agree to it so the form will appear as it would on the PC display.

Topic A8.2.2: Access the Page for Form1.designer.vb

Normally a person first designs the software Graphical User Interface (sGUI) using the MVS GUI (mGUI) by dragging and dropping 'Tools' from the 'Tool Box' on the left hand side of the mGUI. However alternatively sometimes the developer will simply type the code required for creating the various sGUI controls and thereby circumvent the drag-and-drop procedure while only requiring a single page of code (instead of the designer and functional code on separate pages). However the extra code for the controls becomes mixed with the functional code needed for the control events/animations as well as the other functional methods; this mixing can make it more difficult to focus on those portions of the code that need to be developed or maintained. In the present situation, we place much of the sGUI information into the Form1.Designer.vb file where it would automatically be placed during the drag-drop procedure. It should be pointed out that all of the sGUI can be reconstructed here by dragging-and-dropping the controls listed for the form1.vb.designer and then setting the properties as also given by the designer content.

To start, make sure the Toolbox appears to the left of Form1 (left side), and the Solution Explorer on the right side and the Properties pane on the right side. If they are not visible, click the following sequences from the mGUI main menu:

View > Toolbox and View > Solution Explorer

Next, the Form1.Designer.vb code (listed below) needs to be added to the Form1.Designer.vb page. There are a couple of methods to access the Designer.vb page.

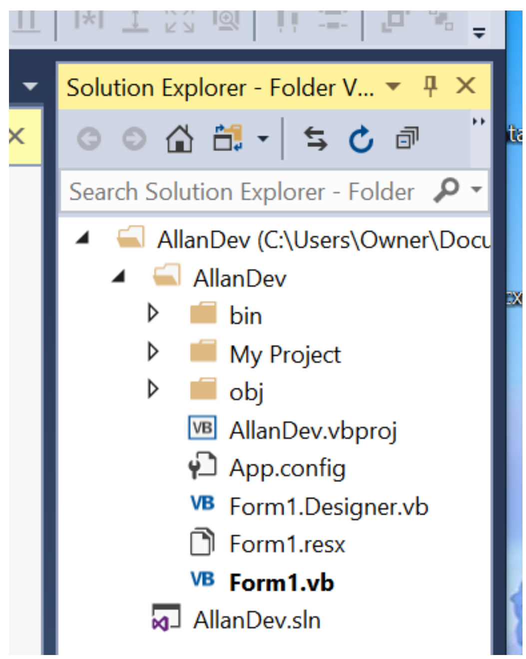


Figure A8.3: Fourth icon from the left should provide access to the Form1.Designer.vb page.

The first method of accessing the designer page uses the toolbar appearing at the top of **Solution Explorer** in the MVS (Figure A8.3). Click on the fourth icon which looks like a folder with a couple of arrows. It might be necessary to click the arrows until the files similar to those shown in Figure A8.3 appear – the files should include Form1.Designer.vb. Double Click the **Form1.Designer.vb** entry in the list in the Solution Explorer.

For the second method, drag a textbox from the Toolbox:

Toolbox > All Windows Forms > TextBox

and drop it on Form1. Double click the textbox once it has been placed on Form1. In the Form1 coding window, find 'TextBox1.TextChanged '. Right click 'TextBox1' and select 'Go to definition'. The Form1.Designer.vb page will appear. Go back to the functional coding window and delete any code related to the subroutine including 'sub' and 'end sub'.

Now select all of the content in the Form1.Designer.vb page and delete it. Copy the code listed in Topic A8.2.3 into the Form1.Designer.vb. Once completed, the GUI for Form1 should appear similar to that shown in Figure A8.1. Next: Save All. ***Please note that some lines wrap to the next line in the listing below. None of these 'wrapped around' lines have a carriage return in the middle of the line. For example, the line***

```
Me.GroupBox1.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
```

should really be entered as a single line of code:

```
Me.GroupBox1.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
```

Each complete line of code has a typical Carriage Return CR at the end as normally caused by the 'Enter' key on a typical keyboard. If you are simply cutting and pasting the code from a Word document (etc) then you don't need to worry about the wrapped lines.

Before leaving the issue of the designer code, a few comments should be made on the nature of the code.

1. Visual Studio places the lines of code in the designer because of the "Private Sub InitializeComponent()" and normally, using the GUI of Visual Studio, the programmer doesn't need to write any of them. Most of these lines can be easily understood based on the English language.
2. "MyBase" allows the code to call a member (such as a function) in the base class (sometimes called 'parent class') in order to perform a function for the derived class (sometimes called child class). Form1 is derived from the Form base class. Many methods for Form1 can be accessed through the base class using 'mybase'.
3. Notice the various components/controls are instantiated such as
   ```
   Me.Label1 = New System.Windows.Forms.Label()
   ```
   The 'Me' refers to Form1, Label1 is the name of the label given by the programmer in the Visual Studio GUI. Further down in the listing, the various properties of the label are defined such as
   ```
   Me.Label1.ForeColor = System.Drawing.Color.Blue
   Me.Label1.Location = New System.Drawing.Point(10, 299)
   ```
   which defines the text color and the location of the upper left hand corner, respectively.
4. As previously mentioned, the Visual Studio enters all of the parameter information such as that in #3 above into the Properties window on the lower right side of the Visual Studio window. The programmer does the same when using the drag and drop method of constructing the GUI. As a note, the same lines of code can instead be entered into the Functional Code of Form1 if desired.
5. Notice the statements similar to
   ```
   Me.GroupBox3.Controls.Add(Me.Label2)
   ```
   Here, 'Groupbox3' is the name of a group box which is used to simply divide up an area on the Form1 but in this case, it also contains a variety of tools/controls, one of which is an informational label named 'Label2'. The important point is that controls such as buttons are added to a collection of controls for the group box (etc.)
6. Sometimes components/controls have the 'WithEvents' keyword such as

```
Friend WithEvents BtnSave As Button
```

The 'WithEvents' keyword indicates the button will produce an event when the button is clicked. The block of code known as the event handler on the functional code page (Form1.vb) will do something in response.

After having copied the listing in Topic A8.2.3 into the Form1.Designer.vb page, proceed to Appendix Section A8.3 for the Form1 functional code. Again, for the designer, note some code wraps around to the next line – do not place a carriage return where the wrap occurs – the code should be all one coding line in the program.

Topic A8.2.3: Listing for Form1.Designer.vb

Delete all code in the Form1.Designer.vb and replace it with the following listing. Once completed, be sure to SAVE ALL: either click the two diskettes on the MVS menu or else use the menu sequence: File > Save All. Once copied, the GUI should look similar to Figure A8.1.

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()>
Partial Class Form1
  Inherits System.Windows.Forms.Form

  'Form overrides dispose to clean up the component list.
  <System.Diagnostics.DebuggerNonUserCode()>
  Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    Try
      If disposing AndAlso components IsNot Nothing Then
        components.Dispose()
      End If
    Finally
      MyBase.Dispose(disposing)
    End Try
  End Sub

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.IContainer

  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  <System.Diagnostics.DebuggerStepThrough()>
  Private Sub InitializeComponent()
    Me.GroupBox1 = New System.Windows.Forms.GroupBox()
    Me.RbFT1File = New System.Windows.Forms.RadioButton()
    Me.RbF1File = New System.Windows.Forms.RadioButton()
    Me.RbTF1File = New System.Windows.Forms.RadioButton()
    Me.RbTF2Files = New System.Windows.Forms.RadioButton()
    Me.BtnSel = New System.Windows.Forms.Button()
    Me.tbxFileName = New System.Windows.Forms.TextBox()
    Me.BtnRun = New System.Windows.Forms.Button()
    Me.GroupBox2 = New System.Windows.Forms.GroupBox()
    Me.RbFreqRaw = New System.Windows.Forms.RadioButton()
    Me.RbFreqError = New System.Windows.Forms.RadioButton()
    Me.RbFreqFraction = New System.Windows.Forms.RadioButton()
    Me.tbxContent = New System.Windows.Forms.TextBox()
    Me.Label1 = New System.Windows.Forms.Label()
```

```
        Me.TbxNominal = New System.Windows.Forms.TextBox()
        Me.TbxGTime = New System.Windows.Forms.TextBox()
        Me.BtnSave = New System.Windows.Forms.Button()
        Me.Label3 = New System.Windows.Forms.Label()
        Me.GroupBox3 = New System.Windows.Forms.GroupBox()
        Me.tbxActual = New System.Windows.Forms.TextBox()
        Me.Label2 = New System.Windows.Forms.Label()
        Me.tbxAveFreq = New System.Windows.Forms.TextBox()
        Me.Label4 = New System.Windows.Forms.Label()
        Me.GroupBox1.SuspendLayout()
        Me.GroupBox2.SuspendLayout()
        Me.GroupBox3.SuspendLayout()
        Me.SuspendLayout()
        '
        'GroupBox1
        '
        Me.GroupBox1.Controls.Add(Me.RbFT1File)
        Me.GroupBox1.Controls.Add(Me.RbF1File)
        Me.GroupBox1.Controls.Add(Me.RbTF1File)
        Me.GroupBox1.Controls.Add(Me.RbTF2Files)
        Me.GroupBox1.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.GroupBox1.ForeColor = System.Drawing.Color.Blue
        Me.GroupBox1.Location = New System.Drawing.Point(13, 13)
        Me.GroupBox1.Margin = New System.Windows.Forms.Padding(4)
        Me.GroupBox1.Name = "GroupBox1"
        Me.GroupBox1.Padding = New System.Windows.Forms.Padding(4)
        Me.GroupBox1.Size = New System.Drawing.Size(179, 142)
        Me.GroupBox1.TabIndex = 0
        Me.GroupBox1.TabStop = False
        Me.GroupBox1.Text = "File Structure"
        '
        'RbFT1File
        '
        Me.RbFT1File.AutoSize = True
        Me.RbFT1File.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte),
Integer), CType(CType(64, Byte), Integer))
        Me.RbFT1File.Location = New System.Drawing.Point(23, 54)
        Me.RbFT1File.Margin = New System.Windows.Forms.Padding(4)
        Me.RbFT1File.Name = "RbFT1File"
        Me.RbFT1File.Size = New System.Drawing.Size(139, 20)
        Me.RbFT1File.TabIndex = 14
        Me.RbFT1File.Text = "1 Files:  Freq, Time"
        Me.RbFT1File.UseVisualStyleBackColor = True
        '
        'RbF1File
        '
        Me.RbF1File.AutoSize = True
        Me.RbF1File.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte), Integer),
CType(CType(64, Byte), Integer))
        Me.RbF1File.Location = New System.Drawing.Point(23, 116)
        Me.RbF1File.Margin = New System.Windows.Forms.Padding(4)
        Me.RbF1File.Name = "RbF1File"
        Me.RbF1File.Size = New System.Drawing.Size(122, 20)
        Me.RbF1File.TabIndex = 14
        Me.RbF1File.Text = "1 File: Freq Only"
        Me.RbF1File.UseVisualStyleBackColor = True
        '
```

```
        'RbTF1File
        '
        Me.RbTF1File.AutoSize = True
        Me.RbTF1File.Checked = True
        Me.RbTF1File.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte),
Integer), CType(CType(64, Byte), Integer))
        Me.RbTF1File.Location = New System.Drawing.Point(23, 23)
        Me.RbTF1File.Margin = New System.Windows.Forms.Padding(4)
        Me.RbTF1File.Name = "RbTF1File"
        Me.RbTF1File.Size = New System.Drawing.Size(132, 20)
        Me.RbTF1File.TabIndex = 12
        Me.RbTF1File.TabStop = True
        Me.RbTF1File.Text = "1 File: Time , Freq"
        Me.RbTF1File.UseVisualStyleBackColor = True
        '
        'RbTF2Files
        '
        Me.RbTF2Files.AutoSize = True
        Me.RbTF2Files.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte),
Integer), CType(CType(64, Byte), Integer))
        Me.RbTF2Files.Location = New System.Drawing.Point(23, 85)
        Me.RbTF2Files.Margin = New System.Windows.Forms.Padding(4)
        Me.RbTF2Files.Name = "RbTF2Files"
        Me.RbTF2Files.Size = New System.Drawing.Size(136, 20)
        Me.RbTF2Files.TabIndex = 13
        Me.RbTF2Files.Text = "2 Files: Time; Freq"
        Me.RbTF2Files.UseVisualStyleBackColor = True
        '
        'BtnSel
        '
        Me.BtnSel.BackColor = System.Drawing.Color.FromArgb(CType(CType(255, Byte), Integer), CType(CType(255, Byte),
Integer), CType(CType(192, Byte), Integer))
        Me.BtnSel.FlatStyle = System.Windows.Forms.FlatStyle.Popup
        Me.BtnSel.Location = New System.Drawing.Point(219, 21)
        Me.BtnSel.Name = "BtnSel"
        Me.BtnSel.Size = New System.Drawing.Size(75, 41)
        Me.BtnSel.TabIndex = 1
        Me.BtnSel.Text = "SELECT" & Global.Microsoft.VisualBasic.ChrW(13) & Global.Microsoft.VisualBasic.ChrW(10) & "FA2 TXT"
        Me.BtnSel.UseVisualStyleBackColor = False
        '
        'tbxFileName
        '
        Me.tbxFileName.Location = New System.Drawing.Point(219, 72)
        Me.tbxFileName.MaxLength = 0
        Me.tbxFileName.Multiline = True
        Me.tbxFileName.Name = "tbxFileName"
        Me.tbxFileName.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.tbxFileName.Size = New System.Drawing.Size(286, 77)
        Me.tbxFileName.TabIndex = 2
        '
        'BtnRun
        '
        Me.BtnRun.BackColor = System.Drawing.Color.FromArgb(CType(CType(255, Byte), Integer), CType(CType(192, Byte),
Integer), CType(CType(192, Byte), Integer))
        Me.BtnRun.FlatStyle = System.Windows.Forms.FlatStyle.Popup
        Me.BtnRun.Location = New System.Drawing.Point(300, 21)
        Me.BtnRun.Name = "BtnRun"
        Me.BtnRun.Size = New System.Drawing.Size(75, 41)
```

```
        Me.BtnRun.TabIndex = 3
        Me.BtnRun.Text = "RUN"
        Me.BtnRun.UseVisualStyleBackColor = False
        '
        'GroupBox2
        '
        Me.GroupBox2.Controls.Add(Me.RbFreqRaw)
        Me.GroupBox2.Controls.Add(Me.RbFreqError)
        Me.GroupBox2.Controls.Add(Me.RbFreqFraction)
        Me.GroupBox2.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.GroupBox2.ForeColor = System.Drawing.Color.Blue
        Me.GroupBox2.Location = New System.Drawing.Point(13, 170)
        Me.GroupBox2.Margin = New System.Windows.Forms.Padding(4)
        Me.GroupBox2.Name = "GroupBox2"
        Me.GroupBox2.Padding = New System.Windows.Forms.Padding(4)
        Me.GroupBox2.Size = New System.Drawing.Size(179, 119)
        Me.GroupBox2.TabIndex = 3
        Me.GroupBox2.TabStop = False
        Me.GroupBox2.Text = "Frequency Format"
        '
        'RbFreqRaw
        '
        Me.RbFreqRaw.AutoSize = True
        Me.RbFreqRaw.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte),
Integer), CType(CType(64, Byte), Integer))
        Me.RbFreqRaw.Location = New System.Drawing.Point(23, 91)
        Me.RbFreqRaw.Margin = New System.Windows.Forms.Padding(4)
        Me.RbFreqRaw.Name = "RbFreqRaw"
        Me.RbFreqRaw.Size = New System.Drawing.Size(53, 20)
        Me.RbFreqRaw.TabIndex = 2
        Me.RbFreqRaw.Text = "Raw"
        Me.RbFreqRaw.UseVisualStyleBackColor = True
        '
        'RbFreqError
        '
        Me.RbFreqError.AutoSize = True
        Me.RbFreqError.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte),
Integer), CType(CType(64, Byte), Integer))
        Me.RbFreqError.Location = New System.Drawing.Point(23, 57)
        Me.RbFreqError.Margin = New System.Windows.Forms.Padding(4)
        Me.RbFreqError.Name = "RbFreqError"
        Me.RbFreqError.Size = New System.Drawing.Size(55, 20)
        Me.RbFreqError.TabIndex = 1
        Me.RbFreqError.Text = "Error"
        Me.RbFreqError.UseVisualStyleBackColor = True
        '
        'RbFreqFraction
        '
        Me.RbFreqFraction.AutoSize = True
        Me.RbFreqFraction.Checked = True
        Me.RbFreqFraction.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte),
Integer), CType(CType(64, Byte), Integer))
        Me.RbFreqFraction.Location = New System.Drawing.Point(23, 25)
        Me.RbFreqFraction.Margin = New System.Windows.Forms.Padding(4)
        Me.RbFreqFraction.Name = "RbFreqFraction"
        Me.RbFreqFraction.Size = New System.Drawing.Size(85, 20)
        Me.RbFreqFraction.TabIndex = 0
```

```
        Me.RbFreqFraction.TabStop = True
        Me.RbFreqFraction.Text = "Fractional"
        Me.RbFreqFraction.UseVisualStyleBackColor = True
        '
        'tbxContent
        '
        Me.tbxContent.Location = New System.Drawing.Point(219, 158)
        Me.tbxContent.MaxLength = 0
        Me.tbxContent.Multiline = True
        Me.tbxContent.Name = "tbxContent"
        Me.tbxContent.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.tbxContent.Size = New System.Drawing.Size(286, 317)
        Me.tbxContent.TabIndex = 4
        '
        'Label1
        '
        Me.Label1.AutoSize = True
        Me.Label1.ForeColor = System.Drawing.Color.Blue
        Me.Label1.Location = New System.Drawing.Point(10, 299)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(125, 16)
        Me.Label1.TabIndex = 5
        Me.Label1.Text = "Nominal Frequency"
        '
        'TbxNominal
        '
        Me.TbxNominal.Location = New System.Drawing.Point(13, 318)
        Me.TbxNominal.Name = "TbxNominal"
        Me.TbxNominal.Size = New System.Drawing.Size(157, 22)
        Me.TbxNominal.TabIndex = 6
        Me.TbxNominal.Text = "10000000.000"
        Me.TbxNominal.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'TbxGTime
        '
        Me.TbxGTime.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte),
Integer), CType(CType(64, Byte), Integer))
        Me.TbxGTime.Location = New System.Drawing.Point(7, 44)
        Me.TbxGTime.Name = "TbxGTime"
        Me.TbxGTime.Size = New System.Drawing.Size(76, 22)
        Me.TbxGTime.TabIndex = 8
        Me.TbxGTime.Text = "10"
        Me.TbxGTime.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'BtnSave
        '
        Me.BtnSave.BackColor = System.Drawing.Color.FromArgb(CType(CType(255, Byte), Integer), CType(CType(255, Byte),
Integer), CType(CType(192, Byte), Integer))
        Me.BtnSave.FlatStyle = System.Windows.Forms.FlatStyle.Popup
        Me.BtnSave.Location = New System.Drawing.Point(381, 21)
        Me.BtnSave.Name = "BtnSave"
        Me.BtnSave.Size = New System.Drawing.Size(75, 41)
        Me.BtnSave.TabIndex = 9
        Me.BtnSave.Text = "SAVE FILE"
        Me.BtnSave.UseVisualStyleBackColor = False
        '
        'Label3
        '
```

```
        Me.Label3.AutoSize = True
        Me.Label3.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte), Integer),
CType(CType(64, Byte), Integer))
        Me.Label3.Location = New System.Drawing.Point(7, 25)
        Me.Label3.Name = "Label3"
        Me.Label3.Size = New System.Drawing.Size(60, 16)
        Me.Label3.TabIndex = 10
        Me.Label3.Text = "Gate Set"
        '
        'GroupBox3
        '
        Me.GroupBox3.Controls.Add(Me.tbxActual)
        Me.GroupBox3.Controls.Add(Me.Label2)
        Me.GroupBox3.Controls.Add(Me.TbxGTime)
        Me.GroupBox3.Controls.Add(Me.Label3)
        Me.GroupBox3.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.GroupBox3.ForeColor = System.Drawing.Color.Blue
        Me.GroupBox3.Location = New System.Drawing.Point(13, 399)
        Me.GroupBox3.Margin = New System.Windows.Forms.Padding(4)
        Me.GroupBox3.Name = "GroupBox3"
        Me.GroupBox3.Padding = New System.Windows.Forms.Padding(4)
        Me.GroupBox3.Size = New System.Drawing.Size(179, 75)
        Me.GroupBox3.TabIndex = 11
        Me.GroupBox3.TabStop = False
        Me.GroupBox3.Text = "Gate Time, Cycle Time"
        '
        'tbxActual
        '
        Me.tbxActual.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte),
Integer), CType(CType(64, Byte), Integer))
        Me.tbxActual.Location = New System.Drawing.Point(96, 44)
        Me.tbxActual.Name = "tbxActual"
        Me.tbxActual.ReadOnly = True
        Me.tbxActual.Size = New System.Drawing.Size(76, 22)
        Me.tbxActual.TabIndex = 11
        Me.tbxActual.Text = "10"
        Me.tbxActual.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Label2
        '
        Me.Label2.AutoSize = True
        Me.Label2.ForeColor = System.Drawing.Color.FromArgb(CType(CType(64, Byte), Integer), CType(CType(0, Byte), Integer),
CType(CType(64, Byte), Integer))
        Me.Label2.Location = New System.Drawing.Point(96, 25)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(79, 16)
        Me.Label2.TabIndex = 12
        Me.Label2.Text = "Cycle Time:"
        '
        'tbxAveFreq
        '
        Me.tbxAveFreq.Location = New System.Drawing.Point(12, 365)
        Me.tbxAveFreq.Name = "tbxAveFreq"
        Me.tbxAveFreq.Size = New System.Drawing.Size(157, 22)
        Me.tbxAveFreq.TabIndex = 13
        Me.tbxAveFreq.Text = "0"
        Me.tbxAveFreq.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
```

```
        '
        'Label4
        '
        Me.Label4.AutoSize = True
        Me.Label4.ForeColor = System.Drawing.Color.Blue
        Me.Label4.Location = New System.Drawing.Point(12, 346)
        Me.Label4.Name = "Label4"
        Me.Label4.Size = New System.Drawing.Size(127, 16)
        Me.Label4.TabIndex = 12
        Me.Label4.Text = "Average Frequency"
        '
        'Form1
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(8.0!, 16.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(525, 487)
        Me.Controls.Add(Me.tbxAveFreq)
        Me.Controls.Add(Me.Label4)
        Me.Controls.Add(Me.GroupBox3)
        Me.Controls.Add(Me.BtnSave)
        Me.Controls.Add(Me.TbxNominal)
        Me.Controls.Add(Me.Label1)
        Me.Controls.Add(Me.tbxContent)
        Me.Controls.Add(Me.GroupBox2)
        Me.Controls.Add(Me.BtnRun)
        Me.Controls.Add(Me.tbxFileName)
        Me.Controls.Add(Me.BtnSel)
        Me.Controls.Add(Me.GroupBox1)
        Me.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Margin = New System.Windows.Forms.Padding(3, 2, 3, 2)
        Me.Name = "Form1"
        Me.Text = "FA-2 File Converter"
        Me.GroupBox1.ResumeLayout(False)
        Me.GroupBox1.PerformLayout()
        Me.GroupBox2.ResumeLayout(False)
        Me.GroupBox2.PerformLayout()
        Me.GroupBox3.ResumeLayout(False)
        Me.GroupBox3.PerformLayout()
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub

    Friend WithEvents GroupBox1 As GroupBox
    Friend WithEvents BtnSel As Button
    Friend WithEvents tbxFileName As TextBox
    Friend WithEvents BtnRun As Button
    Friend WithEvents GroupBox2 As GroupBox
    Friend WithEvents RbFreqRaw As RadioButton
    Friend WithEvents RbFreqError As RadioButton
    Friend WithEvents RbFreqFraction As RadioButton
    Friend WithEvents tbxContent As TextBox
    Friend WithEvents Label1 As Label
    Friend WithEvents TbxNominal As TextBox
    Friend WithEvents TbxGTime As TextBox
    Friend WithEvents BtnSave As Button
    Friend WithEvents Label3 As Label
```

```
    Friend WithEvents GroupBox3 As GroupBox
    Friend WithEvents RbF1File As RadioButton
    Friend WithEvents RbTF1File As RadioButton
    Friend WithEvents RbTF2Files As RadioButton
    Friend WithEvents tbxActual As TextBox
    Friend WithEvents Label2 As Label
    Friend WithEvents RbFT1File As RadioButton
    Friend WithEvents tbxAveFreq As TextBox
    Friend WithEvents Label4 As Label
End Class
```

Section A8.3: Form1 Source Code for FA2Convert

The previous section provided the code listing for the Graphical User Interface GUI of the FA2 Conversion Software (FA2Convert). Now the various controls/components need to be made functional by placing code on the Form1.vb page. At this point, the FA2Convert GUI should be visible in the Visual Studio. As previously discussed, the GUI design could have been made by simply dragging-dropping the various tools/controls into the Visual Studio GUI design area and using the parameters listed in the previous section to fill-in the properties box for each component. Having the source code in this appendix makes it easy to modify and improve the software as the programmer sees fit for the application. Some comments on the source code can be found in Topic A8.3.3 after the source code listing.

Topic A8.3.1: Notes on Form1.vb and the EXE file

The FA2Convert GUI should be visible in Visual Studio. In the Solution Explorer window, *right* click Form1.vb under the folder named the same as your software (FA2Convert). Then select 'View Code'. The Form1 code window should be displayed. If it didn't work, double click one of the components of the GUI. In either case, delete any and all code on the page. The code listed below should be copied directly into the page. Once completed, make sure any errors have been resolved. Don't worry about the 1 or 2 'messages' that might appear; the messengers don't understand the current application and don't give relevant messages – ignore them. Then type CTRL F5, or else click the 'Start' on the tool bar just below the main menu.

As a note, once the program has successfully run, the EXE file can be added to the desktop and run by double clicking on the icon. To access the EXE file, open the directory containing the FA2Convert using Windows Explorer. The directory will contain FA2Convert.sln and also a folder labeled as FA2Convert. Open this second folder and then open the 'bin' folder and then copy the .exe file. It should be pointed out that Visual Studio can make an installer (MSI) although it involves some extra steps. The .exe file by itself does not install in the Windows registry.

Topic A8.3.2: Form1.vb Source Code

Open the function code page, namely Form1.vb, and delete any and all code that might be there, and then copy the code listed below into the Form1.vb. The comments can be omitted as desired.

```
'set carat
```

```
' https://docs.microsoft.com/en-us/dotnet/framework/wpf/controls/position-the-cursor-at-the-beginning-or-end-of-text
'string builder
' https://www.tutorialsteacher.com/csharp/csharp-stringbuilder
'convert HH:MM:SS to seconds
' https://stackoverflow.com/questions/45589909/vb-convert-mmss-timecode-to-seconds
'inherit list
'http://www.vbforums.com/showthread.php?454471-Having-problems-with-lists-(inherits-list-(of-T))

Imports System.Text

Public Class Form1

    ''' <summary>
    ''' TimeFreqClass: Associates the measured frequency with the
    ''' time of the measurement (Hertz). This class is essentially
    ''' the ordered pair (time,freq)
    ''' </summary>
    Public Class TimeFreqClass
        Private _time As Decimal
        Private _freq As Decimal

        Public Sub New()
        End Sub
        Friend Sub New(ByVal time As Decimal, ByVal freq As Decimal)
            _time = time : _freq = freq
        End Sub
        Friend Property Time As Decimal
            Set(value As Decimal)
                _time = value
            End Set
            Get
                Return _time
            End Get
        End Property
        Friend Property Freq As Decimal
            Set(value As Decimal)
                _freq = value
            End Set
            Get
                Return _freq
            End Get
        End Property

    End Class

    ''' <summary>
    ''' MyList: A list of the ordered pairs of (time,freq) for the TimeFreqClass
    ''' A hodgepodge of functions and subroutines - some should only be used
    ''' once to prevent corrupting the list data
    ''' </summary>
    Friend Class MyList
        Inherits List(Of TimeFreqClass)
        'Inherits System.Collections.ObjectModel.Collection(Of TimeFreqClass)


        ''' <summary>
        ''' removes time offset from current instance and then adds the gate time
        ''' to the current instance
```

```
''' Example: if first time entry is 23sec and gate time is 10sec
'''       then the new first entry becomes 23-23=0 then
'''       include the first 10sec gate time so new first
'''       entry is 10 sec.
''' Use gate time of 1 sec if want time sequence to be 1, 2, ...
''' </summary>
''' <param name="GateTime">Acquisition averaging time of frequency counter</param>
Friend Sub TimeIndexOffset(ByVal GateTime As Decimal)
   If Me.Count = 0 Then Throw New Exception("Error: List empty")
   Dim tim As Decimal = Me.Item(0).Time
   For i As Integer = 0 To Me.Count - 1
      Me.Item(i).Time += GateTime - tim
   Next
End Sub

''' <summary>
''' for the current instance calculates the acquisition/cycle
''' time which includes the gate and dead time of the
''' frequency counter
''' </summary>
''' <returns>average acquisition cycle time</returns>
Friend Function TimeCycle() As Decimal
   If Me.Count - 1 <= 0 Then Throw New Exception("Error: insufficient number of data points")
   Dim ave As Decimal = 0
   For i As Decimal = 1 To Me.Count - 1
      'calculate average without summing many large numbers
      ave = ((i - 1) / i) * ave + (Me.Item(i).Time - Me.Item(i - 1).Time) / i
      Application.DoEvents()
   Next i
   Return ave
End Function

''' <summary>
''' calculates the average frequency over all freqs in the class
''' </summary>
''' <returns>average frequency</returns>
Friend Function FreqAve() As Decimal
   If Me.Count - 1 <= 0 Then Throw New Exception("Error: insufficient number of data points")
   Dim ave As Decimal = 0
   For i As Decimal = 1 To Me.Count
      'calculate average without summing many large numbers
      ave = ((i - 1) / i) * ave + (Me.Item(i - 1).Freq) / i
      Application.DoEvents()
   Next i
   Return ave
End Function

''' <summary>
''' Replaces the current-instance frequency with the fractional frequency
''' Frequency calculations should only be used once.
''' </summary>
''' <param name="FreqNominal">designed nominal frequency: usually 10,000,000</param>
Friend Sub ConvToFracFreq(ByVal FreqNominal As Decimal, ByVal ListDest As MyList)
   If (FreqNominal <= 0 Or Me.Count < 1) Then
      Throw New Exception("Nominal frequency must be great than or equal to 0 and # samples > 0")
   End If

   Dim NewFreq As Decimal = 0
```

```
        ListDest.Clear()
        For i As Integer = 0 To Me.Count - 1
            NewFreq = (Me.Item(i).Freq - FreqNominal) / FreqNominal
            ListDest.Add(New TimeFreqClass(Me.Item(i).Time, NewFreq)) 'populates destination list
            Application.DoEvents()
        Next i
    End Sub

    ''' <summary>
    ''' Deep copies current instance of MyList to another named ListDest
    ''' supplied in the argument
    ''' </summary>
    ''' <param name="ListDest">a list to receive deep copied content</param>
    Friend Sub RawCopy(ByVal ListDest As MyList) '(Of TimeFreqClass))
        If Me.Count > 0 Then
            ListDest.Clear()
            For i As Integer = 0 To Me.Count - 1
                Dim z As New TimeFreqClass(Me.Item(i).Time, Me.Item(i).Freq)
                ListDest.Add(z) 'populates destination list
            Next i
        End If
    End Sub

    ''' <summary>
    ''' Replaces the current-instance frequency with the error frequency
    ''' Frequency calculations should only be used once.
    ''' </summary>
    ''' <param name="FreqNominal">designed nominal frequency: usually 10,000,000</param>
    Friend Sub ConvToErrorFreq(ByVal FreqNominal As Decimal, ByVal ListDest As MyList)
        If (FreqNominal <= 0 Or Me.Count < 1) Then
            Throw New Exception("Nominal frequency must be great than or equal to 0 and # samples > 0")
        End If

        Dim NewFreq As Decimal = 0
        ListDest.Clear()
        For i As Integer = 0 To Me.Count - 1
            NewFreq = Me.Item(i).Freq - FreqNominal
            ListDest.Add(New TimeFreqClass(Me.Item(i).Time, NewFreq)) 'populates destination list
            Application.DoEvents()
        Next i
    End Sub

End Class

Friend listTimeFreqOrig As New MyList   'original Time,Freq values - keep as backup
Friend listTimeFreqMod As New MyList    'working modifiable list of Time,Freq values
' array of conversions to seconds from hh:mm:ss
Friend TimConvert = {1, 60, 60 * 60, 24 * 60 * 60}  '1sec/sec  60sec/min  60min/hr  24 * Hrs/day

Friend WithEvents OFD As OpenFileDialog
Friend WithEvents SFD As SaveFileDialog

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    OFD = New OpenFileDialog()
    SFD = New SaveFileDialog()
    SFD.DefaultExt = ".txt"
End Sub
Private Sub Form1_FormClosing(sender As Object, e As FormClosingEventArgs) Handles MyBase.FormClosing
```

```
    If OFD IsNot Nothing Then OFD.Dispose()
    If SFD IsNot Nothing Then SFD.Dispose()
End Sub

''' <summary>
''' Select and open and process a file from FA-2 through Termite comm. software
''' Write file content to textbox as well as filename, cycle time, ave freq
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
Private Sub BtnSel_Click(sender As Object, e As EventArgs) Handles BtnSel.Click
    Try
        If OFD.ShowDialog() = DialogResult.OK Then
            'clear original list if it has previously been populated with file data
            If listTimeFreqOrig IsNot Nothing Then
                listTimeFreqOrig.Clear()
            Else
                listTimeFreqOrig = New MyList
            End If

            'use stringbuilder instead of string to make it faster to print to textbox
            Dim sbTimFrq As New StringBuilder(1200)        'cummulative text read in
            Dim strReadIn As String                        'line of text from file

            Dim FilePathAndName As String = OFD.FileName    'assign selected filename
            tbxFileName.Text = FilePathAndName             'print filenamee to textbox

            Dim file As System.IO.StreamReader             'check for dispose
            file = My.Computer.FileSystem.OpenTextFileReader(FilePathAndName)
            tbxContent.Text = ""

            While Not file.EndOfStream
                strReadIn = file.ReadLine()                'read a line of text from selected file
                sbTimFrq.Append(strReadIn + vbCrLf)        'cumulate for quick write to Textbox later
                ' Next convert to time(secs),Freq;
                ' Then save in ordered pair of TimeFreqClass
                ' Then add to MyList named listTimeFreqOrig <- not to be modified
                listTimeFreqOrig.Add(ProcessTimeFreq(strReadIn))
                Application.DoEvents()
            End While

            tbxAveFreq.Text = listTimeFreqOrig.FreqAve.ToString 'print ave freq to textbox
            tbxActual.Text = listTimeFreqOrig.TimeCycle         'print cycle time to textbox
            tbxContent.Text = sbTimFrq.ToString                 'print read-in text to textbox
            sbTimFrq.Clear()
            Application.DoEvents()

            file.Close()
        End If
    Catch ex As Exception
        MessageBox.Show(ex.ToString, ErrorToString, MessageBoxButtons.OK)
    End Try
End Sub

''' <summary>
''' extracts frequency and time from input string strSrcTimFrq
''' Calculates time in seconds
''' places Time,Freq in ordered pair for TimeFreqClass
```

```
''' </summary>
''' <param name="strSrcTimFrq">input string from file</param>
''' <returns>ordered pair Time,Freq in form of TimeFreqClass</returns>
Private Function ProcessTimeFreq(ByRef strSrcTimFrq As String) As TimeFreqClass

    Dim timFrqResult As New TimeFreqClass                   'holds Time,Freq results
    Dim indx As Integer = strSrcTimFrq.IndexOf("F:")        'find start of freq

    timFrqResult.Freq = CDec(strSrcTimFrq.Substring(indx + 2))  'save freq

    indx = strSrcTimFrq.IndexOf(": ")                       'find break between Time and Freq
    strSrcTimFrq = strSrcTimFrq.Substring(0, indx)          'working string for Time

    timFrqResult.Time = 0
    Dim strTime = strSrcTimFrq.Split(":")                   'isolate hr min sec into array
    indx = UBound(strTime)
    For i As Integer = indx To 0 Step -1                    'calculate seconds
        timFrqResult.Time += CDec(strTime(i)) * TimConvert(indx - i)
    Next i

    Return timFrqResult
End Function

''' <summary>
''' Modifies the working list entries for fractional frequency or error frequency
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
Private Sub BtnRun_Click(sender As Object, e As EventArgs) Handles BtnRun.Click
    Try

        If listTimeFreqMod IsNot Nothing Then
            listTimeFreqMod.Clear()
        Else
            listTimeFreqMod = New MyList
        End If

        Dim NomFreq As Decimal = Decimal.Parse(TbxNominal.Text)     'read the nominal frequency

        Select Case True                                    'select freq format and apply
            Case RbFreqFraction.Checked
                listTimeFreqOrig.ConvToFracFreq(NomFreq, listTimeFreqMod)  'copy orig list to working list w fract frequency
            Case RbFreqError.Checked
                listTimeFreqOrig.ConvToErrorFreq(NomFreq, listTimeFreqMod)  'copy orig list to working list w error frequency
            Case RbFreqRaw.Checked
                listTimeFreqOrig.RawCopy(listTimeFreqMod)       'copy orig list to working list, no mods
        End Select

        Dim GateTime As Decimal = Decimal.Parse(TbxGTime.Text)      'read gate time from textbox
        listTimeFreqMod.TimeIndexOffset(GateTime)               'modify working list to (time-offset)+gatetime

        Dim sbTimFrq As New StringBuilder(1200)                 'stringbuilder for quick transfer to textbox
        sbTimFrq.Clear()                                    'make sure clear
        For i As Integer = 0 To listTimeFreqMod.Count - 1
            sbTimFrq.Append(listTimeFreqMod(i).Time.ToString + "  " + listTimeFreqMod(i).Freq.ToString + vbCrLf)
            If i Mod 100 = 0 Then Application.DoEvents()        'allow events every 100 calculations
        Next i
```

```
      tbxContent.Text = sbTimFrq.ToString

   Catch ex As Exception
      MsgBox(ex.ToString,, "Error")
   End Try

End Sub

''' <summary>
''' Save results to file
''' Will save as single file or multiple depending on UI radio buttons
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
Private Sub BtnSave_Click(sender As Object, e As EventArgs) Handles BtnSave.Click
   Try
      Dim sfdDirectoryWOslash As String = ""
      Dim sfdFilenameWOextWOpath As String = ""
      Dim sfdFullFilename As String = ""
      Dim sfdExtWdot As String = ""

      Dim StreamWriteFName As String = ""
      Dim FullNameWrite02 As String = ""

      SFD.Title = "Filename will be given -T -F -TF -FT"

      If SFD.ShowDialog() = DialogResult.OK Then

         sfdFullFilename = SFD.FileName
         sfdDirectoryWOslash = IO.Path.GetDirectoryName(sfdFullFilename)
         sfdExtWdot = IO.Path.GetExtension(sfdFullFilename)
         sfdFilenameWOextWOpath = IO.Path.GetFileNameWithoutExtension(sfdFullFilename)

         Dim FileStrWrite As System.IO.StreamWriter
         Dim utf8WoBom As New System.Text.UTF8Encoding(False) 'eliminates 3 extra bytes in file: Byte order mark (BOM)

         Select Case True
            Case RbTF1File.Checked  'writes CSV of Time, Freq in one file
               StreamWriteFName = sfdDirectoryWOslash + "\" + sfdFilenameWOextWOpath + "-TF" + sfdExtWdot
               'BOM: Byte Order Marker: do not include these three bytes in file:
               FileStrWrite = My.Computer.FileSystem.OpenTextFileWriter(StreamWriteFName, False, utf8WoBom)
               For i As Integer = 0 To listTimeFreqMod.Count - 1   'write all data to text file
                  FileStrWrite.WriteLine(listTimeFreqMod.Item(i).Time.ToString + "," + listTimeFreqMod.Item(i).Freq.ToString)
               Next i
               FileStrWrite.Close()
               FileStrWrite.Dispose()

            Case RbFT1File.Checked  'writes CSV of Freq, Time in one file
               StreamWriteFName = sfdDirectoryWOslash + "\" + sfdFilenameWOextWOpath + "-FT" + sfdExtWdot
               'BOM: Byte Order Marker: do not include these three bytes in file:
               FileStrWrite = My.Computer.FileSystem.OpenTextFileWriter(StreamWriteFName, False, utf8WoBom)
               For i As Integer = 0 To listTimeFreqMod.Count - 1
                  FileStrWrite.WriteLine(listTimeFreqMod.Item(i).Freq.ToString + "," + listTimeFreqMod.Item(i).Time.ToString)
               Next i
               FileStrWrite.Close()
               FileStrWrite.Dispose()

            Case RbTF2Files.Checked 'writes two separe files: one for time and another for freqs
```

```
            StreamWriteFName = sfdDirectoryWOslash + "\" + sfdFilenameWOextWOpath + "-T" + sfdExtWdot
            'BOM:Byte Order Marker: do not include these three bytes in file:
            FileStrWrite = My.Computer.FileSystem.OpenTextFileWriter(StreamWriteFName, False, utf8WoBom)
            For i As Integer = 0 To listTimeFreqMod.Count - 1
              FileStrWrite.WriteLine(listTimeFreqMod.Item(i).Time.ToString)
            Next i
            FileStrWrite.Close()

            StreamWriteFName = sfdDirectoryWOslash + "\" + sfdFilenameWOextWOpath + "-F" + sfdExtWdot
            'BOM:Byte Order Marker: do not include these three bytes in file:
            FileStrWrite = My.Computer.FileSystem.OpenTextFileWriter(StreamWriteFName, False, utf8WoBom)
            For i As Integer = 0 To listTimeFreqMod.Count - 1
              FileStrWrite.WriteLine(listTimeFreqMod.Item(i).Freq.ToString)
            Next i
            FileStrWrite.Close()

            FileStrWrite.Dispose()

          Case RbF1File.Checked   'write a single file with only frequency

            StreamWriteFName = sfdDirectoryWOslash + "\" + sfdFilenameWOextWOpath + "-F" + sfdExtWdot
            'BOM:Byte Order Marker: do not include these three bytes in file:
            FileStrWrite = My.Computer.FileSystem.OpenTextFileWriter(StreamWriteFName, False, utf8WoBom)
            For i As Integer = 0 To listTimeFreqMod.Count - 1
              FileStrWrite.WriteLine(listTimeFreqMod.Item(i).Freq.ToString)
            Next i
            FileStrWrite.Close()
            FileStrWrite.Dispose()

        End Select

      End If

    Catch ex As Exception
      MsgBox("Save File Error",, "Error")
      If SFD IsNot Nothing Then SFD.Dispose()

    End Try

    If SFD IsNot Nothing Then SFD.Dispose()
    GC.Collect()

  End Sub

End Class
```

Topic A8.3.3: Comments on the FORM1 Code

There are several aspects to the programming: (i) Design of the GUI as in Section A8.2. (ii) VB.net is fully object oriented (as are other .net languages) and so the program must define and instantiate the objects of interest. (iii) Given that VB.net is event driven, most of the computation will take place in response to events. (iv) The program can be prepped for an installer MSI or for use as a stand-alone EXE file. The EXE file doesn't require any additional effort. The FA2Convert GUI was discussed in Section A8.3.1. The code behind the GUI can be found on Form1 as described in Topic A8.3.3.

Form1 has two classes.

1. The class **'TimeFreqClass'** defines a (heap-based) object resembling an ordered pair of Decimal numbers for Time in seconds and Frequency in Hz. The associated properties simply assign numbers to each ordered pair.

The **second class, 'MyList',** inherits from the Microsoft Visual Studio (MVS) 'List' configured to only accept the TimeFreqClass objects. So objects instantiated from MyList serve to store TimeFreqClass ordered pairs; these pairs are the data from the FA-2. The class uses the parent NEW subroutine to instantiate new MyList objects.

2. TimeIndexOffset finds the first sample time in the file and subtracts it as on offset from each myList entry and then adds the FA2 Gate time. For example, if the first time entry in the saved file is 23 seconds and the gate time is 10sec then the new first entry becomes 23-23=0 but then including the 10 second gate time yields a new first entry of 10seconds.
3. TimeCycle calculates the actual gate time plus dead time. For the FA2, the Cycle Time is about 0.3 seconds longer than the 10 second time. That is, the dead time is about 0.3 seconds for the 9600 baud rate. The TimeCycle routine simply takes an average of all the differences in the sample times. One way to do this would be
$$Ave\ Diff = \frac{1}{N-1}\sum_{i=2}^{N}(Time_i - Time_{i-1})$$
Prior to division by N-1, the summation can become quite large for a large number N of samples. An alternative is to calculate an average for each entry and modify the previous using
```
ave = ((i - 1) / i) * ave + (Me.Item(i).Time - Me.Item(i - 1).Time) / i
```
where item(i) refers to the ith entry of the list and Ave is the time difference average.
4. FreqAve finds the average frequency so it can be compared with the nominal. Notice that the average is computed in a manner similar to #3 above.
5. RawCopy copies the presently used (original) MyList to another denoted by ListDest. That way the original MyList can be reused without needing to reread the file.
6. ConvToFracFreq converts the stored frequency to a fractional frequency and places it in the passed ListDest. The Form1 code will receive the ListDest and use it as the modified MyList.
7. ConvToErrorFreq converts the stored frequency to the error frequency and places it in the passed ListDest. The Form1 code will receive the ListDest and use it as the modified MyList.

After defining the two classes, the Form1 code declares some global variables and then continues with the subroutines and functions.

8. The variable listTimeFreqOrig is a 'MyList' and holds an original copy of the file data set … not to be modified. The listTimeFreqMod is a copy of the original MyList and the stored values will generally be modified from the original. The array TimConvert holds conversion for the time: 60 converts minutes to seconds, 60*60 converts hours to seconds, and 24*60*60 converts days to seconds. The user should exercise caution in that the software does not have routines to convert days to seconds nor have we run the FA-2 long enough to determine if it uses the days as opposed to hours beyond 24.

9. The OFD and SFD refer to the OpenFileDialog and SaveFileDialog that appears when Selecting and Saving a file, respectively. Normally these statements could be omitted by dragging-dropping the relevant tool from the ToolBox onto the FA2Convert GUI. Notice that the OFD and SFD have not yet been instantiated (no memory reserved) but MVS does now know the meaning of OFD and SFD and knows events are associated with them.
10. The Form1_load event handler instantiates the OFD and SFD
11. The Form1_FormClosing event handler executes when exiting the FA2Convert. It makes sure that the system resources associated with the OFD and SFD have been released back to the system.
12. BtnSel_Click handles the event that occurs when the Select button is clicked. BtnSel_Click will open a file from the FA2 and the content read into a MyList object for processing. First either the existing listTimeFreqOrig object is cleared of all data or, if nonexistent, a new one is instantiated. Next, because the routine will be printing the content of the file to a GUI textbox and because the GUI textbox prints strings and because appending strings takes enormous amounts of CPU time, we define a string builder variable sbTimFrq to continuously receive text from the file. The String Builders don't need to make a new string variable for appending characters and strings. The OFD returns the filename to FilePathandName. A StreamReader is defined as 'file'. The file is read one line (defined by the CR in the file) at a time into a string variable strReadIn and then appended to the string builder sbTimeFrq. A function ProcessTimeFreq is called to separate the Time and Frequency content of strReadIn into an ordered pair (TimeFreqClass) for the statement
```
listTimeFreqOrig.Add(ProcessTimeFreq(strReadIn))
```
The subroutine 'Add' as a method of listTimeFreqOrig places the ordered pair into the list. Once encountering the end of the file, the routine prints the file text from sbTimFrq, which is the compilation of all the data in the file, to the lower right textbox on the GUI. Additionally the frequency average (.FreqAve) and the Cycle Time (.TimeCycle) is printed to their respective textboxes on the GUI. The string builder sbTimFrq is cleared in preparation for the next file.
13. ProcessTimeFreq reads in a string strSrcTimFrq consisting of a line of text from the FA2 output file and after processing, returns a TimeFreqClass ordered pair. The line of text from the file has the form discussed in Section A8.1

    strSrcTimFrq = hh:mm:ss: * F:0010000023.123456789

    First the subroutine searches for the 'F:' and converts the text after that to a decimal number and then stores it as the frequency in a MyList object as
```
timFrqResult.Freq = CDec(strSrcTimFrq.Substring(indx + 2))
```
Next the colon after the time is found by searching for the colon with a space character ': ' and then it places the time string portion (hh:mm:ss) in strSrcTimFrq. Next the 'Split' function divides the content of strSrcTimFrq into an array strTime as hours, minutes, seconds. Next the TimConvert array is multiplied into the strTime array to convert the elements to seconds. The result is stored in the time component timFrqResult.Time. The function then returns the timFrqResult ordered pair. This ordered pair is considered to be the original frequency and time data from the FA2 file. This function will be called once for each line of the FA2 file so as to build the complete MyList object for the time and frequency.
14. BtnRun_Click handles events generated by clicking the Run button on the GUI. The BtnRun even handler is actually does most of the processing in converting the original Time Frequency data into the modified version suitable for plotting and Allan Deviation routines. A Select case structure determines which of the GUI radio buttons in the Frequency Format group box is selected so that the corresponding MyList subroutine is executed (ConvToFractFreq, or ConvToErrorFreq, or RawCop) to copy modified data into the listTimeFreqMod instantiation of

MyList. Next the subroutine '.TimeIndexOffset' starts the listTimeFreqMod time at one gate time. For example if the original sample times were 2:21, 2:31, … then the modified times would be 10, 20, … . Next a string builder variable is defined, namely sbTimFrq, and all of the Time and Frequency values from the modified MyList are appended to the string builder. The sbTimFrq content is then transferred the lower right textbox on the GUI.

15. BtnSave_Click handles event generated by clicking the Save File button on the GUI. The event saves the modified MyList content, namely listTimeFreqMod, to file in a format selected by one of the radio buttons in the 'File Structure' group box.

Section A8.4 References

[A8.1] The source code, listings, downloads, flash drives, preprogrammed microcontroller:
(A) Chapter 10 and Appendix 6 provide the source code for the C/C++ program
(B) Appendices 7-9 provide the PC software
(C) Check EBay.com in connection with the title of this book: software and preprogrammed MEGA328P
(D) check www.AngstromLogic.com for current offerings, or write to Sales@AngstromLogic.com
Keep in mind that Angstrom Logic, LLC is not setup/able to answer questions on the book or software.
(E) Check Amazon in connection with the title of this book.
(F) Note: The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the words) and the contents are copyrighted with all rights reserved. The software/firmware is scanned for viruses before posting. Download only from secure sources and rescan for viruses before using.
(G) The Original Works are provided under this License on an "AS IS" BASIS and WITHOUT WARRANTY, either express or implied, including, without limitation, the warranties of non-infringement, merchantability or fitness for a particular purpose. THE ENTIRE RISK AS TO THE QUALITY OF THE ORIGINAL WORK IS WITH YOU. This DISCLAIMER OF WARRANTY constitutes an essential part of this License. No license to the Original Works is granted by this License except under this disclaimer. Limitation of Liability: Under no circumstances and under no legal theory, whether in tort (including negligence), contract, or otherwise, shall the Licensor be liable to anyone for any indirect, special, incidental, or consequential damages of any character arising as a result of this License or the use of the Original Work including, without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses.

Appendix 9: The Windows FE5650A Interface

Software for the PC provides an easy-to-use interface for the FE-5650A/5680A. The FE-5650A PC Interface FEPCI Software allows the modified FEI Rb standard to be tested with any of the commands listed in the main chapters, assists with determining the true reference frequency, and offers functionality to set the operating frequency by Fcode or by typing the frequency. As discussed in previous appendices and chapters, demo software accompanies the book for Windows 10 and later machines. Section A9.1 provides an overview of the Graphical User Interface (GUI) and the basic program flow of the software. Sections A9.2 and A9.3 provide the relevant software listings for those readers who prefer to enter their own programs and improve on the one provided here, It should be pointed out that the software has been written in Visual Basic (using Visual Studio 2017 and later) since the code is almost self-documenting and especially since the language does not use the single character tokens (such as '{' and '}' and ';') that can be difficult to accurately copy. Online resources can translate the code into Microsoft C# if so desired. Section A9.2 lists the design code that can be copied and pasted into the visual studio designer page which translates into the various controls (such as textbox, Numeric Up Down). Section A9.3 lists the actual VB.net software coding. [A9.1]

Section A9.1: Graphical User Interface (GUI) and Flowchart

The Graphical User Interface (GUI) requires a USB-RS232 adapter to be installed for the PC to send commands and receive response data. Depending on the version, the FEPCI software can be either installed using the Microsoft MSI installer or it can be run by double clicking the .exe file. Figure A9.1 shows the GUI which can be modified using the source code (Sections A9.2 and A9.3).

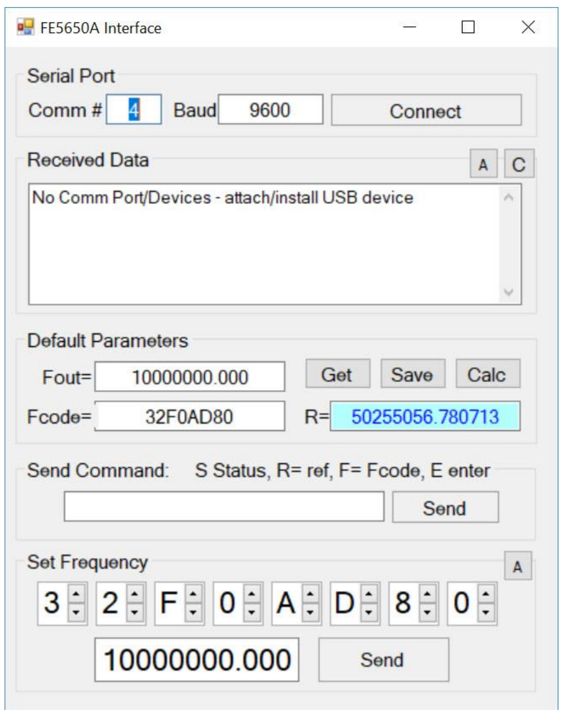


Figure A9.1: The Graphical User Interface (GUI)

Group boxes divide the form into the five sections of Serial Port, Received Data, Default Parameters, Send Command and Set Frequency.

1. The 'Serial Port' block opens communications with the FE-5650A. The Baud rate is set to 9600 although it can be changed by typing over the currently displayed number. Be aware that the numbers should be carefully entered without errors as the software have very little error-checking. If the USB-RS232 adapter was connected prior to executing the FE5650A software, either the correct port will display in the Comm# textbox or else the Received Data textbox will show several suggestions. One of the suggestions can be entered into the Comm# textbox. Click Connect to open the serial port to the FE-5650A.

2. The 'Received Data' block shows the data returned by the FE-5650A as both English Text and the HEX equivalent. Various different encodings can be displayed by changing a line of source code.

As mentioned in the main chapters, some of the FE-5650A units return what-appears-to-be encrypted text at 9600 baud. Generally, the Received Data textbox is meant to work with the Status command to show the FE-5650A stored value of the reference frequency and the value of the Fcode (in HEX). The FE-5650A uses Fcode and the Reference frequency Rstored used to set the default output frequency (often 8.388608 Hz).

a. The button labeled by ‘C’ clears the textbox
b. The toggle button labeled by ‘A’ can be clicked to automatically clear the Received Data textbox prior to displaying returned data. Actually it’s a checkbox fashioned into a button with memory for the previous state.
c. The Received Data textbox has a vertical scrollbar on the right hand side to view returned data that overflows the viewable area. It’s probably best to use the auto-clear feature to keep the execution time to a minimum.

3. The ‘Default Parameters’ Section is a bit awkward but this program uses the true reference frequency R to set the output frequency (see ‘Set Frequency’ below). The ‘Default Parameters’ block can be used to calculate the true Reference frequency R based on the values of Fout (in Hz) and Fcode (in Hex). The Default Parameters block does not retrieve/store these values at the FE-5650A but it can Get/Save them from/to a file on the PC. As a note, recall that the FE5650A value of R, termed Rstored, can be retrieved using the Status S command in #4 below.
   a. The R textbox shows the value of R used for calculations. It can be manually changed by double clicking the R textbox, which removes the read-only lockout, and then typing the value and then double clicking the textbox again to relock it so the value won’t be accidently changed. The R textbox can also be changed by calculation or by retrieving a value from file.
   b. The Calc button uses the value of Fout and Fcode found in their respective textboxes to calculate a new value for R. New values of Fout and Fcode can be entered by hand; the actual values normally obtain from a calibration procedure or from the response of the FE5650A to the Status S command.
   c. The Get button retrieves the last saved parameters, such as R, from a file. The use of files makes it possible to save and retrieve the parameters for several different FE5650A units. Clicking the Get button brings an Open File Dialog to select a text file with the parameters if available.
   d. The Save button opens a Save File Dialog that will place the Fout, Fcode and R values into a file.

4. The ‘Send Command’ section sends a command typed into the corresponding text box. The carriage return <cr> is automatically added to the command string. The commands can be one of the following: Status S, Reference frequency such as R=50255056.7800713, the Fcode such as F=32F0AD80, and the Enter E which stores the Reference and Fcode into the FE5650A. It’s probably best to stay away from the Enter E command since the factory set value will be overwritten. By the way, the sent Fcode can be up to 64 bytes such as F=32F0AD8001234567 but the last digits do not appear to affect the output frequency.

5. The ‘Set Frequency’ section allows the user to either type a frequency in Hz or set the Fcode in the Numeric Up-Down controls (a.k.a., spinners) which is combined with the R value in the ‘Default Parameters’ in order to set the FE-5650A output frequency.

a. The spinners can be set by clicking the up and down arrows next to each Hex digit. The frequency (Hz) textbox will update as the spinner values change. The updated value will not be sent to the FE-5650A unless the A toggle button has been set.

b. The A button retains its memory of the last click to determine whether or not the software will send the updated output frequency to the FE5650A when the spinner values change. The button is actually a reformed checkbox. Clicking the A button does not cause the software to send the updated frequency from the Hz textbox

c. The Frequency Hz textbox displays the frequency to be sent to the FE5650 using the Fcode shown on the spinners. Changing the textbox frequency will update the spinner display. The actual output frequency will be in error up to 1 Hex digit displayed by the spinners (which translates to roughly 0.012 Hz at 10MHz). In all cases, the frequency Fout is calculated using the equations associated with the AD9830A Direct Digital Synthesizer on the DDS board in the FE5650A.

$$F_{out} = R\frac{F_{code}}{2^{32}} \quad \text{(A9.1)}$$

d. The Send button formats the frequency in the textbox and sends it to the FE5650.

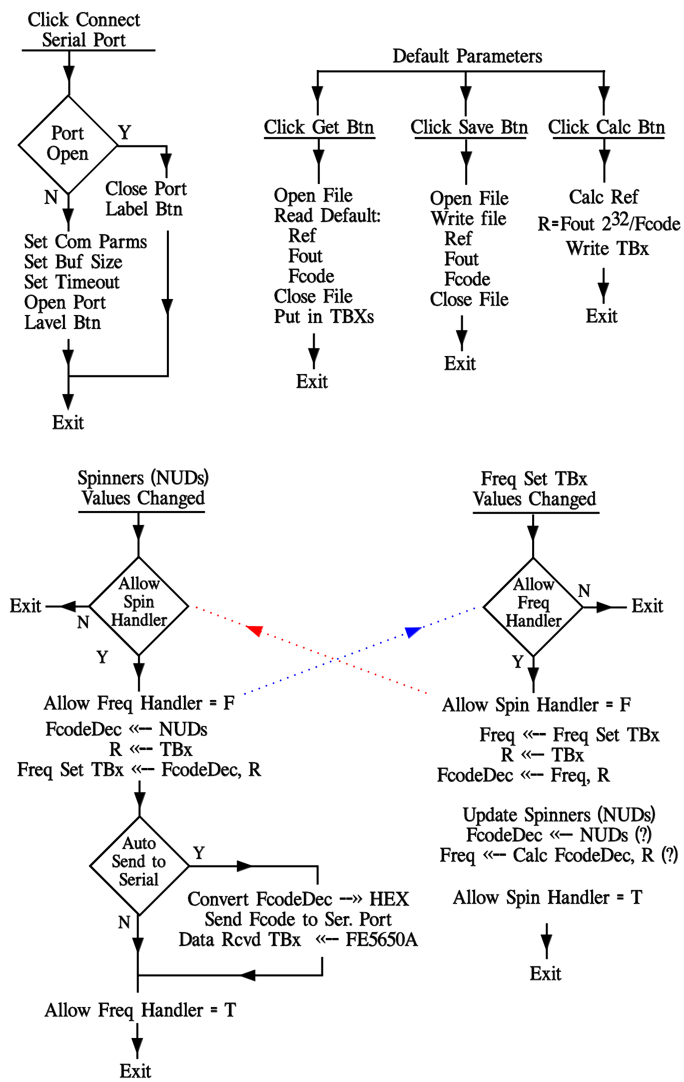


Figure A9.2: Basic flow charts

Prior to discussing the code in the next couple of sections, consider the basic flow chart shown in Figure A9.2. The top two sub-diagrams show the Serial Port and Default Parameters. The bottom portion of the figure shows the interrelation between the event handlers for changing the spinners and the frequency textbox. A type of run-away situation can develop. Changing the spinners causes the textbox to be written with a new frequency. Then that change in textbox text causes the textbox event handler to fire and change the spinners. If there's any change to the spinners then event handlers would cause the textbox to change. To prevent this pseudo runaway behavior, Boolean variables 'Allow Freq Handler' and 'Allow Spin Handler' are defined. When the Spinner handler executes, the 'Allow Freq Handler' will be set to False, and when the Frequency Textbox Change handler attempts to run, it will immediately exit. Similar situation occurs for the Frequency Change Textbox handler and the "Allow Spin Handler" Boolean. An ordinarily better solution would be to remove the event handler designators from the routine but in the case of the spinners, all eight spinners are handled by the same event handler.

Section A9.2: Form1.designer.vb for the FE5650A Interface

The present section contains the design code for the FE5650A Interface software. As previously mentioned, the software is written in Visual Basic .NET (VB.NET) rather than C# since the VB does not have the single character tokens such as '{' and '}' and ';'. These tokens can be easily missed while copying. The VB.NET can be easily translated to C# if desired by using free online translators. The program listed here uses the older .net Forms rather than the Universal or Windows Presentation Foundation. As with all Microsoft Visual Studio projects/solutions, the programmer has the option of dragging/dropping controls/tools onto a given Form so as to construct the interface, or develop both the visual and functional aspects entirely within the code behind the displayed Form. The drag-and-drop method is cumbersome to list in a book but is probably the better method for learning. Placing the Design and the Form Functional code into the same Visual Studio Form can also be cumbersome due to the extensive size. So we have divided the project/solution into listings for the Form1.Designer.vb and Form1.vb pages. The Designer.vb code must be placed prior to the Form1 functional code. Alternatively, the graphical interface of Visual Studio can be used by dragging and dropping the desired tool from the 'tool box' and then altering property parameters at the right hand side of the visual Studio according to the parameters listed below. As a note, we use the Microsoft Visual Studio Professional 2017. However, the Microsoft website does offer a free version.

Topic A9.2.1: Access Form1.Designer.vb

From within the Microsoft Visual Studio (MVS), select the menu sequence

File > New > Project

Then select

Visual Basic

from the left menu and single click 'Windows Forms App (.Net Framework)' on the right hand list. At the bottom, select .NET Framework 4.6.1 or later, check the box for making a directory, select a reasonable Location for the Solution/Project, and finally enter a name such as FE5650 Interface. After clicking OK, the Microsoft Visual Studio (MVS) should show Form1.vb. If MVS asks to resize the form to 100%, then agree to it so the form will appear as it would on the PC display. Be sure to 'Save All' at this point.

Make sure the Toolbox (left side), and Solution Explorer and Properties (right side) are both visible. If they are not visible, click the following sequences:

View > Toolbox and View > Solution Explorer

Make sure the 'Serial Port' tool can be found in the Toolbox under the 'All Windows Forms' section. Sometimes the Serial Port needs to be downloaded to the toolbox (especially older versions of Visual Studio). To download it, select the menu sequence

Tools > Choose Toolbox Items

Under the tab for the '.NET Framework Components', check the box next to 'Serial Port'.

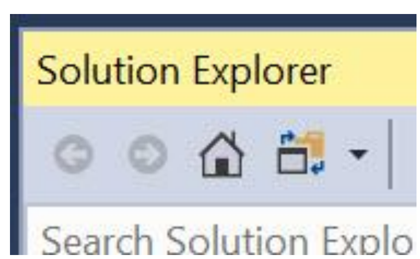


Figure A9.3: Fourth icon from the left should provide access to the Form1.Designer.vb page.

Next the Form1.Designer.vb code (below) needs to be added to the Form1.Designer.vb page. There are a couple of methods to access the Designer.vb page. The first method consists of using the toolbar appearing at the top of **Solution Explorer**. Click on the fourth icon which looks like a folder with a couple of arrows (see Figure A9.3). It might be necessary to click the downward pointing arrow and select the top line showing 'c:…'. Double Click the **Form1.Designer.vb** entry in the list in the Solution Explorer. The second method consists of dragging and dropping a textbox onto the graphical Form1. Double click the textbox. In the coding window, find 'textbox1.textchanged' in the handler. Right click 'textbox1' and select 'Go to definition'. The Form1.Designer.vb page will appear. Go back to the Graphical view of Form1.vb and delete the text box and also delete any code behind (the coding page) and delete the remnants of the textbox subroutine code.

Now select all of the content in the Form1.Designer.vb page and delete it. Copy the code listed in Topic A9.2.2 into the Form1.Designer.vb. Once completed, the GUI for Form1 should appear similar to that shown in Figure A9.1.

Before leaving the issue of the designer code, a few comments should be made on the nature of the code.

1. Visual Studio places the lines of code into Form1.designer.vb and normally, using the GUI of Visual Studio, the programmer doesn't need to write any of them. Most of these lines can be easily understood based on the English language.
2. "MyBase" allows the code to call a member (such as a function) in the base class (sometimes called 'parent class') in order to perform a function for the derived class (sometimes called child class). Form1 is derived from the Form base class. Many methods for Form1 can be accessed through the base class using 'mybase'.
3. Notice the various components/controls are instantiated such as

   Me.TBx_RcvdData=New System.Windows.Forms.TextBox()

   where the 'Me' refers to Form1, TBx_RcvdData is the name of the textbox given by the programmer in the Visual Studio GUI.
4. Further down in the listing, the various properties of the textbox are defined such as

   Me.TBx_RcvdData.Name = "TBx_RcvdData"
   Me.TBx_RcvdData.Font
   Me.TBx_RcvdData.Location = New System.Drawing.Point(10, 28)

   which define the name of the textbox, its font, and the location of the upper left hand corner, respectively.
5. As previously mentioned, the Visual Studio enters all of the parameter information such as that in #4 above into the Properties window on the lower right side of the Visual Studio window. The programmer does the same when using the drag and drop method of constructing the GUI. As a note, the same lines of code can instead be entered into the Functional Code of Form1 if desired.
6. Further down the list, notice the statements similar to

   Me.GBx_SerialPort.Controls.Add(Me.Btn_Conct)

   Here, GBx_SerialPort is the name of a group box simply used to visually divide up Form1 but in this case, it also contains a variety of tools/controls, one of which is a button named 'Btn_conct' (the Connect button for the serial port). The important point is that controls such as buttons are added to a collection of controls for the group box.
7. Page down the listing to lines such as

   Me.GBx_SetFreq.Controls.Add(Me.NUD7)

   The group box is named "GBx_SetFreq". The Add method places the 7th spinner (i.e., NUD7) into the control collection. The NUDs are defined further down the list. T
8. *An important point*, the handler detailed in the source code of the next section iterates through the spinners in the group box to calculate the Fcode. Each NUD has a tag property such as

   Me.NUD7.Tag = "7"

A function accesses the tag property for the calculation. Be sure the tag number increases from right to left as 0 to 7 on the GUI.

9. Finally notice toward the bottom of the list that the variously 'names' are defined as specific components/controls such as

Friend WithEvents TBx_RcvdData As TextBox

The 'WithEvents' keyword indicates the textbox will produce events such as when the text in the textbox is changed. The event handler (see source code in the next section) will do something such as extract numbers from the text to set the spinners. Setting the spinners will cause spinner events to trigger and so the listing also has lines similar to "Friend WithEvents NUD7 As NumericUpDown".

After having copied the listing from Topic A9.2.2 into the Form1.Designer.vb page, proceed to Appendix Section A9.3 for the Form1 functional code. Note some code wraps around to the next line – do not place a carriage return where the wrap occurs – the code should be all one coding line in the program.

Topic A9.2.2: Designer Listing

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()>
Partial Class Form1
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()>
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()>
    Private Sub InitializeComponent()
        Me.TBx_RcvdData = New System.Windows.Forms.TextBox()
        Me.GBx_SerialPort = New System.Windows.Forms.GroupBox()
        Me.Btn_Conct = New System.Windows.Forms.Button()
        Me.TBx_SerPrtBaud = New System.Windows.Forms.TextBox()
        Me.Lbl_Baud = New System.Windows.Forms.Label()
        Me.TBx_SerPrtNum = New System.Windows.Forms.TextBox()
        Me.Lbl_ComNum = New System.Windows.Forms.Label()
        Me.GBx_RcvDat = New System.Windows.Forms.GroupBox()
        Me.CBxAutoClear = New System.Windows.Forms.CheckBox()
        Me.Btn_Clear = New System.Windows.Forms.Button()
        Me.GBx_DefParam = New System.Windows.Forms.GroupBox()
```

```
    Me.BtnCalc = New System.Windows.Forms.Button()
    Me.BtnSave = New System.Windows.Forms.Button()
    Me.BtnGet = New System.Windows.Forms.Button()
    Me.Tbx_DefFreq = New System.Windows.Forms.TextBox()
    Me.Lbl_DefFcode = New System.Windows.Forms.Label()
    Me.TBx_DefFcode = New System.Windows.Forms.TextBox()
    Me.TBx_R = New System.Windows.Forms.TextBox()
    Me.Lbl_DefFreq = New System.Windows.Forms.Label()
    Me.Lbl_DefR = New System.Windows.Forms.Label()
    Me.GroupBox2 = New System.Windows.Forms.GroupBox()
    Me.Btn_Send = New System.Windows.Forms.Button()
    Me.TBx_Cmd = New System.Windows.Forms.TextBox()
    Me.GBx_SetFreq = New System.Windows.Forms.GroupBox()
    Me.CBx_AutoSend = New System.Windows.Forms.CheckBox()
    Me.NUD7 = New System.Windows.Forms.NumericUpDown()
    Me.NUD6 = New System.Windows.Forms.NumericUpDown()
    Me.NUD5 = New System.Windows.Forms.NumericUpDown()
    Me.NUD4 = New System.Windows.Forms.NumericUpDown()
    Me.NUD3 = New System.Windows.Forms.NumericUpDown()
    Me.NUD2 = New System.Windows.Forms.NumericUpDown()
    Me.NUD1 = New System.Windows.Forms.NumericUpDown()
    Me.Btn_SendFreq = New System.Windows.Forms.Button()
    Me.TBx_Fset = New System.Windows.Forms.TextBox()
    Me.NUD0 = New System.Windows.Forms.NumericUpDown()
    Me.GBx_SerialPort.SuspendLayout()
    Me.GBx_RcvDat.SuspendLayout()
    Me.GBx_DefParam.SuspendLayout()
    Me.GroupBox2.SuspendLayout()
    Me.GBx_SetFreq.SuspendLayout()
    CType(Me.NUD7, System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.NUD6, System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.NUD5, System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.NUD4, System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.NUD3, System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.NUD2, System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.NUD1, System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.NUD0, System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()
    '
    'TBx_RcvdData
    '
    Me.TBx_RcvdData.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
    Me.TBx_RcvdData.Location = New System.Drawing.Point(10, 28)
    Me.TBx_RcvdData.Margin = New System.Windows.Forms.Padding(4)
    Me.TBx_RcvdData.Multiline = True
    Me.TBx_RcvdData.Name = "TBx_RcvdData"
    Me.TBx_RcvdData.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
    Me.TBx_RcvdData.Size = New System.Drawing.Size(388, 96)
    Me.TBx_RcvdData.TabIndex = 0
    '
    'GBx_SerialPort
    '
    Me.GBx_SerialPort.Controls.Add(Me.Btn_Conct)
    Me.GBx_SerialPort.Controls.Add(Me.TBx_SerPrtBaud)
    Me.GBx_SerialPort.Controls.Add(Me.Lbl_Baud)
    Me.GBx_SerialPort.Controls.Add(Me.TBx_SerPrtNum)
    Me.GBx_SerialPort.Controls.Add(Me.Lbl_ComNum)
```

```
        Me.GBx_SerialPort.Font = New System.Drawing.Font("Microsoft Sans Serif", 11.25!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.GBx_SerialPort.Location = New System.Drawing.Point(8, 13)
        Me.GBx_SerialPort.Margin = New System.Windows.Forms.Padding(4)
        Me.GBx_SerialPort.Name = "GBx_SerialPort"
        Me.GBx_SerialPort.Padding = New System.Windows.Forms.Padding(4)
        Me.GBx_SerialPort.Size = New System.Drawing.Size(410, 59)
        Me.GBx_SerialPort.TabIndex = 3
        Me.GBx_SerialPort.TabStop = False
        Me.GBx_SerialPort.Text = "Serial Port"
        '
        'Btn_Conct
        '
        Me.Btn_Conct.FlatStyle = System.Windows.Forms.FlatStyle.Popup
        Me.Btn_Conct.Location = New System.Drawing.Point(248, 24)
        Me.Btn_Conct.Name = "Btn_Conct"
        Me.Btn_Conct.Size = New System.Drawing.Size(150, 25)
        Me.Btn_Conct.TabIndex = 4
        Me.Btn_Conct.Text = "Connect"
        Me.Btn_Conct.UseVisualStyleBackColor = True
        '
        'TBx_SerPrtBaud
        '
        Me.TBx_SerPrtBaud.Location = New System.Drawing.Point(159, 24)
        Me.TBx_SerPrtBaud.Name = "TBx_SerPrtBaud"
        Me.TBx_SerPrtBaud.Size = New System.Drawing.Size(83, 24)
        Me.TBx_SerPrtBaud.TabIndex = 3
        Me.TBx_SerPrtBaud.Text = "9600"
        Me.TBx_SerPrtBaud.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Lbl_Baud
        '
        Me.Lbl_Baud.AutoSize = True
        Me.Lbl_Baud.Location = New System.Drawing.Point(121, 27)
        Me.Lbl_Baud.Name = "Lbl_Baud"
        Me.Lbl_Baud.Size = New System.Drawing.Size(42, 18)
        Me.Lbl_Baud.TabIndex = 2
        Me.Lbl_Baud.Text = "Baud"
        '
        'TBx_SerPrtNum
        '
        Me.TBx_SerPrtNum.Location = New System.Drawing.Point(71, 24)
        Me.TBx_SerPrtNum.Name = "TBx_SerPrtNum"
        Me.TBx_SerPrtNum.Size = New System.Drawing.Size(44, 24)
        Me.TBx_SerPrtNum.TabIndex = 1
        Me.TBx_SerPrtNum.Text = "4"
        Me.TBx_SerPrtNum.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Lbl_ComNum
        '
        Me.Lbl_ComNum.AutoSize = True
        Me.Lbl_ComNum.Location = New System.Drawing.Point(7, 27)
        Me.Lbl_ComNum.Name = "Lbl_ComNum"
        Me.Lbl_ComNum.Size = New System.Drawing.Size(66, 18)
        Me.Lbl_ComNum.TabIndex = 0
        Me.Lbl_ComNum.Text = "Comm #"
        '
        'GBx_RcvDat
```

```
'
Me.GBx_RcvDat.Controls.Add(Me.CBxAutoClear)
Me.GBx_RcvDat.Controls.Add(Me.Btn_Clear)
Me.GBx_RcvDat.Controls.Add(Me.TBx_RcvdData)
Me.GBx_RcvDat.Location = New System.Drawing.Point(8, 79)
Me.GBx_RcvDat.Name = "GBx_RcvDat"
Me.GBx_RcvDat.Size = New System.Drawing.Size(410, 132)
Me.GBx_RcvDat.TabIndex = 5
Me.GBx_RcvDat.TabStop = False
Me.GBx_RcvDat.Text = "Received Data"
'
'CBxAutoClear
'
Me.CBxAutoClear.Appearance = System.Windows.Forms.Appearance.Button
Me.CBxAutoClear.AutoSize = True
Me.CBxAutoClear.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
Me.CBxAutoClear.Location = New System.Drawing.Point(356, 0)
Me.CBxAutoClear.Name = "CBxAutoClear"
Me.CBxAutoClear.Size = New System.Drawing.Size(24, 25)
Me.CBxAutoClear.TabIndex = 35
Me.CBxAutoClear.Text = "A"
Me.CBxAutoClear.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
Me.CBxAutoClear.UseVisualStyleBackColor = True
'
'Btn_Clear
'
Me.Btn_Clear.Location = New System.Drawing.Point(383, 0)
Me.Btn_Clear.Name = "Btn_Clear"
Me.Btn_Clear.Size = New System.Drawing.Size(26, 25)
Me.Btn_Clear.TabIndex = 3
Me.Btn_Clear.Text = "C"
Me.Btn_Clear.UseVisualStyleBackColor = True
'
'GBx_DefParam
'
Me.GBx_DefParam.Controls.Add(Me.BtnCalc)
Me.GBx_DefParam.Controls.Add(Me.BtnSave)
Me.GBx_DefParam.Controls.Add(Me.BtnGet)
Me.GBx_DefParam.Controls.Add(Me.Tbx_DefFreq)
Me.GBx_DefParam.Controls.Add(Me.Lbl_DefFcode)
Me.GBx_DefParam.Controls.Add(Me.TBx_DefFcode)
Me.GBx_DefParam.Controls.Add(Me.TBx_R)
Me.GBx_DefParam.Controls.Add(Me.Lbl_DefFreq)
Me.GBx_DefParam.Controls.Add(Me.Lbl_DefR)
Me.GBx_DefParam.Location = New System.Drawing.Point(8, 221)
Me.GBx_DefParam.Name = "GBx_DefParam"
Me.GBx_DefParam.Size = New System.Drawing.Size(410, 92)
Me.GBx_DefParam.TabIndex = 6
Me.GBx_DefParam.TabStop = False
Me.GBx_DefParam.Text = "Default Parameters"
'
'BtnCalc
'
Me.BtnCalc.Location = New System.Drawing.Point(345, 23)
Me.BtnCalc.Name = "BtnCalc"
Me.BtnCalc.Size = New System.Drawing.Size(53, 25)
Me.BtnCalc.TabIndex = 10
```

```
        Me.BtnCalc.Text = "Calc"
        Me.BtnCalc.UseVisualStyleBackColor = True
        '
        'BtnSave
        '
        Me.BtnSave.Location = New System.Drawing.Point(286, 23)
        Me.BtnSave.Name = "BtnSave"
        Me.BtnSave.Size = New System.Drawing.Size(53, 25)
        Me.BtnSave.TabIndex = 9
        Me.BtnSave.Text = "Save"
        Me.BtnSave.UseVisualStyleBackColor = True
        '
        'BtnGet
        '
        Me.BtnGet.Location = New System.Drawing.Point(227, 23)
        Me.BtnGet.Name = "BtnGet"
        Me.BtnGet.Size = New System.Drawing.Size(53, 25)
        Me.BtnGet.TabIndex = 8
        Me.BtnGet.Text = "Get"
        Me.BtnGet.UseVisualStyleBackColor = True
        '
        'Tbx_DefFreq
        '
        Me.Tbx_DefFreq.Location = New System.Drawing.Point(62, 26)
        Me.Tbx_DefFreq.Name = "Tbx_DefFreq"
        Me.Tbx_DefFreq.Size = New System.Drawing.Size(150, 24)
        Me.Tbx_DefFreq.TabIndex = 7
        Me.Tbx_DefFreq.Text = "10000000.000"
        Me.Tbx_DefFreq.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Lbl_DefFcode
        '
        Me.Lbl_DefFcode.AutoSize = True
        Me.Lbl_DefFcode.Location = New System.Drawing.Point(6, 60)
        Me.Lbl_DefFcode.Name = "Lbl_DefFcode"
        Me.Lbl_DefFcode.Size = New System.Drawing.Size(59, 18)
        Me.Lbl_DefFcode.TabIndex = 6
        Me.Lbl_DefFcode.Text = "Fcode="
        '
        'TBx_DefFcode
        '
        Me.TBx_DefFcode.Location = New System.Drawing.Point(62, 57)
        Me.TBx_DefFcode.Name = "TBx_DefFcode"
        Me.TBx_DefFcode.Size = New System.Drawing.Size(150, 24)
        Me.TBx_DefFcode.TabIndex = 5
        Me.TBx_DefFcode.Text = "32F0AD80"
        Me.TBx_DefFcode.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'TBx_R
        '
        Me.TBx_R.BackColor = System.Drawing.Color.FromArgb(CType(CType(192, Byte), Integer), CType(CType(255, Byte),
Integer), CType(CType(255, Byte), Integer))
        Me.TBx_R.ForeColor = System.Drawing.Color.Blue
        Me.TBx_R.Location = New System.Drawing.Point(247, 57)
        Me.TBx_R.Name = "TBx_R"
        Me.TBx_R.ReadOnly = True
        Me.TBx_R.Size = New System.Drawing.Size(150, 24)
        Me.TBx_R.TabIndex = 2
```

```
Me.TBx_R.Text = "50255056.780713"
Me.TBx_R.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
'
'Lbl_DefFreq
'
Me.Lbl_DefFreq.AutoSize = True
Me.Lbl_DefFreq.Location = New System.Drawing.Point(18, 29)
Me.Lbl_DefFreq.Name = "Lbl_DefFreq"
Me.Lbl_DefFreq.Size = New System.Drawing.Size(47, 18)
Me.Lbl_DefFreq.TabIndex = 1
Me.Lbl_DefFreq.Text = "Fout="
'
'Lbl_DefR
'
Me.Lbl_DefR.AutoSize = True
Me.Lbl_DefR.Location = New System.Drawing.Point(224, 60)
Me.Lbl_DefR.Name = "Lbl_DefR"
Me.Lbl_DefR.Size = New System.Drawing.Size(28, 18)
Me.Lbl_DefR.TabIndex = 0
Me.Lbl_DefR.Text = "R="
'
'GroupBox2
'
Me.GroupBox2.Controls.Add(Me.Btn_Send)
Me.GroupBox2.Controls.Add(Me.TBx_Cmd)
Me.GroupBox2.Location = New System.Drawing.Point(8, 323)
Me.GroupBox2.Name = "GroupBox2"
Me.GroupBox2.Size = New System.Drawing.Size(409, 66)
Me.GroupBox2.TabIndex = 7
Me.GroupBox2.TabStop = False
Me.GroupBox2.Text = "Send Command:     S Status, R= ref, F= Fcode, E enter"
'
'Btn_Send
'
Me.Btn_Send.FlatStyle = System.Windows.Forms.FlatStyle.Popup
Me.Btn_Send.Location = New System.Drawing.Point(296, 26)
Me.Btn_Send.Name = "Btn_Send"
Me.Btn_Send.Size = New System.Drawing.Size(84, 25)
Me.Btn_Send.TabIndex = 5
Me.Btn_Send.Text = "Send"
Me.Btn_Send.UseVisualStyleBackColor = True
'
'TBx_Cmd
'
Me.TBx_Cmd.Location = New System.Drawing.Point(38, 26)
Me.TBx_Cmd.Name = "TBx_Cmd"
Me.TBx_Cmd.Size = New System.Drawing.Size(252, 24)
Me.TBx_Cmd.TabIndex = 4
'
'GBx_SetFreq
'
Me.GBx_SetFreq.Controls.Add(Me.CBx_AutoSend)
Me.GBx_SetFreq.Controls.Add(Me.NUD7)
Me.GBx_SetFreq.Controls.Add(Me.NUD6)
Me.GBx_SetFreq.Controls.Add(Me.NUD5)
Me.GBx_SetFreq.Controls.Add(Me.NUD4)
Me.GBx_SetFreq.Controls.Add(Me.NUD3)
Me.GBx_SetFreq.Controls.Add(Me.NUD2)
```

```
        Me.GBx_SetFreq.Controls.Add(Me.NUD1)
        Me.GBx_SetFreq.Controls.Add(Me.Btn_SendFreq)
        Me.GBx_SetFreq.Controls.Add(Me.TBx_Fset)
        Me.GBx_SetFreq.Controls.Add(Me.NUD0)
        Me.GBx_SetFreq.Location = New System.Drawing.Point(8, 395)
        Me.GBx_SetFreq.Name = "GBx_SetFreq"
        Me.GBx_SetFreq.Size = New System.Drawing.Size(409, 113)
        Me.GBx_SetFreq.TabIndex = 8
        Me.GBx_SetFreq.TabStop = False
        Me.GBx_SetFreq.Text = "Set Frequency"
        '
        'CBx_AutoSend
        '
        Me.CBx_AutoSend.Appearance = System.Windows.Forms.Appearance.Button
        Me.CBx_AutoSend.AutoSize = True
        Me.CBx_AutoSend.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.CBx_AutoSend.Location = New System.Drawing.Point(383, 0)
        Me.CBx_AutoSend.Name = "CBx_AutoSend"
        Me.CBx_AutoSend.Size = New System.Drawing.Size(24, 25)
        Me.CBx_AutoSend.TabIndex = 34
        Me.CBx_AutoSend.Text = "A"
        Me.CBx_AutoSend.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.CBx_AutoSend.UseVisualStyleBackColor = True
        '
        'NUD7
        '
        Me.NUD7.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.NUD7.Hexadecimal = True
        Me.NUD7.Location = New System.Drawing.Point(17, 25)
        Me.NUD7.Maximum = New Decimal(New Integer() {15, 0, 0, 0})
        Me.NUD7.Name = "NUD7"
        Me.NUD7.Size = New System.Drawing.Size(40, 35)
        Me.NUD7.TabIndex = 33
        Me.NUD7.Tag = "7"
        Me.NUD7.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        Me.NUD7.Value = New Decimal(New Integer() {3, 0, 0, 0})
        '
        'NUD6
        '
        Me.NUD6.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.NUD6.Hexadecimal = True
        Me.NUD6.Location = New System.Drawing.Point(63, 25)
        Me.NUD6.Maximum = New Decimal(New Integer() {15, 0, 0, 0})
        Me.NUD6.Name = "NUD6"
        Me.NUD6.Size = New System.Drawing.Size(40, 35)
        Me.NUD6.TabIndex = 32
        Me.NUD6.Tag = "6"
        Me.NUD6.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        Me.NUD6.Value = New Decimal(New Integer() {2, 0, 0, 0})
        '
        'NUD5
        '
        Me.NUD5.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.NUD5.Hexadecimal = True
```

```
        Me.NUD5.Location = New System.Drawing.Point(109, 25)
        Me.NUD5.Maximum = New Decimal(New Integer() {15, 0, 0, 0})
        Me.NUD5.Name = "NUD5"
        Me.NUD5.Size = New System.Drawing.Size(40, 35)
        Me.NUD5.TabIndex = 31
        Me.NUD5.Tag = "5"
        Me.NUD5.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        Me.NUD5.Value = New Decimal(New Integer() {15, 0, 0, 0})
        '
        'NUD4
        '
        Me.NUD4.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.NUD4.Hexadecimal = True
        Me.NUD4.Location = New System.Drawing.Point(155, 25)
        Me.NUD4.Maximum = New Decimal(New Integer() {15, 0, 0, 0})
        Me.NUD4.Name = "NUD4"
        Me.NUD4.Size = New System.Drawing.Size(40, 35)
        Me.NUD4.TabIndex = 30
        Me.NUD4.Tag = "4"
        Me.NUD4.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'NUD3
        '
        Me.NUD3.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.NUD3.Hexadecimal = True
        Me.NUD3.Location = New System.Drawing.Point(201, 25)
        Me.NUD3.Maximum = New Decimal(New Integer() {15, 0, 0, 0})
        Me.NUD3.Name = "NUD3"
        Me.NUD3.Size = New System.Drawing.Size(40, 35)
        Me.NUD3.TabIndex = 29
        Me.NUD3.Tag = "3"
        Me.NUD3.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        Me.NUD3.Value = New Decimal(New Integer() {10, 0, 0, 0})
        '
        'NUD2
        '
        Me.NUD2.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.NUD2.Hexadecimal = True
        Me.NUD2.Location = New System.Drawing.Point(247, 25)
        Me.NUD2.Maximum = New Decimal(New Integer() {15, 0, 0, 0})
        Me.NUD2.Name = "NUD2"
        Me.NUD2.Size = New System.Drawing.Size(40, 35)
        Me.NUD2.TabIndex = 28
        Me.NUD2.Tag = "2"
        Me.NUD2.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        Me.NUD2.Value = New Decimal(New Integer() {13, 0, 0, 0})
        '
        'NUD1
        '
        Me.NUD1.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.NUD1.Hexadecimal = True
        Me.NUD1.Location = New System.Drawing.Point(293, 25)
        Me.NUD1.Maximum = New Decimal(New Integer() {15, 0, 0, 0})
        Me.NUD1.Name = "NUD1"
```

```
        Me.NUD1.Size = New System.Drawing.Size(40, 35)
        Me.NUD1.TabIndex = 27
        Me.NUD1.Tag = "1"
        Me.NUD1.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        Me.NUD1.Value = New Decimal(New Integer() {8, 0, 0, 0})
        '
        'Btn_SendFreq
        '
        Me.Btn_SendFreq.FlatStyle = System.Windows.Forms.FlatStyle.Popup
        Me.Btn_SendFreq.Location = New System.Drawing.Point(238, 69)
        Me.Btn_SendFreq.Name = "Btn_SendFreq"
        Me.Btn_SendFreq.Size = New System.Drawing.Size(103, 35)
        Me.Btn_SendFreq.TabIndex = 26
        Me.Btn_SendFreq.Text = "Send"
        Me.Btn_SendFreq.UseVisualStyleBackColor = True
        '
        'TBx_Fset
        '
        Me.TBx_Fset.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.TBx_Fset.Location = New System.Drawing.Point(62, 69)
        Me.TBx_Fset.Name = "TBx_Fset"
        Me.TBx_Fset.Size = New System.Drawing.Size(161, 35)
        Me.TBx_Fset.TabIndex = 25
        Me.TBx_Fset.Text = "10000000.000"
        Me.TBx_Fset.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'NUD0
        '
        Me.NUD0.Font = New System.Drawing.Font("Microsoft Sans Serif", 18.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.NUD0.Hexadecimal = True
        Me.NUD0.Location = New System.Drawing.Point(339, 25)
        Me.NUD0.Maximum = New Decimal(New Integer() {15, 0, 0, 0})
        Me.NUD0.Name = "NUD0"
        Me.NUD0.Size = New System.Drawing.Size(40, 35)
        Me.NUD0.TabIndex = 24
        Me.NUD0.Tag = "0"
        Me.NUD0.TextAlign = System.Windows.Forms.HorizontalAlignment.Center
        '
        'Form1
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(9.0!, 18.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(434, 521)
        Me.Controls.Add(Me.GBx_SetFreq)
        Me.Controls.Add(Me.GroupBox2)
        Me.Controls.Add(Me.GBx_DefParam)
        Me.Controls.Add(Me.GBx_RcvDat)
        Me.Controls.Add(Me.GBx_SerialPort)
        Me.Font = New System.Drawing.Font("Microsoft Sans Serif", 11.25!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Name = "Form1"
        Me.Text = "FE5650A Interface"
        Me.GBx_SerialPort.ResumeLayout(False)
        Me.GBx_SerialPort.PerformLayout()
        Me.GBx_RcvDat.ResumeLayout(False)
        Me.GBx_RcvDat.PerformLayout()
```

```
        Me.GBx_DefParam.ResumeLayout(False)
        Me.GBx_DefParam.PerformLayout()
        Me.GroupBox2.ResumeLayout(False)
        Me.GroupBox2.PerformLayout()
        Me.GBx_SetFreq.ResumeLayout(False)
        Me.GBx_SetFreq.PerformLayout()
        CType(Me.NUD7, System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.NUD6, System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.NUD5, System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.NUD4, System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.NUD3, System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.NUD2, System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.NUD1, System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.NUD0, System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)

    End Sub

    Friend WithEvents TBx_RcvdData As TextBox
    Friend WithEvents GBx_SerialPort As GroupBox
    Friend WithEvents TBx_SerPrtNum As TextBox
    Friend WithEvents Lbl_ComNum As Label
    Friend WithEvents Btn_Conct As Button
    Friend WithEvents TBx_SerPrtBaud As TextBox
    Friend WithEvents Lbl_Baud As Label
    Friend WithEvents GBx_RcvDat As GroupBox
    Friend WithEvents GBx_DefParam As GroupBox
    Friend WithEvents TBx_R As TextBox
    Friend WithEvents Lbl_DefFreq As Label
    Friend WithEvents Lbl_DefR As Label
    Friend WithEvents Btn_Clear As Button
    Friend WithEvents GroupBox2 As GroupBox
    Friend WithEvents TBx_DefFcode As TextBox
    Friend WithEvents Tbx_DefFreq As TextBox
    Friend WithEvents Lbl_DefFcode As Label
    Friend WithEvents Btn_Send As Button
    Friend WithEvents TBx_Cmd As TextBox
    Friend WithEvents GBx_SetFreq As GroupBox
    Friend WithEvents CBx_AutoSend As CheckBox
    Friend WithEvents NUD7 As NumericUpDown
    Friend WithEvents NUD6 As NumericUpDown
    Friend WithEvents NUD5 As NumericUpDown
    Friend WithEvents NUD4 As NumericUpDown
    Friend WithEvents NUD3 As NumericUpDown
    Friend WithEvents NUD2 As NumericUpDown
    Friend WithEvents NUD1 As NumericUpDown
    Friend WithEvents Btn_SendFreq As Button
    Friend WithEvents TBx_Fset As TextBox
    Friend WithEvents NUD0 As NumericUpDown
    Friend WithEvents BtnSave As Button
    Friend WithEvents BtnGet As Button
    Friend WithEvents BtnCalc As Button
    Friend WithEvents CBxAutoClear As CheckBox
End Class
```

Section A9.3: Form1 Source Code for the FE5650A Interface

The previous section provided the code listing for the Graphical User Interface GUI for the FE5650A Interface software. Now the various controls/components need to be made functional by placing code on the Form1.vb page. As previously discussed, the GUI design could have been made by simply dragging-dropping the various tools/controls into the Visual Studio GUI design area and using the parameters listed in the previous section to fill-in the properties box for each component. Having the source code in this appendix makes it easy to modify and improve the software as the programmer sees fit for the application. Some comments on the source code can be found in Topic A9.3.3 after the source code listing.

Topic A9.3.1: Notes on Form1.vb and the EXE file

The FE5650A Interface GUI should be visible in Visual Studio. In the Solution Explorer window, *right* click Form1.vb under the folder named 'FE5650A Interface'. Select 'View Code'. The functional code window should be displayed. Delete any initial code on the page. The code listed below should be copied directly into the page. Once completed, make sure any errors have been resolved, and then type CTRL F5, or else click the 'Start' on the tool bar just below the main menu.

As a note, once the program has successfully run, the EXE file can be added to the desktop and run by double clicking on the icon. To access the EXE file, open the directory containing the FE5650 Interface program using Windows Explorer. The directory will contain FE5650 Interface.sln and a folder labeled as FE5650 Interface. Open this second folder and then open the 'bin' folder and then copy the .exe file. It should be pointed out that Visual Studio can make an installer (MSI) although it involves some extra steps. The .exe file by itself does not install in the Windows registry.

Topic A9.3.2: Form1.vb Source Code

```
Imports System.IO
Imports System.Text
Imports System.IO.Ports

Public Class Form1

  Structure Flags
    Friend AllowSpinHandler As Boolean
    Friend AllowFreqHandler As Boolean
    Friend Form1Loaded As Boolean
    Sub New(ByVal value As Boolean)
      AllowSpinHandler = value
      AllowFreqHandler = value
      Form1Loaded = False
    End Sub
  End Structure
  Friend Flag As New Flags(True)

  Friend WithEvents SerialPort1 As IO.Ports.SerialPort
  Friend WithEvents OFD As OpenFileDialog
  Friend WithEvents SFD As SaveFileDialog
```

```
Structure Matrix
   Friend buf As Byte()
   Friend len As Byte
   Sub New(ByVal n As Byte)
      ReDim buf(n)
   End Sub
End Structure

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
   MyBase.Text = "FE5650A Interface"
   Me.SerialPort1 = New System.IO.Ports.SerialPort()
   Dim Count As Integer = My.Computer.Ports.SerialPortNames.Count
   Select Case Count
      Case > 1
         For Each s In My.Computer.Ports.SerialPortNames
            TBx_RcvdData.Text = s + vbLf + vbCr
         Next s
      Case = 1
         Dim S As String = My.Computer.Ports.SerialPortNames(0)
         S = S.Substring(3)
         TBx_SerPrtNum.Text = S.Trim
      Case = 0
         TBx_RcvdData.Text = "No Comm Port/Devices - attach/install USB device"
   End Select
   SubUpdateNUDs({0, 8, 13, 10, 0, 15, 2, 3})
   Me.OFD = New OpenFileDialog()
   Me.SFD = New SaveFileDialog()
   OFD.InitialDirectory = My.Application.Info.DirectoryPath
   SFD.InitialDirectory = My.Application.Info.DirectoryPath
   If TBx_R.ReadOnly Then
      TBx_R.BackColor = Color.FromArgb(255, 180, 255, 255)
   Else
      TBx_R.BackColor = Color.White
   End If
   Flag.Form1Loaded = True
End Sub

Private Sub Form1_Closing(sender As Object, e As EventArgs) Handles MyBase.Closing
   If SerialPort1.IsOpen Then SerialPort1.Close()
End Sub

Private Sub Btn_Conct_Click(sender As Object, e As EventArgs) Handles Btn_Conct.Click
   Try
      If SerialPort1.IsOpen Then  'Close port
         SerialPort1.Close()
         Btn_Conct.Text = "Connect"
      Else    'Read parms and open port
         With SerialPort1
            .BaudRate = Integer.Parse(TBx_SerPrtBaud.Text)
            .Parity = Parity.None
            .DataBits = 8
            .StopBits = 1
            .Handshake = Handshake.None
            .PortName = "COM" + (Integer.Parse(TBx_SerPrtNum.Text)).ToString
            .ReadBufferSize = 512
            .ReadTimeout = 500
         End With
         SerialPort1.Open()
```

```
            Btn_Conct.Text = "Disconnect"
         End If
      Catch ex As Exception
         MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK)
      End Try
   End Sub

   Private Sub Btn_Clear_Click(sender As Object, e As EventArgs) Handles Btn_Clear.Click
      TBx_RcvdData.Text = ""
   End Sub

   Private Sub Btn_Send_Click(sender As Object, e As EventArgs) Handles Btn_Send.Click
      Dim cmdSend As String = TBx_Cmd.Text + vbCr 'note vbcr included
      SendCmd_WriteTBx(cmdSend)
   End Sub

   Private Sub Btn_SendFreq_Click(sender As Object, e As EventArgs) Handles Btn_SendFreq.Click
      Dim FcodeU32 As UInt32 = FnFcodeU32_fromNUDs()
      Dim FcodeHexDigBytes() As Byte = FnU32ToHexDig(FcodeU32) 'array will have 8 numeric bytes
      Dim FcodeStr As String = ""
      For i As Integer = 7 To 0 Step -1 : FcodeStr += Hex(FcodeHexDigBytes(i)) : Next i 'String.Format("{0:X2}", FcodeBytes(i)) :
Next i
      Dim cmdSend As String = "F=" + FcodeStr + vbCr
      SendCmd_WriteTBx(cmdSend)
   End Sub

   Private Sub SendCmd_WriteTBx(ByVal cmdSend As String)
      Dim FE5650byte As New Matrix(100)
      FE5650byte = FnComm(cmdSend)
      If CBxAutoClear.Checked Then TBx_RcvdData.Text = ""
      TBx_RcvdData.Text += System.Text.Encoding.ASCII.GetString(FE5650byte.buf)
      TBx_RcvdData.Text += vbCrLf
      'Print HEX version of FE5650A returned data
      For i As Integer = 0 To FE5650byte.len - 1
         TBx_RcvdData.Text += String.Format("{0:X2}", FE5650byte.buf(i)) + " "
      Next i
      TBx_RcvdData.Text += vbCrLf
   End Sub

   Function FnComm(ByVal cmdSend As String) As Matrix
      Dim FE5650byte As New Matrix(100)
      FE5650byte.len = 0
      Try 'check serial port and stops loop at time out
         SerialPort1.Write(cmdSend)
         Do
            FE5650byte.buf(FE5650byte.len) = SerialPort1.ReadByte
            FE5650byte.len += 1
         Loop
      Catch ex As Exception
      End Try
      Return FE5650byte
   End Function

   'Calc Fout from Fcode and Ref
   Function FnFout(ByVal FcodeDec As Decimal, ByVal Ref As Decimal) As Decimal
      Return ((Ref / 2 ^ 32) * FcodeDec)
   End Function
```

```
Function FnRef(ByVal FoutDec As Decimal, ByVal FcodeDec As Decimal) As Decimal
   Return ((2 ^ 32 / FcodeDec) * FoutDec)
End Function

'Calc Fcode from Fout and Ref
Function FnFcode(ByVal Fout As Decimal, ByVal Ref As Decimal) As Decimal
   Dim fnFcode1 As Decimal = 0
   fnFcode1 = (Fout / Ref) * 2 ^ 32
   Return Math.Round(fnFcode1, 0, MidpointRounding.AwayFromZero)
End Function

Private Function FnFcodeU32_fromNUDs() As UInt32
   Dim FcodeDec As Decimal = 0
   For Each O As Object In GBx_SetFreq.Controls
      If (TypeOf O Is NumericUpDown) Then
         FcodeDec += O.value * (16 ^ O.tag)
      End If
   Next O
   Return FcodeDec
End Function

Private Function FnU32ToHexDig(ByVal FcodeU32 As UInt32) As Byte()
   Dim HexDigitArray(7) As Byte 'holds 8 digits
   For i As Integer = 0 To 7   'includes leading zeros
      HexDigitArray(i) = FcodeU32 Mod 16
      FcodeU32 = Math.Floor(FcodeU32 / 16) 'prevents rounding up to next number
   Next i
   Return HexDigitArray
End Function

'updates NumericUpDown controls. Input: Array(0)=LSB
Private Sub SubUpdateNUDs(ByVal HexDigArray() As Byte)
   For Each O As Object In GBx_SetFreq.Controls
      If (TypeOf O Is NumericUpDown) Then
         O.value = HexDigArray(O.Tag)
      End If
   Next O
End Sub

Private Sub NUD_ValueChanged(sender As Object, e As EventArgs) Handles _
      NUD0.ValueChanged, NUD1.ValueChanged, NUD2.ValueChanged, NUD3.ValueChanged,
      NUD4.ValueChanged, NUD5.ValueChanged, NUD6.ValueChanged, NUD7.ValueChanged
   If Not Flag.Form1Loaded Then Exit Sub
   If Not Flag.AllowSpinHandler Then Exit Sub
   Flag.AllowFreqHandler = False
   Try
      Dim FcodeDec As Decimal = FnFcodeU32_fromNUDs()
      Dim Rdec As Decimal = Decimal.Parse(TBx_R.Text.Trim)
      TBx_Fset.Text = (Math.Round(FnFout(FcodeDec, Rdec), 3)).ToString("#.000")
      If CBx_AutoSend.Checked Then 'And Flag.AllowSpinHandler Then
         Dim cmdSend As String = "F=" + Hex(FcodeDec) + vbCr
         SendCmd_WriteTBx(cmdSend)
      End If
   Catch ex As Exception
      MsgBox(ex.ToString)
   End Try
   Flag.AllowFreqHandler = True
End Sub
```

```
Private Sub TBx_Fset_TextChanged(sender As Object, e As EventArgs) Handles TBx_Fset.TextChanged
    If Not Flag.Form1Loaded Then Exit Sub
    If Not Flag.AllowFreqHandler Then Exit Sub
    If TBx_Fset.Text = "" Then Exit Sub
    Flag.AllowSpinHandler = False
    Try
        Dim FrqDec As Decimal = Decimal.Parse(TBx_Fset.Text.Trim)
        Dim Rdec As Decimal = Decimal.Parse(TBx_R.Text.Trim)
        Dim FcodeDec As Decimal = Math.Round(FnFcode(FrqDec, Rdec), 0, MidpointRounding.ToEven)
        Dim FcodeU32 = CType(FcodeDec, UInt32)
        SubUpdateNUDs(FnU32ToHexDig(FcodeU32))
        'FcodeDec = FnFcodeU32_fromNUDs()
        'FrqDec = FnFout(FcodeDec, Rdec)
    Catch ex As Exception
    End Try
    Flag.AllowSpinHandler = True
End Sub

Private Sub BtnGet_Click(sender As Object, e As EventArgs) Handles BtnGet.Click
    Try
        If OFD.ShowDialog() = DialogResult.OK Then
            Dim FilePathAndName As String = OFD.FileName '+ vbCrLf
            Dim file As System.IO.StreamReader
            file = My.Computer.FileSystem.OpenTextFileReader(FilePathAndName)
            Tbx_DefFreq.Text = file.ReadLine()
            TBx_DefFcode.Text = file.ReadLine()
            TBx_R.Text = file.ReadLine()
            file.Close()
        End If
    Catch ex As Exception
        MessageBox.Show(ex.ToString, ErrorToString, MessageBoxButtons.OK)
    End Try
End Sub

Private Sub BtnSave_Click(sender As Object, e As EventArgs) Handles BtnSave.Click
    Try
        If SFD.ShowDialog() = DialogResult.OK Then
            Dim FilePathAndName As String = SFD.FileName '+ vbCrLf
            Dim file As System.IO.StreamWriter
            file = My.Computer.FileSystem.OpenTextFileWriter(FilePathAndName, False)
            file.WriteLine(Tbx_DefFreq.Text)
            file.WriteLine(TBx_DefFcode.Text)
            file.WriteLine(TBx_R.Text)
            file.Close()
        End If
    Catch ex As Exception
        MessageBox.Show(ex.ToString, ErrorToString, MessageBoxButtons.OK)
    End Try
End Sub
Private Sub TBx_R_MouseDoubleClick(sender As Object, e As MouseEventArgs) Handles TBx_R.MouseDoubleClick
    TBx_R.ReadOnly = Not TBx_R.ReadOnly
    If TBx_R.ReadOnly Then
        TBx_R.BackColor = Color.FromArgb(255, 180, 255, 255)
    Else
        TBx_R.BackColor = Color.White
    End If
    TBx_R.Select(TBx_R.Text.Length, 0)
```

```
    End Sub
    Private Sub BtnCalc_Click(sender As Object, e As EventArgs) Handles BtnCalc.Click
        Try
            Dim FoutDec As Decimal = Decimal.Parse(Tbx_DefFreq.Text.Trim)
            Dim FcodeStr As String = TBx_DefFcode.Text.Trim
            Dim FcodeDec As Decimal = CType(Convert.ToInt32(FcodeStr, 16), Decimal)
            Dim Ref As Decimal = FnRef(FoutDec, FcodeDec)
            TBx_R.Text = String.Format(Ref.ToString, "#0.#######")
        Catch ex As Exception
        End Try
    End Sub

    Private Sub CBxAutoClear_CheckedChanged(sender As Object, e As EventArgs) Handles CBxAutoClear.CheckedChanged
        If CBxAutoClear.Checked Then
            CBxAutoClear.BackColor = Color.Yellow
        Else
            CBxAutoClear.BackColor = Color.Transparent
        End If
    End Sub

    Private Sub CBx_AutoSend_CheckedChanged(sender As Object, e As EventArgs) Handles CBx_AutoSend.CheckedChanged
        If CBx_AutoSend.Checked Then
            CBx_AutoSend.BackColor = Color.Yellow
        Else
            CBx_AutoSend.BackColor = Color.Transparent
        End If
    End Sub

End Class
```

Topic A9.3.3: Comments on the Form1 Source Code

The functionality of the source code in Topic A9.3.2 can be mostly ascertained from the flow charts (Figure A9.2). While there are many details, there are several aspects to the programming: (i) Design of the GUI as in Section A9.2. (ii) VB.net is fully object oriented (as are other .net languages) and so the program must define and instantiate the objects of interest. (iii) Given that VB.net is event driven, most of the computation will take place in response to events. (iv) The program must be packed up for an installer MSI or for use as a stand-alone EXE file. The EXE file doesn't require any additional effort.

Some variables are defined (using DIM or FREIEND or PUBLIC) at the modular level (i.e., global level) just under the line "Public Class Form1".

1. A structure is defined for Booleans used in the program although it only serves as a convenient collection of the relevant flags. The structure has the flags to allow the NUD handler and the Frequency Textbox handler to run as previously discussed.
2. The names 'SerialPort1', OFD, SFD are next associated with a Serial Port, Open File Dialog and Save File Dialog, respectively. These do not need to be instantiated using the 'new' keyword till later. As mentioned previously, OFD and SFD provide the functionality for the 'Get' and 'Save' button related to the Default Parameters of R, Fout and Fcode.
3. The Matrix structure defines an array 'buf' and carries with it the length. Normally, string arrays have length functions that can determine their length. The length is determined by searching for the ASCii null character ASCII(0) or \0 or just plain 0. However, data that's not string will have

data that is sometimes zero. So another scheme is required. It is also possible to use the last element of an array to hold the number of relevant entries which might not be the same as the total number of available entries. The program makes it explicit by carrying the length of the relevant entries along with the array in the structure.

Next the various event handlers, user defined functions and subroutines are defined. Of course the event handlers will have the ‘Handles’ keyword toward the end. A given event handler can handle several of the same type of event as will be seen for the Numeric Up Down (NUD) control.

4. The ‘Form1_Load’ handler instantiates the serial port (setup the required memory locations). It looks for Com.Ports (i.e., a USB port with the USB-RS232 adapter) using the My.Computer.Ports class/object and, using a Select Case, sends the results to the textbox for received data TBx_RcvdData or directly sets the Serial Port Number. The subroutine SubUpdateNUDs provides an initial setting for the spinners (NUDs). The OFD and SFD initial directories are set to the same directory as hosting the FE5650A Interface program.
5. The button ‘Btn_Conct’ handler opens the serial port if possible (as mediated by the ‘Try’) and assigns the relevant parameters. The port will read up to 512 bytes to an internal buffer. The ReadTimeout was set to 500mSec to allow the FE5650A plenty of time to send relevant data.
6. The button ‘Btn_Send’ handler calls the subroutine SendCmd_WriteTBx to send a command to the FE5650A and receive a response, and write the received data to TBx_RcvdData.
7. The button ‘Btn_SendFreq’ handler uses separate functions to translate the decimal form of the HEX digits on the spinners (NUDs) to UInt32 in FcodeU32; the spinners show Hex digits 0-F but stores it as a byte 0-15 (a little inconvenient but that’s how it works). The UInt32 number is then converted to a byte array of 8 elements with each element for one of the hex digits. Using a ‘For’ loop and a HEX function that converts a byte to a Hex ASCII character. The handler sends the string along with a <cr> to the FE5650A and then waits for the returned data (up to 500mSec) using the subroutine ‘SendCmd_WriteTBx’.
8. The subroutine ‘SendCmd_WriteTBx’ sends the command cmdSend to the FE5650A using the function ‘FnComm’. The response data is directed to the textbox TBx_RcvdData as English ASCII using the line

   ```
   TBx_RcvdData.Text += System.Text.Encoding.ASCII.GetString(FE5650byte.buf)
   ```

   The encoding can be changed by changing the from ASCII to any of UTF7, UTF8, UTF32, Unicode, or Big Endian Unicode. One returned Chinese characters which, when translated by a Bing Translator, meant ‘Rudder’. Hmmm, not the right encoding. So stick with ASCII. The handler also writes the HEX equivalent of the received data by repeatedly applying the String.Format command to each element of the buffer. The format 0:X2 make sure to write two hex digits per byte and any single digit such as ‘D’ will be written with a zero as ‘0D’.
9. The FnComm function sends the string command and returns the FE5650A response. The function reads the response as individual characters using the Do-Loop. The function breaks out of the loop by virtual of the Try when the read times out at 500mSec as set in the SerialPort1 parameters. If the FE-5650A would include a return at the end of its response or maybe even a 0, then instead of the timeout, one would receive characters until one of those characters is received. Some FE-5650A units encrypt the response so we must wait for the timeout.
10. The three ensuing functions calculate various terms in $Fout=R*Fcode/2^{32}$. Note the use of the largest possible number which is Decimal.
11. The function ‘FnFcodeU32_fromNUDs’ produces a UInt32 unsigned integer (32bits) by scanning NUD tags. Each of the eight NUDs has a tag set to one of the numbers 0 through 7 according to the NUD’s position on the GUI. The right most NUD has tag=0 and the left most one has tag=7. The tag scan is accomplished using a ‘For each’ loop suitable for collections of objects some of

which are NUDs in this case. If the tested object is a NUD, then the tag number is used to calculate the decimal form of Fcode as

$$F_{codeDec} = \sum_{i=0}^{7} Value * 16^{Tag} \qquad \text{(A9.2)}$$

where 'Tag' is the value stored on the NUD tag and Value is the value shown at the window of the spinner.

12. The SubUpdateNUDs subroutine sets the Hex digit on the NUDs. The NUD accepts a byte but shows it as a Hex digit since the related property has been set to true, namely 'NUD.Hexadecimal=true'.
13. The subroutine NUD_ValueChanged is a handler that at minimum converts the hex digits displayed on the NUDs to a frequency shown in the frequency-set textbox TBx_Fset. If the Auto Send button 'CBx_AutoSend' has been clicked then the hex digits are sent to the FE5650A and the response is written to the TBx_RcvdData textbox.
14. The subroutine TBx_Fset_TextChanged is a handler triggered by changing the frequency shown as text. The subroutine calculates an Fcode in Decimal using Equation A9.1 in Section A9.1. The decimal form is then converted to UInt32 which is then used to update the NUDs displayed value using the subroutine SubUpdateNUDs.

Section A9.4 References

[A9.1] The source code, listings, downloads, flash drives, preprogrammed microcontroller:
(A) Chapter 10 and Appendix 6 provide the source code for the C/C++ program
(B) Appendices 7-9 provide the PC software
(C) Check EBay.com in connection with the title of this book: software and preprogrammed MEGA328P
(D) check www.AngstromLogic.com for current offerings, or write to Sales@AngstromLogic.com
Keep in mind that Angstrom Logic, LLC is not setup/able to answer questions on the book or software.
(E) Check Amazon in connection with the title of this book.
(F) Note: The software, programs, microcontroller are for the convenience of the reader and do not come with any kind of warranty/guarantee whatsoever (in the broadest sense of the words) and the contents are copyrighted with all rights reserved. The software/firmware is scanned for viruses before posting. Download only from secure sources and rescan for viruses before using.
(G) The Original Works are provided under this License on an "AS IS" BASIS and WITHOUT WARRANTY, either express or implied, including, without limitation, the warranties of non-infringement, merchantability or fitness for a particular purpose. THE ENTIRE RISK AS TO THE QUALITY OF THE ORIGINAL WORK IS WITH YOU. This DISCLAIMER OF WARRANTY constitutes an essential part of this License. No license to the Original Works is granted by this License except under this disclaimer. Limitation of Liability: Under no circumstances and under no legal theory, whether in tort (including negligence), contract, or otherwise, shall the Licensor be liable to anyone for any indirect, special, incidental, or consequential damages of any character arising as a result of this License or the use of the Original Work including, without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses.