GET STARTED WITH S32K144 USING S32K SDK
基于SDK的S32K144微控制器使用入门

Based on SDK_S32K144_RTM_1.0.0

AGENDA / 目录

- S32K Family Overview
  S32K系列微控制器概述
- S32K144 Architecture and Key Features
  S32K144架构及关键特性
- S32 Design Studio (S32DS) Introduction
  S32 Design Studio开发环境介绍
- S32K SDK Hands-On
  S32K SDK使用操作样例
  - Clocks Lab 时钟配置
  - GPIOs Lab 通用I/O配置
  - Interrupts Lab 中断配置
  - FlexCAN Lab FLexCAN模块配置
  - ADC-FTM Lab ADC-FTM模块配置

S32K FAMILY OVERVIEW
S32K系列MCU概述

S32K – Accelerates Automotive Software Design / 加速汽车软件开发

Select / 选型

Future-proof / 先进的设计

- High performance ARM core
  高性能ARM内核
- Security and Safety
  硬件加密模块和功能安全设计
- Lowest power 低功耗技术
- CAN-FD, Ethernet 以太网

Design / 设计

Minimize Complexity / 复杂度最小化

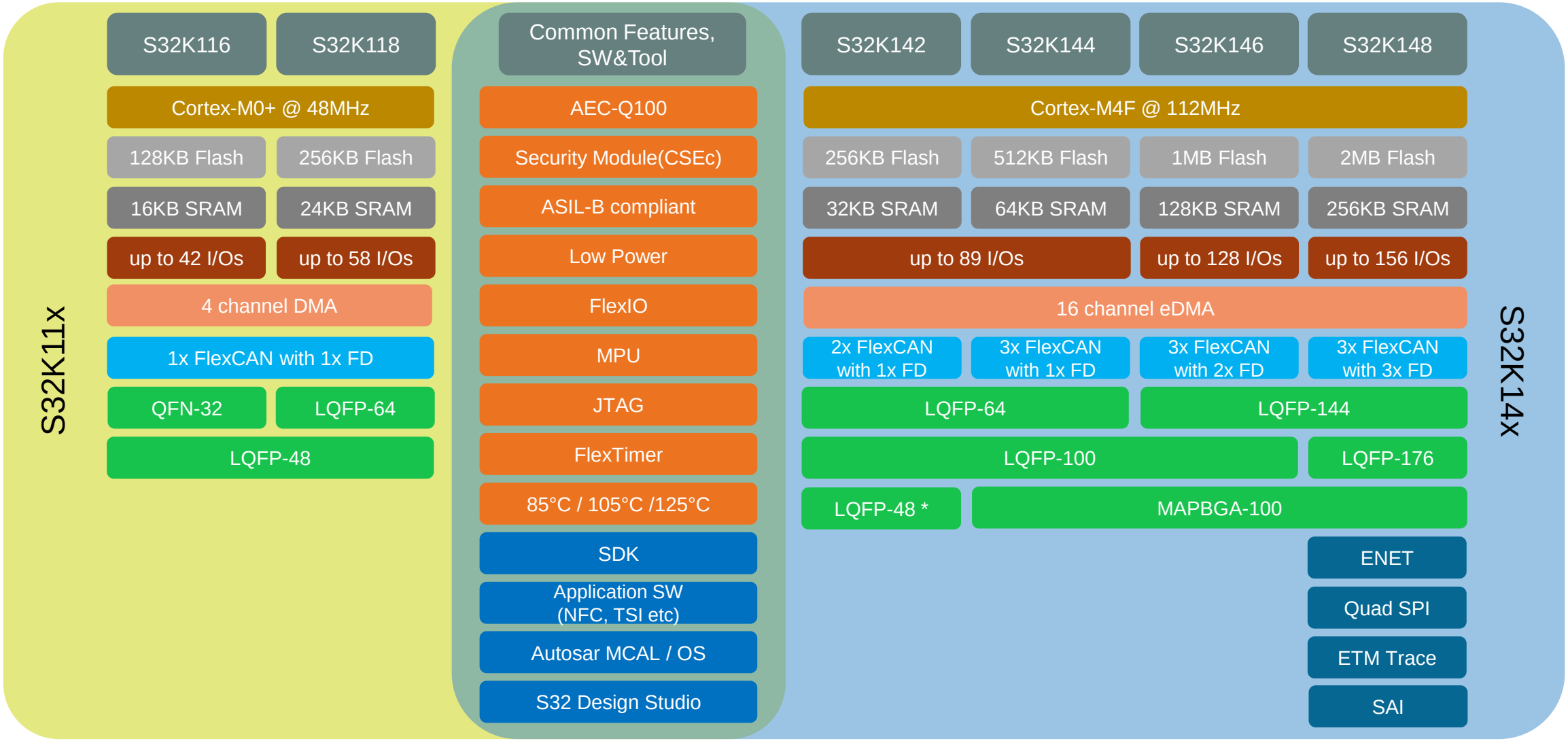
- Auto grade SW: SDK, AUTOSAR
  汽车级软件包：SDK, AUTOSAR
- Low cost HW EVB 低成本开发板
- S32DS and Rich ecosystem
  集成开发环境S32DS，丰富生态系统
- Application specific SW
  提供针对应用的软件

Reuse / 复用

Maximize R&D Efficiency 研发效率最大化

- Most scalable portfolio
  128K-2M flash,32-176pin
  扩展性最好的产品系列
- HW/SW/Tool Compatible
  硬件/软件/工具互相兼容
- Common package strategy
  统一封装

S32K1 product family – S32K14x and S32K11x



*: S32K142 48LQFP is for development only

S32K1 & KEA Product Series Compatibility / 兼容性设计

Pin Compatibility / 管脚兼容

- Within S32K1xx product series
  S32K1系列中管脚完全兼容
- Similar pinout as **KEA** products
  与KEA产品系列管脚设计相似

IP Compatibility / IP兼容

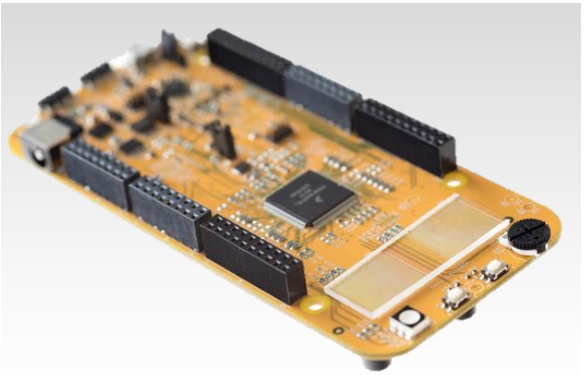
- With MPC55xx/MPC56xxx/MPC57xxx product series: FlexCAN, ACMP, eDMA, QuadSPI
  与MPC55xx/MPC56xxx/MPC57xxx产品系列IP兼容：FlexCAN，ACMP，eDMA，QuadSPI
- With KEA products: FlexTimer, IIC, LSPI, UART, CRC, FlexIO
  与KEA产品系列IP兼容：FlexTimer，IIC，LSPI，UART，CRC，FlexIO

Flash	Pin Count								
	16/24	32	48	64	80	100	100 BGA	144	176
2M							S32K148	S32K148	S32K148
1M						S32K146	S32K146	S32K146	
512K				S32K144		S32K144	S32K144		
256K			S32K142 * S32K118	S32K142 / S32K118		S32K142			
128K		S32K116	S32K116	KEAZ128	KEA128				
64K		KEAZN64		KEAZ(N)64	KEAZ64				
32K		KEAZN32 /		KEAZN32					
16K		KEAZN16		KEAZN16					
8K	KEAZN8								

***: S32K142 48LQFP is for development only**

S32K Solution Offering / S32K提供的解决方案

Hardware Platform 硬件平台



- Low cost development board compatible to Arduino shields
  低成本开发板，兼容Arduino扩展板
- Onboard debugger and system basis chip
  板上集成调试器（OpenSDA）和SBC芯片

Full Hardware evaluation and Development Platforms
完整的硬件评估和开发平台

Runtime Software 运行时软件

AUTOSAR

S32 SOFTWARE DEVELOPMENT KIT
APPLICATION DEVELOPMENT SOFTWARE
NXP

freeRTOS

- Auto Grade NXP Software Development Kit (SDK)
  NXP提供汽车级软件开发包
- NXP Middleware e.g. Core Self Test, LIN Stack
  NXP提供中间件（内核自检，LIN协议栈等）
- Autosar 4.0 and 4.2 MCAL
- FreeRTOS
- Bootloader

Production grade Software for Faster time to market
产品级软件包，加快产品上市速度

Software Dev Tools 软件开发工具

IAR SYSTEMS

S32 DESIGN STUDIO
EMBEDDED SOFTWARE DEVELOPMENT TOOLS
NXP

Green Hills SOFTWARE

ARM

LAUTERBACH DEVELOPMENT TOOLS

- IAR, GHS and GNU toolchains
  支持IAR，GHS和GNU工具链
- Full-featured, no cost development platform (S32 DS)
  全功能一体化免费的开发平台
- FreeMASTER
  图形化上位机

Complete tools package to streamline software development
完整工具包，简化软件开发过程

Application Specific 应用样例



- Motor Control
  电机控制
- Touch Sensing
  触摸传感
- Secure Communication
  加密通信
- Wireless Charging
  无线充电
- Near Field Communication
  近场通信

Application Specific Middleware
特定应用的中间件

AUTOSAR MCAL and Middleware is sold separately
AUTOSAR MCAL、中间件软件需单独购买

S32K144 ARCHITECTURE & KEY FEATURES

S32K144架构及关键特性

S32K – Future Proof / 先进的设计

Superior Performance / 出众的性能

- High speed ARM Cortex-M4F CPU with DSP functionality
  具备DSP功能的高速ARM Cortex-M4F CPU
- IEEE-754 HW floating point unit without SW overhead
  IEEE-754标准硬件浮点运算单元，无需软件开销
- Harvard architecture accelerates data handling
  哈佛架构，提高数据处理能力
- 16 bit instruction set (THUMB 2) → ~31% reduced memory usage
  16位长度指令集（Thumb-2）使用少于31%的内存
- Combined D/I cache for direct execution
  组合的数据/指令缓存用于直接存取
- Concurrent, low latency bus accesses over crossbar
  通过交叉开关实现低时延的总线同时访问
- Parallel DMA operation
  DMA并行操作
- Dedicated FlexNVM to support read while write
  专属FlexNVM，支持边读边写
- 100Mbit/s Ethernet 100Mbit/s以太网
- IEEE 1588 Time Stamping 支持IEEE 1588标准时间戳

Highest Energy Efficiency / 高能耗比

- Low leakage technology (C90TFS) 低漏电流技术
- Multiple low power modes 多种低功耗模式
- Internal oscillators e.g. 48MHz 1.3% 内部振荡电路
- Best in class STOP current: 25-40uA (device depended)
  STOP模式下的电流：25-40微安

CANFD, Safety, Security / CANFD、功能安全、加密引擎

- CAN with Flexible Datarate (FD) option according to ISO/CD 11898-1
  符合ISO/CD 11898-1的CAN-FD
- HW motor control support (BLDC/PMSM)
  支持电机控制硬件模块（BLDC/PMSM）
- ISO26262 compliance (ASIL-B) 符合ISO26262 ASIL-B设计
- Communication protocol emulation module (FlexIO)
  通讯协议硬件模拟模块（FlexIO）
- HW security engine (SHE+ compliant) 硬件加密引擎（符合SHE+规范）

S32K144 Block Diagram / S32K144架构框图

High performance / 高性能

- ARM Cortex M4F up to 112MHz w FPU
  ARM Cortex M4F，最高支持112MHz，具备单精度浮点运算单元
- eDMA from 57xxx family
  从MPC57xxx系列移植的eDMA模块

Software Friendly Architecture / 友好的软件开发架构

- High RAM to Flash ratio 高RAM/Flash比率
- Independent CPU and peripheral clocking CPU及外设的独立时钟配置
- 48MHz 1% IRC 内部RC震荡器
- Registers maintained in all modes 所有功耗模式下寄存器内容均保持
- Programmable triggers for ADC - no SW delay counters or extra interrupts
  可配置的ADC触发 – 无需软件计数延迟或额外中断

Functional safety / 功能安全

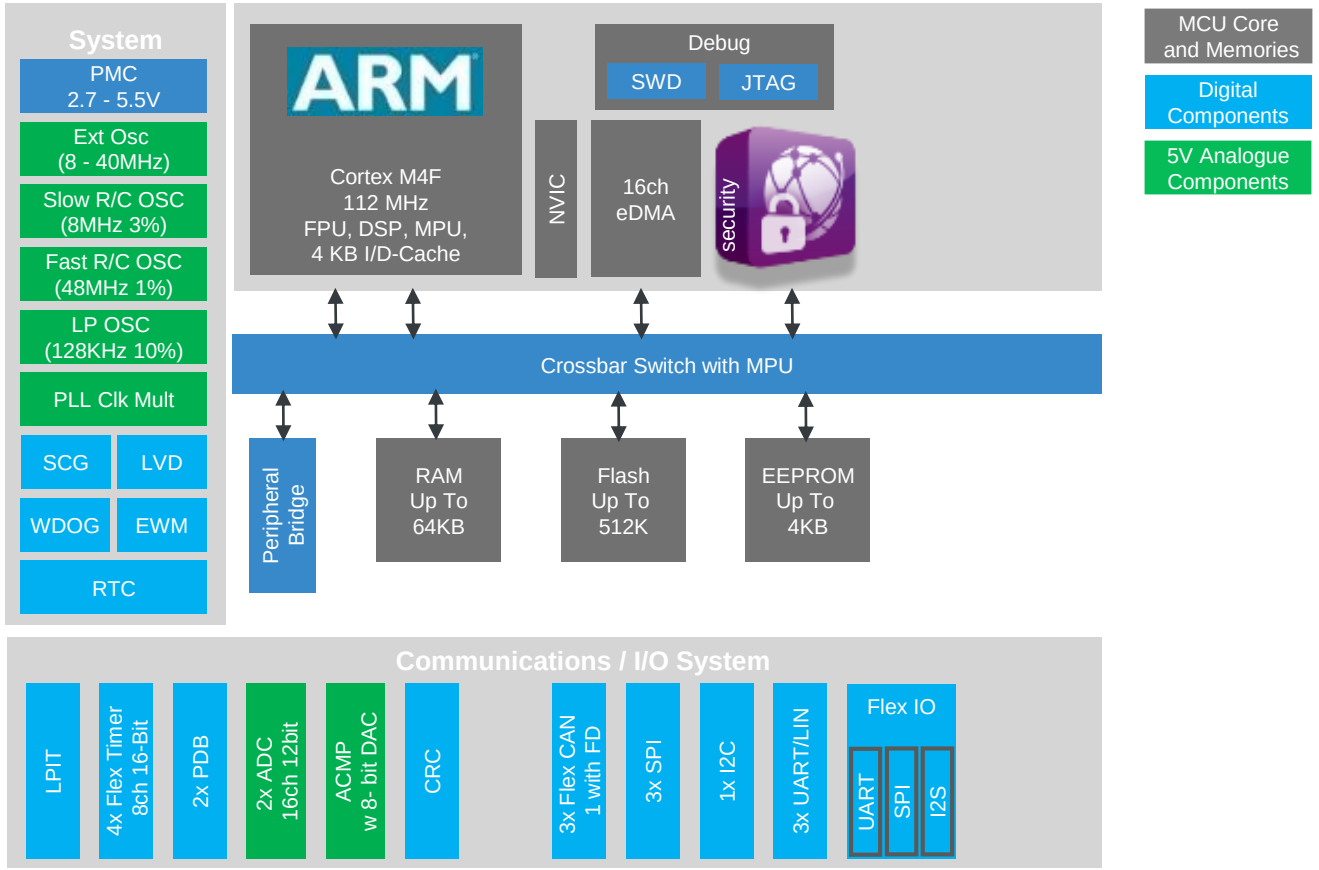
- ISO26262 support for ASIL B or higher 支持ISO26262 ASIL-B或更高设计
- Memory Protection Unit, ECC on Flash/Dataflash and RAM
  内存保护单元，支持Flash，DataFlash和RAM的ECC校验
- Independent internal OSC for Watchdog
  用于看门狗的内部独立OSC
- Diversity between ADC and ACMP, SPI/SCI and FlexIO
  冗余多样性设计（ADC和ACMP，SPI、SCI和FlexIO）
- Core self test libraries 内核自检库
- Scalable LVD protection, CRC 低电压保护，CRC校验

Low power / 低功耗

- Low leakage technology 低漏电流技术
- Multiple VLP modes and IRC combos 多种低功耗模式和IRC组合方式
- Wake-up on analog thresholds 模拟阈值输入唤醒

Security / 加密

- CSEc (SHE-spec) 硬件加密引擎（符合SHE规范）



Operating Characteristics

- Voltage range: 2.7V to 5.5V
- Temperature (ambient): -40°C to +125°C

Packages & IO

- Open-drain for 3.3 V and hi-drive pins
- Powered ESD protection
- Packages: 100 BGA, 64 LQFP, 100 LQFP

ASIL-B Functional Safety / 满足ASIL-B的功能安全设计

Safety Hardware 安全的硬件设计

- Power supplies 电源供电
- Clocks generation 时钟产生器
- Core platform (core, DMA, cache …) 内核系统
- Busses – XBAR 内部互联总线系统
- Memories – NVM, SRAM 内存系统

Safety Process 功能安全的设计流程

- ISO 26262 development process

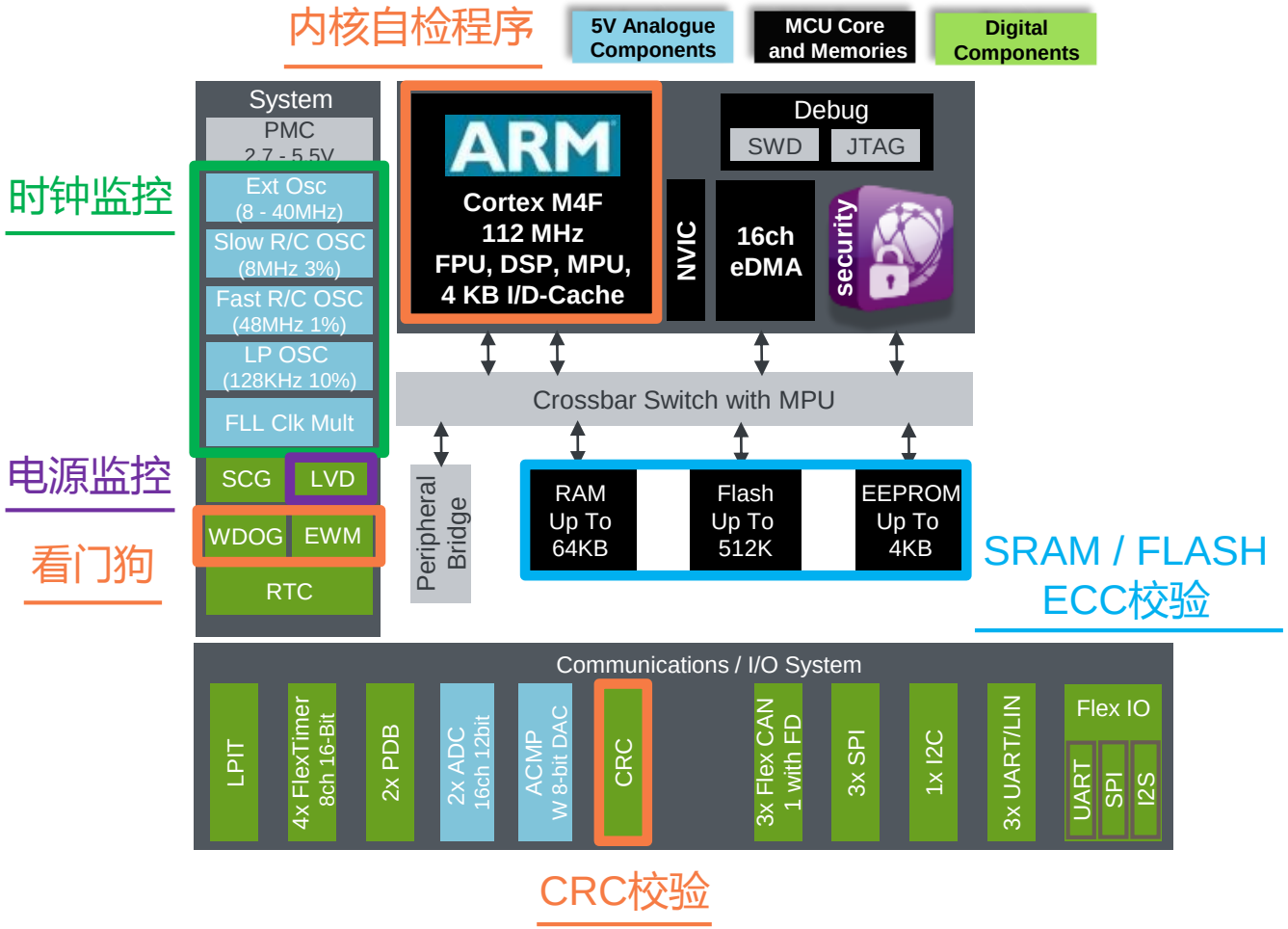
  ISO 26262设计流程

Safety Support 功能安全支持

- FMEDA 失效模式及诊断分析
- Safety manual 安全手册
- Technical support 技术支持

Safety Software 功能安全软件

- S32K core self-test SW S32K内核自检软件
- PT lib 外设检测库



S32K Security – CSEc / CSEc安全加密模块

- Supports all Global OEM Requirements for End Node Security

  支持国际车厂对于边缘节点安全加密的需求

Supports >SHE functionality

满足高于SHE安全加密标准的要求

- Secure key storage

  密钥存储

- AES-128 encryption/decryption

  AES-128加密/解密算法

- AES-128 Cypher-based Message Authentication Code (CMAC) calculation and authentication

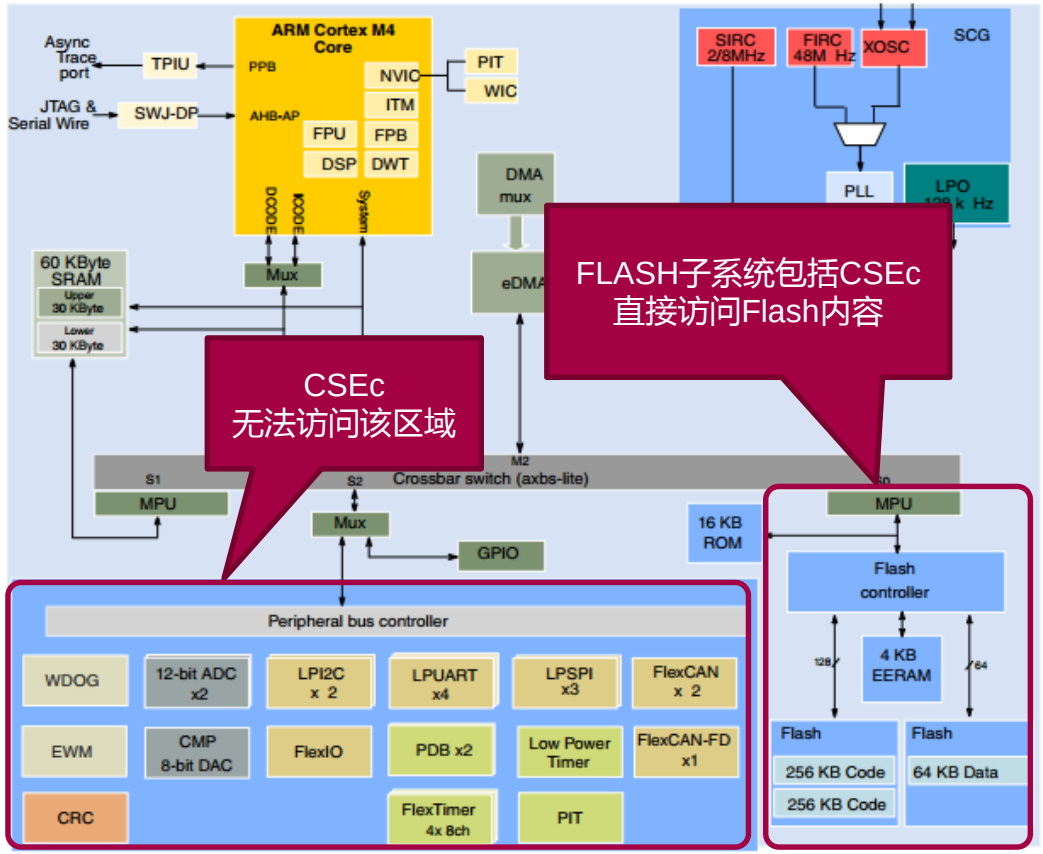
  基于AES-128加密算法的信息认证码（CMAC）的计算和认证

- True and Pseudo random number generation

  真随机数和伪随机数产生器

- User configurable Secure Boot Mode (Sequential, Strict, or Parallel Boot)

  用户可配置的安全启动模式（顺序启动，严格启动，并行启动）



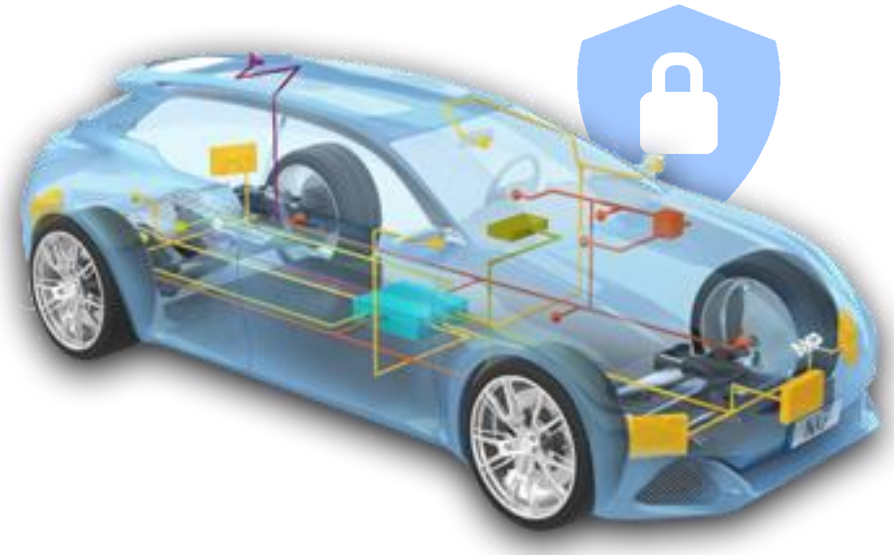
S32K Security – Use Case / S32K安全加密使用案例

In-Vehicle Security 车内安全

- Immobilizer / Component Protection - Detection of replacement or modification
  防盗/零部件保护 – 检测零部件替换或更改
- Mileage Protection - Disabling manipulation
  里程保护 – 防篡改
- Secure Boot and Chain of Trust - Verification of authenticity and integrity of application software
  安全启动和信任链 – 正版认证和应用程序完整度认证
- Secure Communication - Protection of the network from unauthorized access
  安全的通信 – 防止车内网络遭受未授权访问
- DRM - Renault Zoe use DRM in batteries
- 数字版权管理（DRM）- 雷诺Zoe在电池中使用DRM

Connected Vehicle Security 车联安全

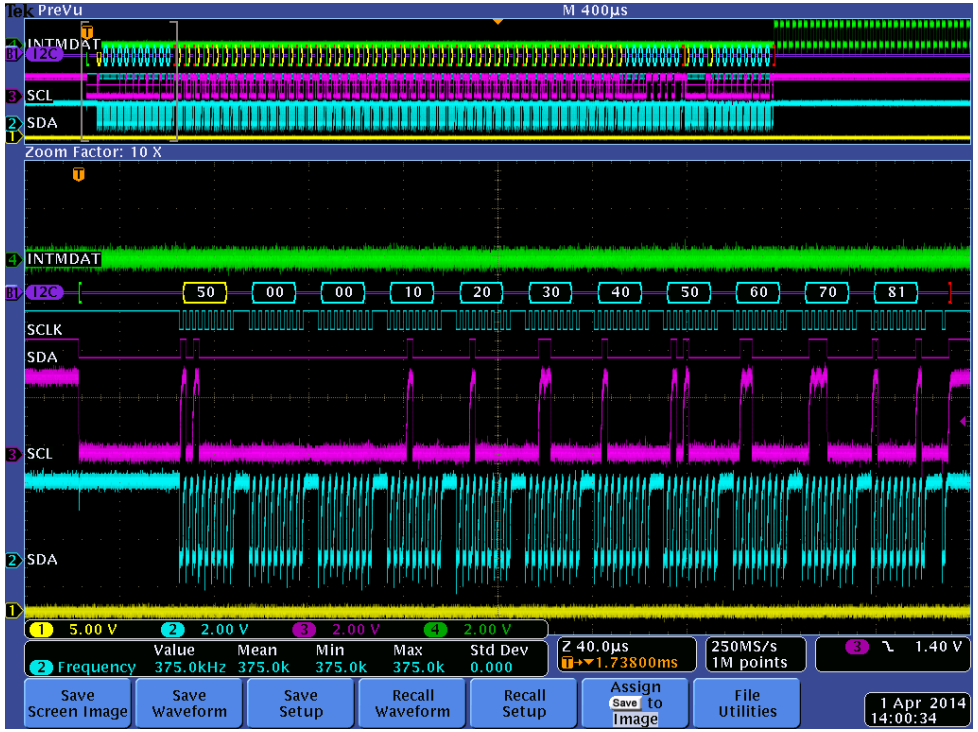
- Android application download
  安卓应用下载
- DRM for content download/streaming
  DRM方式的内容下载/数据流
- Remote ECU firmware update
  ECU固件远程更新
- Black-box for due government or insurance
  政府和保险用的黑盒子



S32K – FlexIO Peripheral Capabilities / 易扩展I/O外设

Flexible input and output peripheral (FlexIO) 易扩展的输入输出外设

- Programmable logic for complex output waveform generation
  可配置的复杂输出波形生成器
- Emulation of standard communication interfaces
  能够模拟多种标准通信接口
  - UART, SPI, I2C, I2S, LCD RGB, PWM, SENT, etc.
- Low CPU overhead
  CPU内核开销低
- DMA support
  支持DMA
- SDK based drivers available
  基于SDK的底层驱动已发布

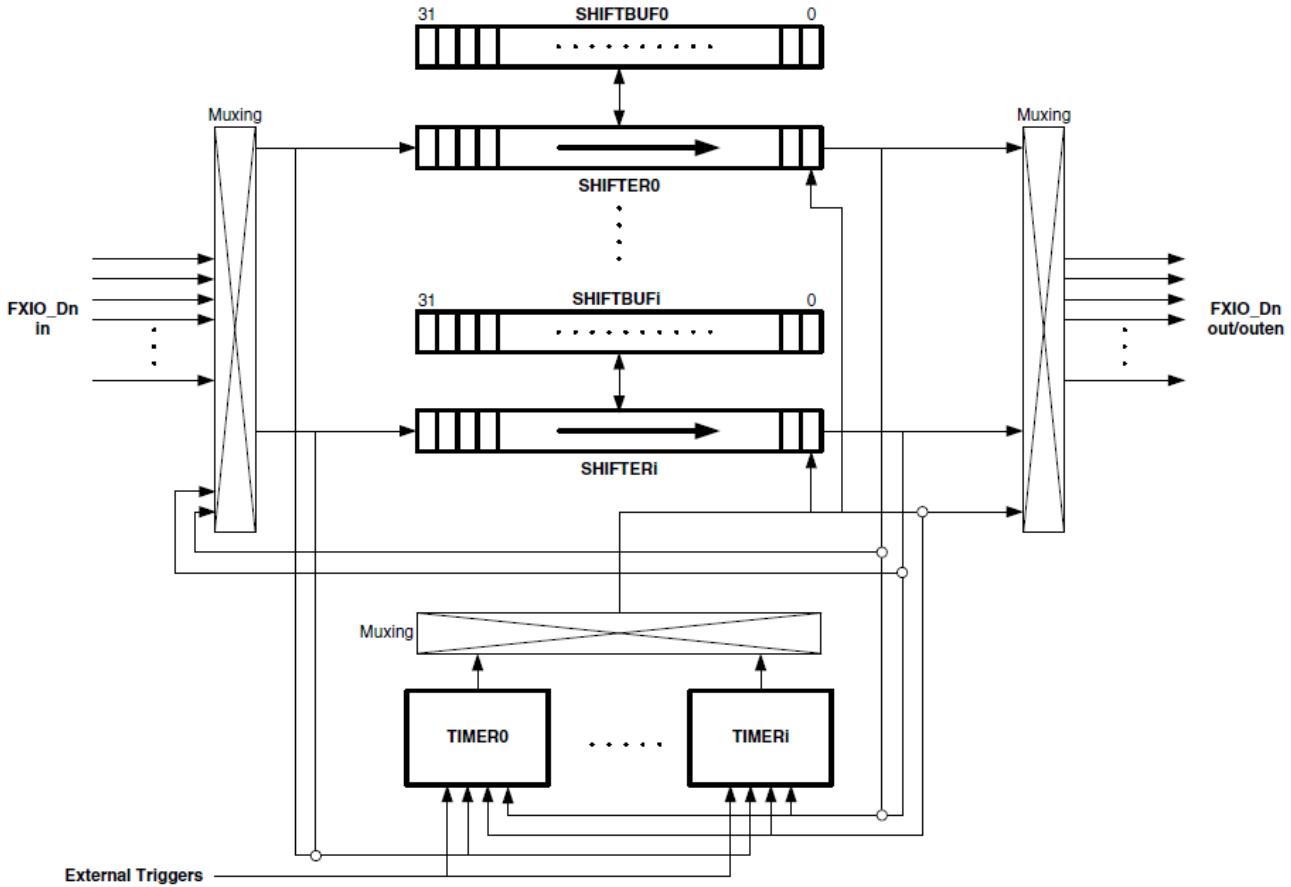


Emulation of I2C / 模拟出的I2C波形

FlexIO – Block Diagram / FlexIO模块框图

- FlexIO main components
  FlexIO主要结构
  - Shifters 移位器x4
  - Timers 定时器x4
  - Muxing interfaces 多路复用接口（8xPin）
- FlexIO: I/O Operation
  输入输出操作
  - Each timer and shifter can be configured to use any FlexIO pin as

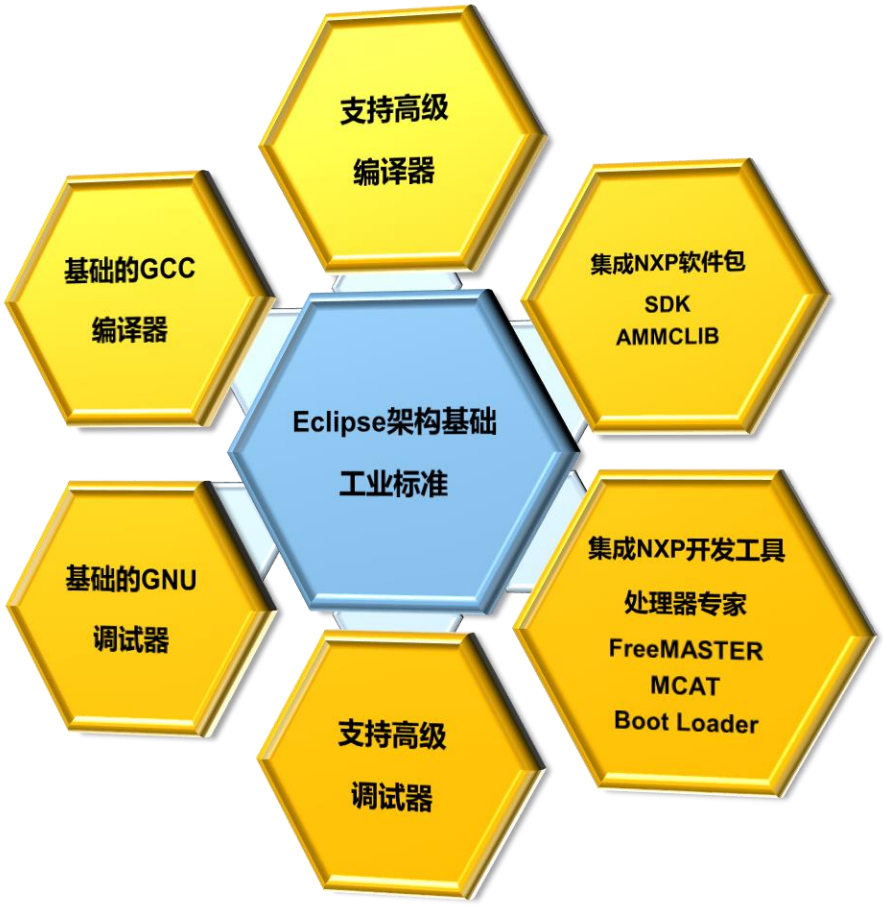
    通过对定时器和移位器的操作，可将任意FlexIO管脚配置成以下模式
    - Input 输入
    - Output data 数据输出
    - Output enable (Open Drain) 输出使能（开漏）
    - Bidirectional output 双向输出



S32 DESIGN STUDIO INTRODUCTION

S32DS开发环境介绍

S32 Design Studio – 针对汽车微控制器的集成开发环境



NXP Software Integrated into the tool as part of shipping package. When customer creates a new project he can include NXP software as part of project creation, no more user needing to search for and integrate NXP software.

NXP 软件包已集成在S32DS开发环境中，用户新建工程时可直接勾选需要的软件包，无需额外手动集成。

Automotive Math And Motor Control Libraries
数学算法和电机控制库

- Libraries included automatically
  库文件自动包含在工程中
- User just needs to drag functions into source files to utilize
  用户只需拖拽函数来实现功能

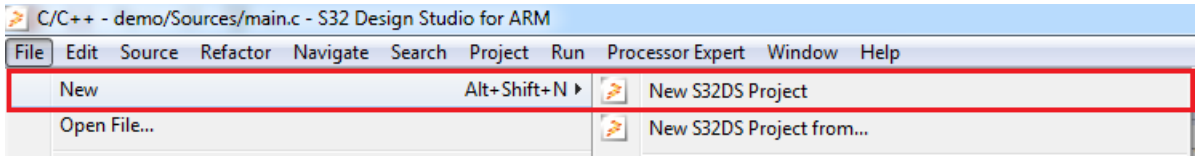
Software SDK functions available in the environment
开发环境中集成软件SDK

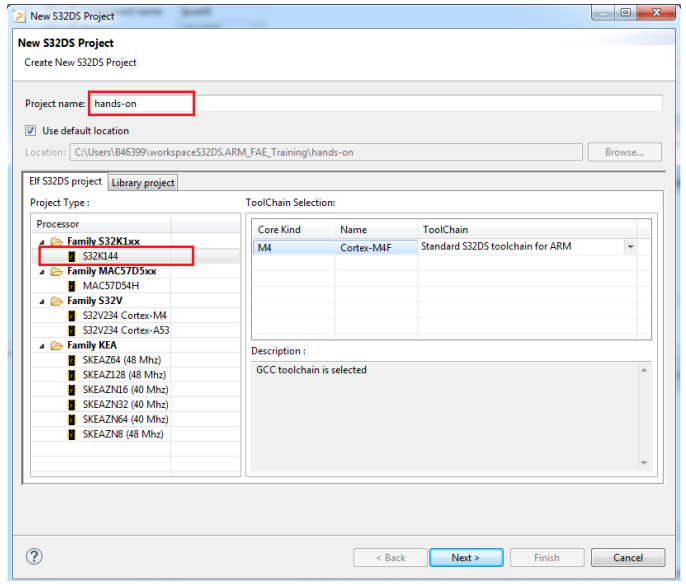
- Bare Metal Drivers available directly in tool for ease of use
  完整的底层驱动，简化使用
- Drag and drop into source to utilize
  拖拽函数实现功能

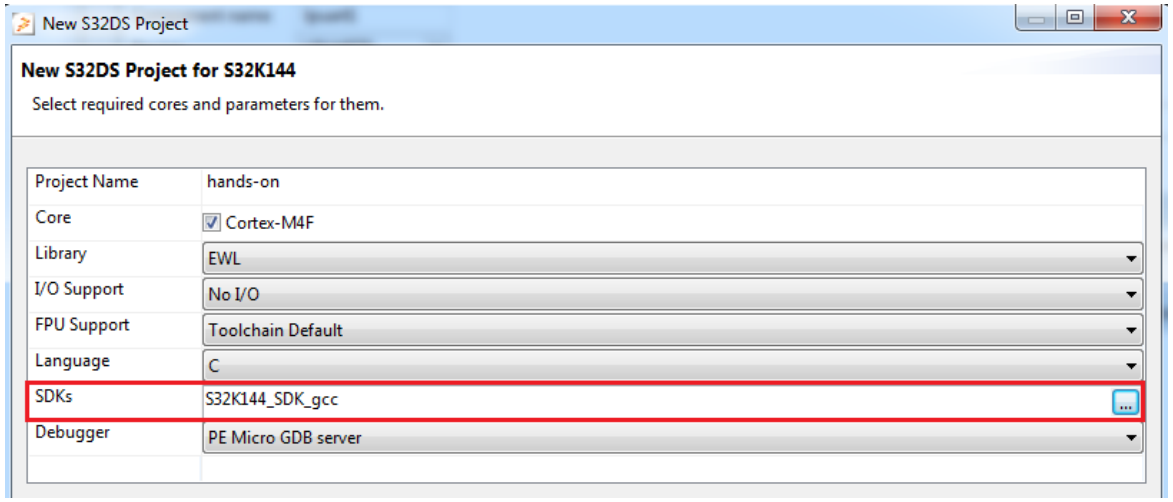
S32DS – How to create a project / 如何新建工程

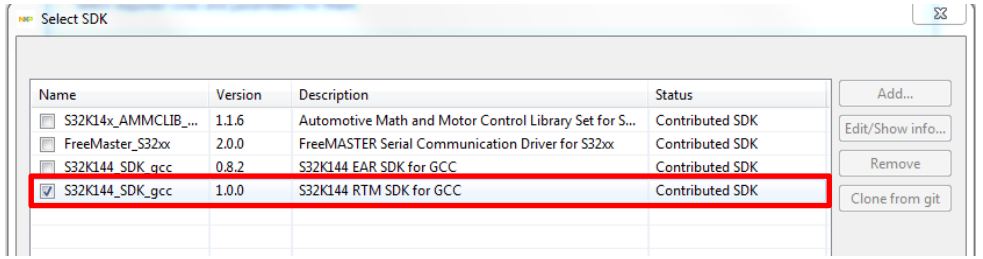
- Create a new S32DS Project

  新建S32DS工程

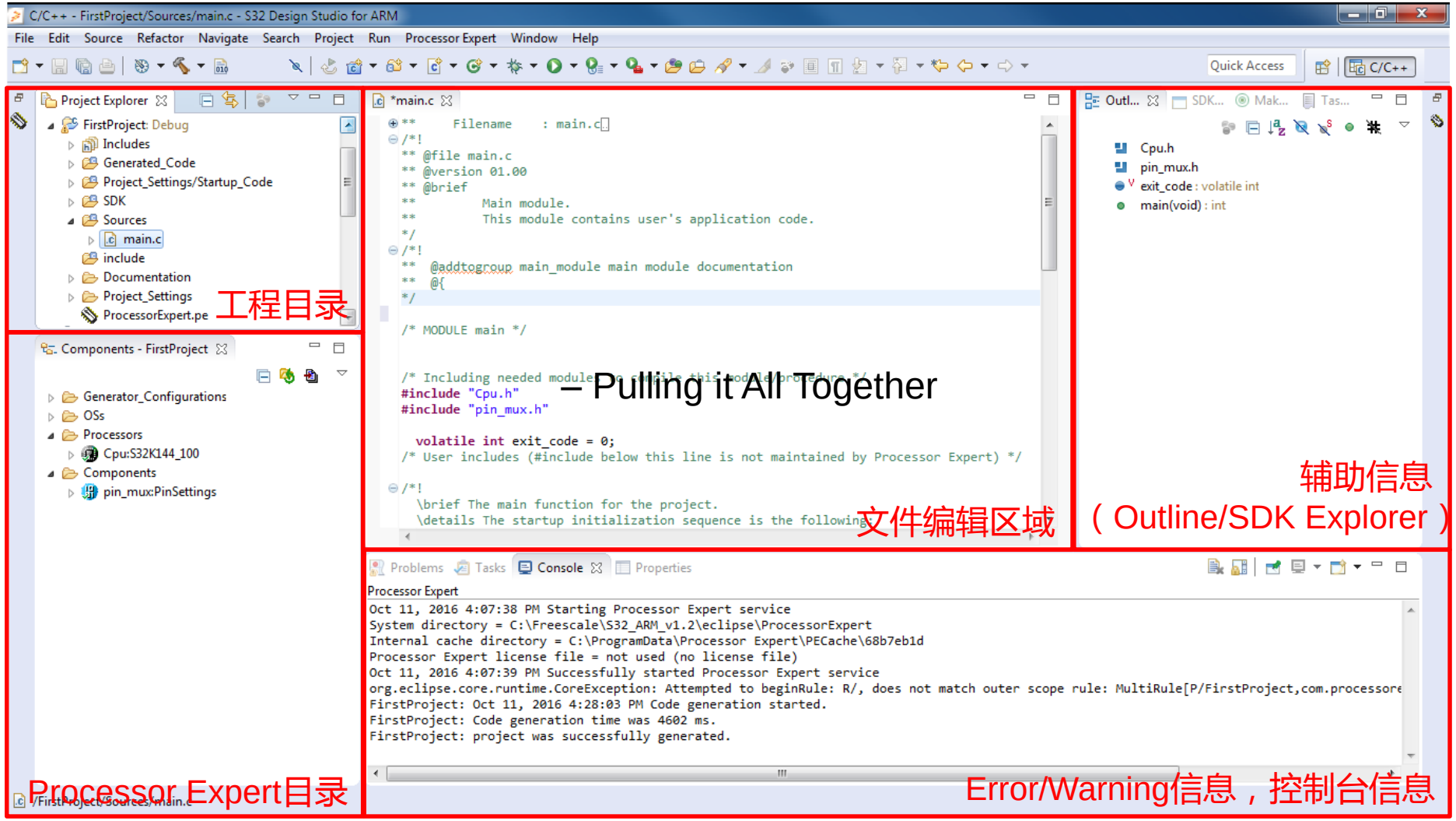




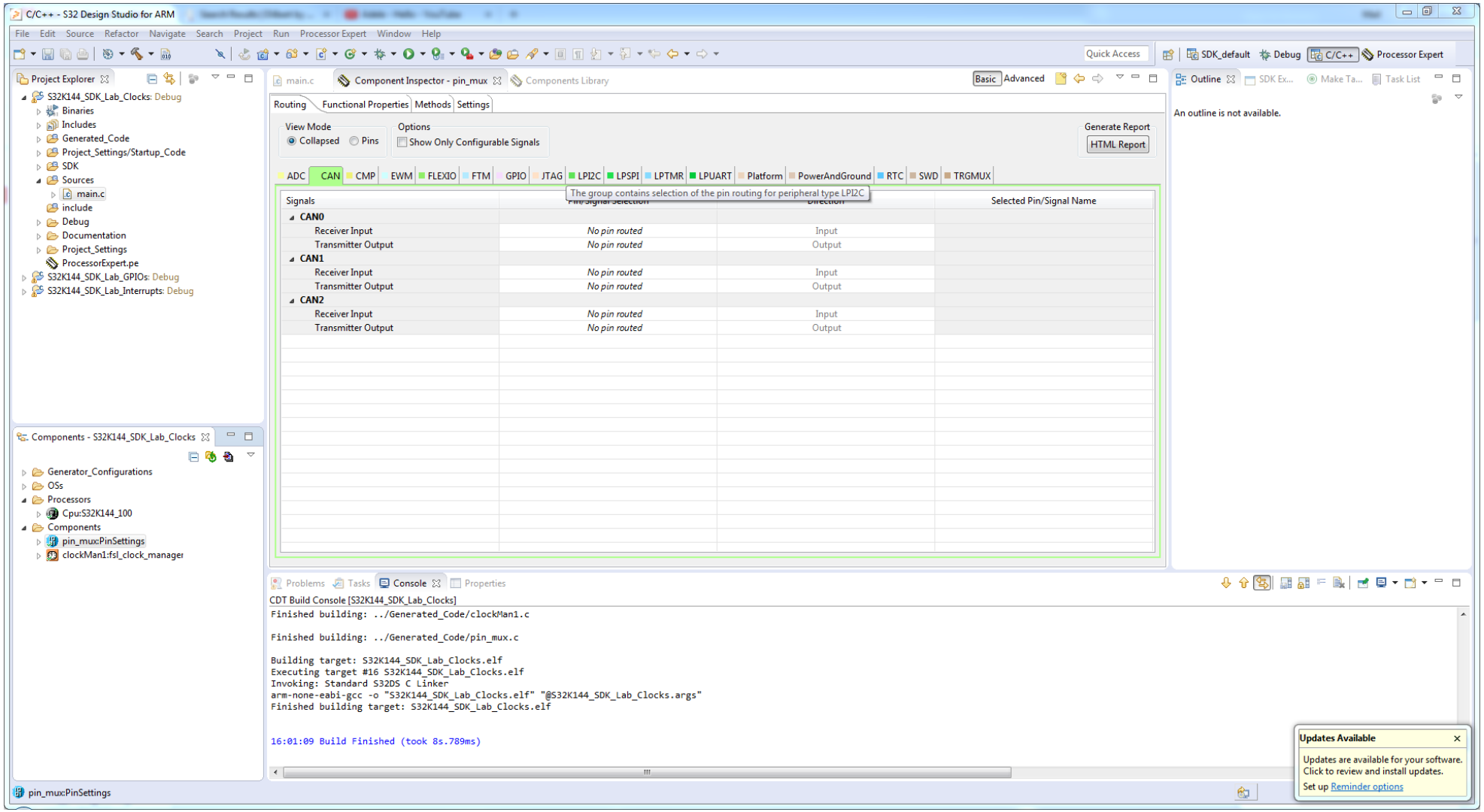




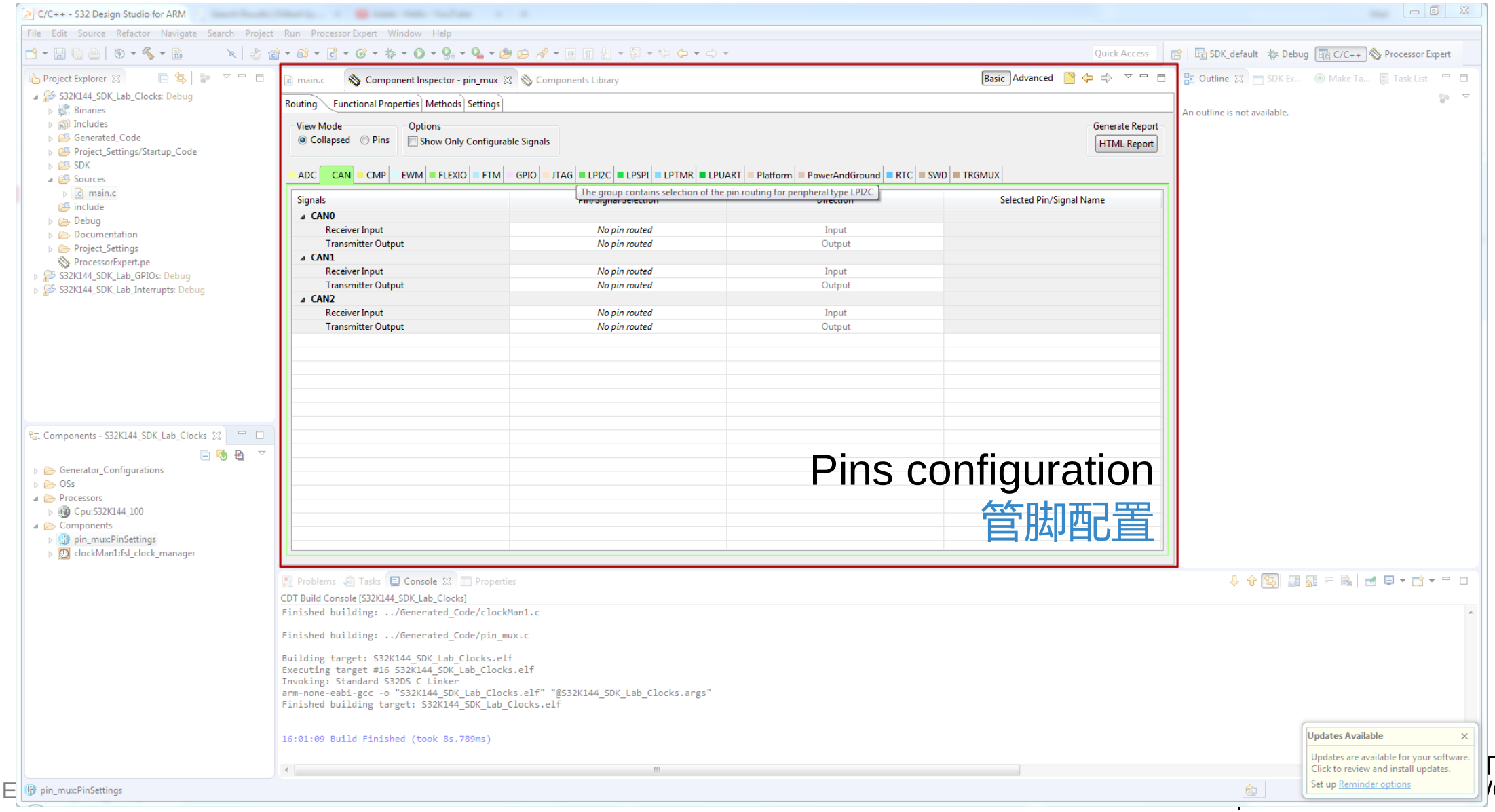
S32 Design Studio



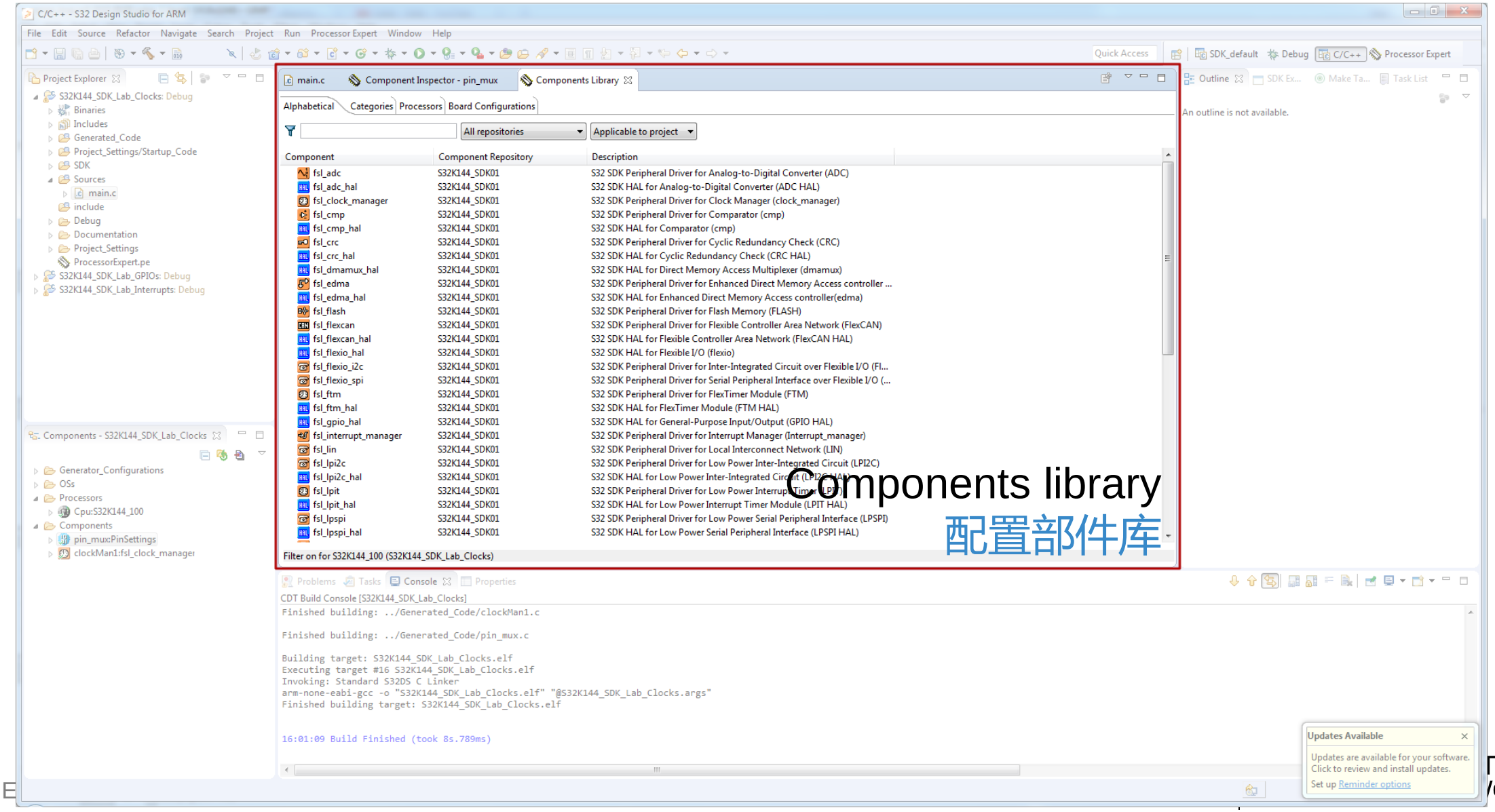
S32DS – Graphical configuration environment / 可视化配置环境



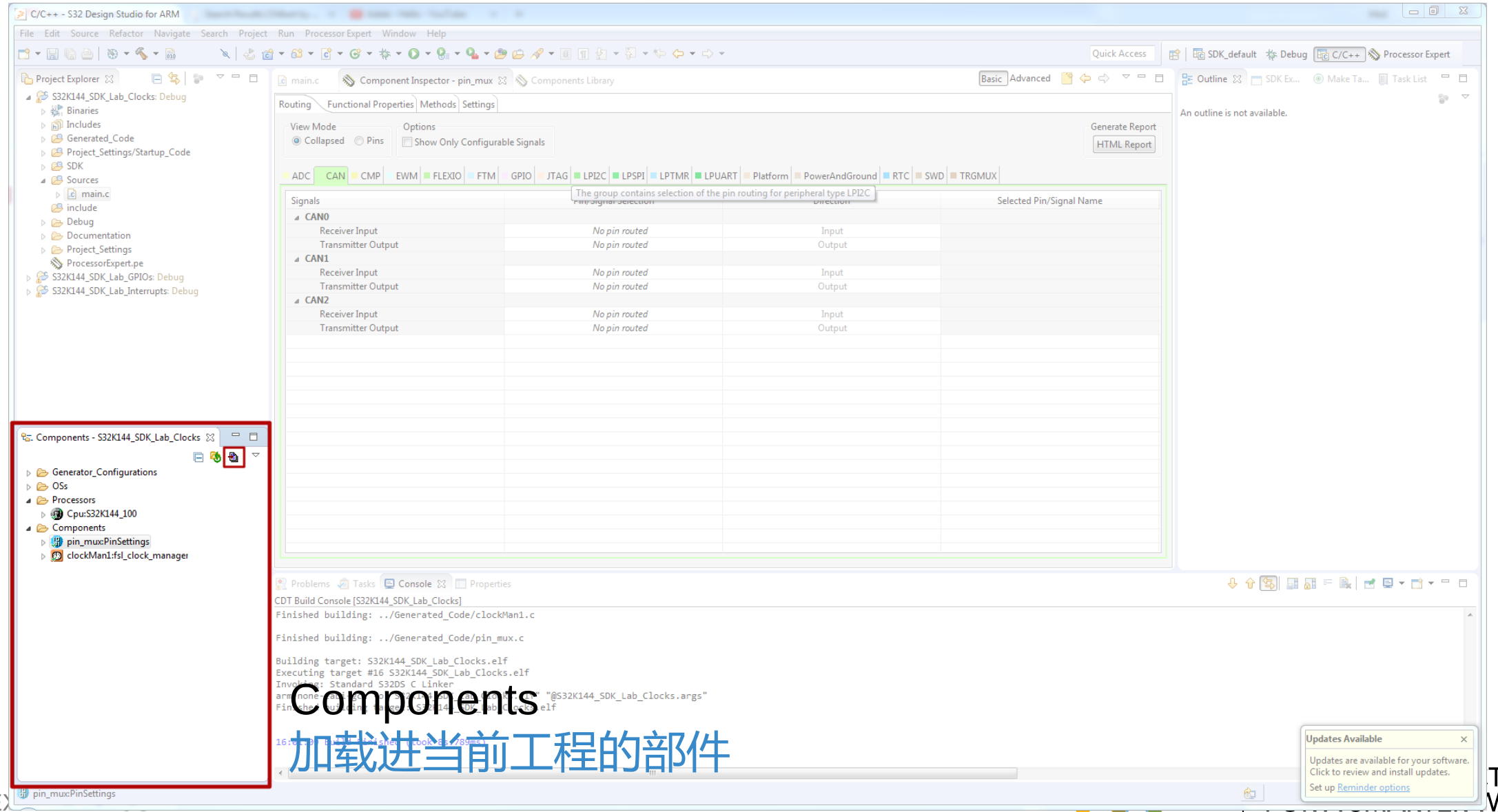
S32DS – Graphical configuration environment / 可视化配置环境



S32DS – Graphical configuration environment / 可视化配置环境



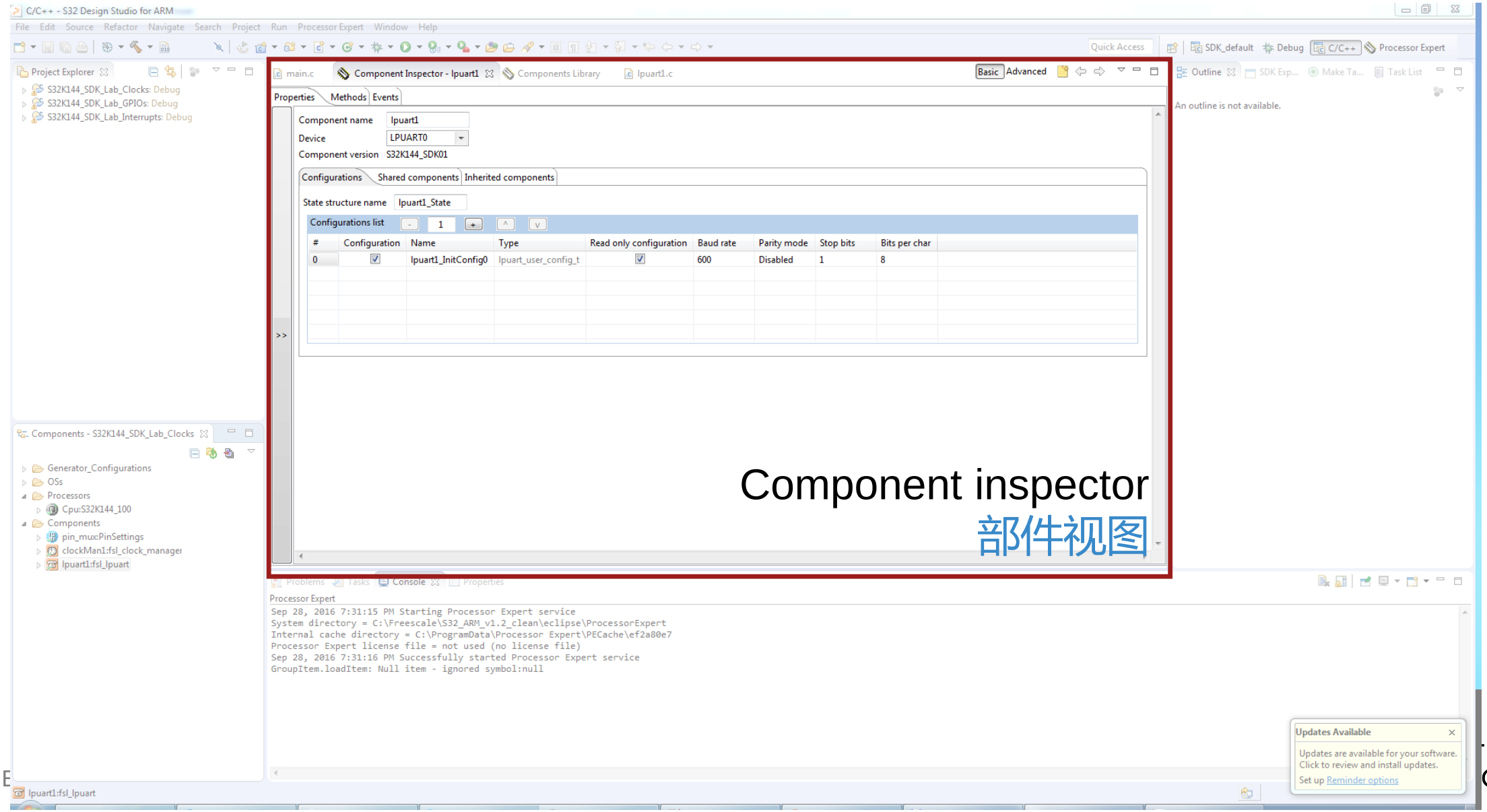
S32DS – Graphical configuration environment / 可视化配置环境



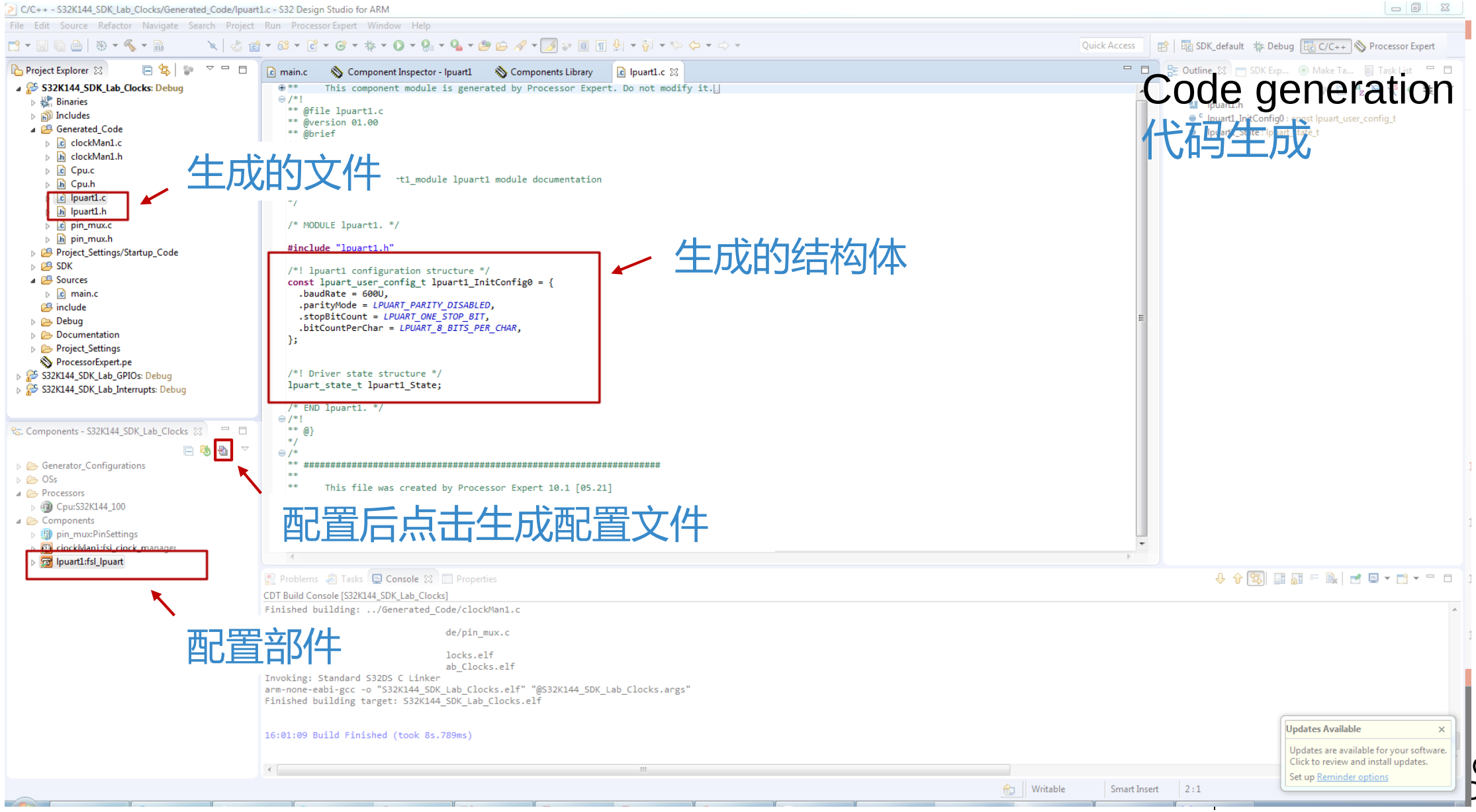
Components

加载进当前工程的部件

S32DS – Graphical configuration environment / 可视化配置环境



S32DS – Graphical configuration environment / 可视化配置环境



S32 Design Studio – Deploying the Application / 配置映射

- Initialize components

  将生成的结构体在主函数中实现

Clock Initialization and Configuration

Pin Initialization

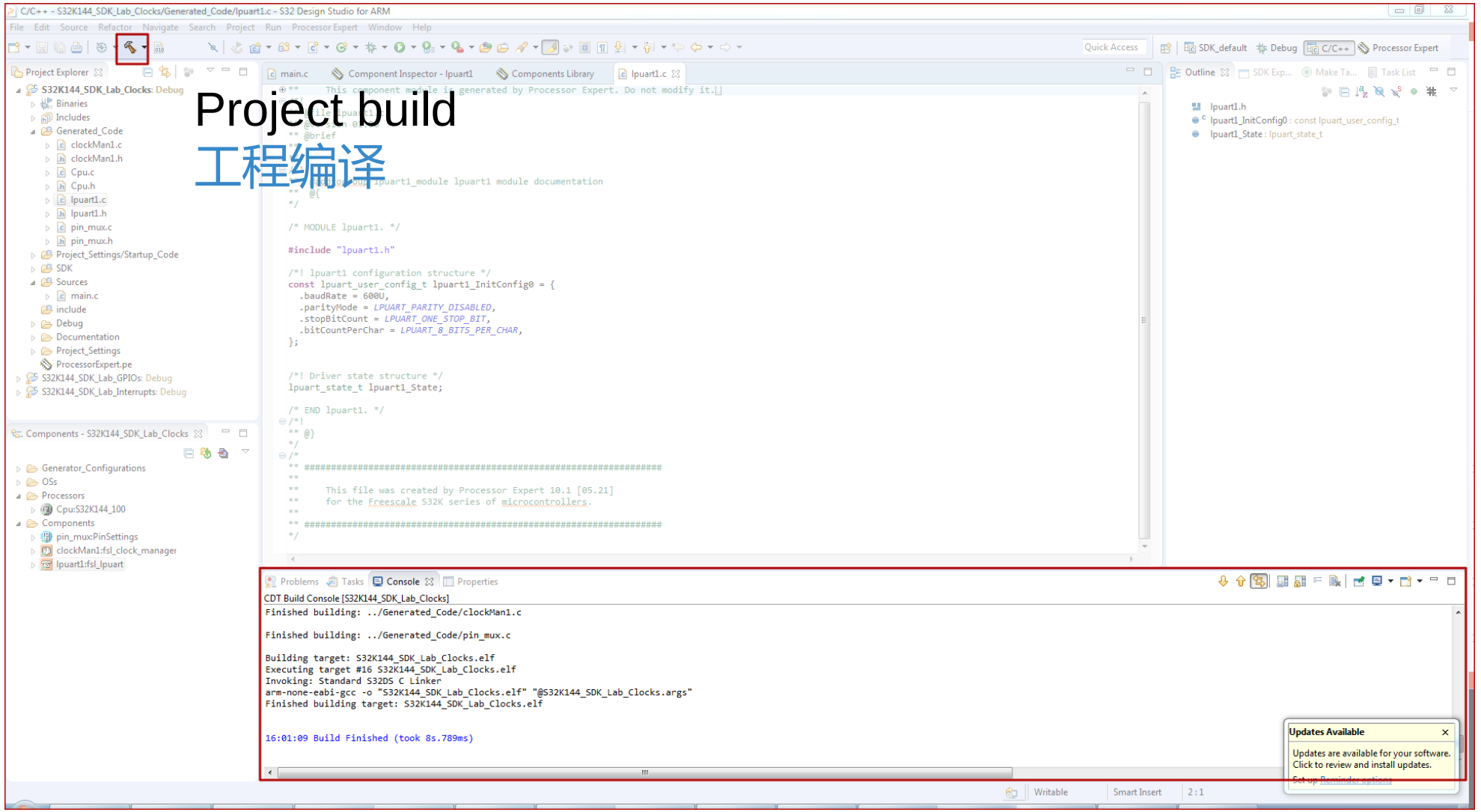
LPUART Initialization

While (1) { … }

main.c

```
/* Write your code here */
/* For example: for(;;) { } */

/* Initialize and configure clocks
 *     -   see clock manager component for details
 */
CLOCK_SYS_Init(g_clockManConfigsArr, FSL_CLOCK_MANAGER_CONFIG_CNT,
               g_clockManCallbacksArr, FSL_CLOCK_MANAGER_CALLBACK_CNT);
CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);

/* Initialize pins
 *     -   See PinSettings component for more info
 */
Pins_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

/* Initialize LPUART instance */
LPUART_DRV_Init(FSL_LPUART1, &lpuart1_State, &lpuart1_InitConfig0);
/* Send a welcome message */
LPUART_DRV_SendData(FSL_LPUART1, (uint8_t *)welcomeMsg, strlen(welcomeMsg));
/* Wait for transmission to be complete */
while(LPUART_DRV_GetTransmitStatus(FSL_LPUART1, NULL) != LPUART_STAT_SUCCESS);

/* Infinite loop:
 *     - Receive data from user
 *     - Echo the received data back
 */
while (1)
  {
    /* Get the received data */
    while(strReceived == false)
```

S32DS – Graphical configuration environment / 可视化配置环境



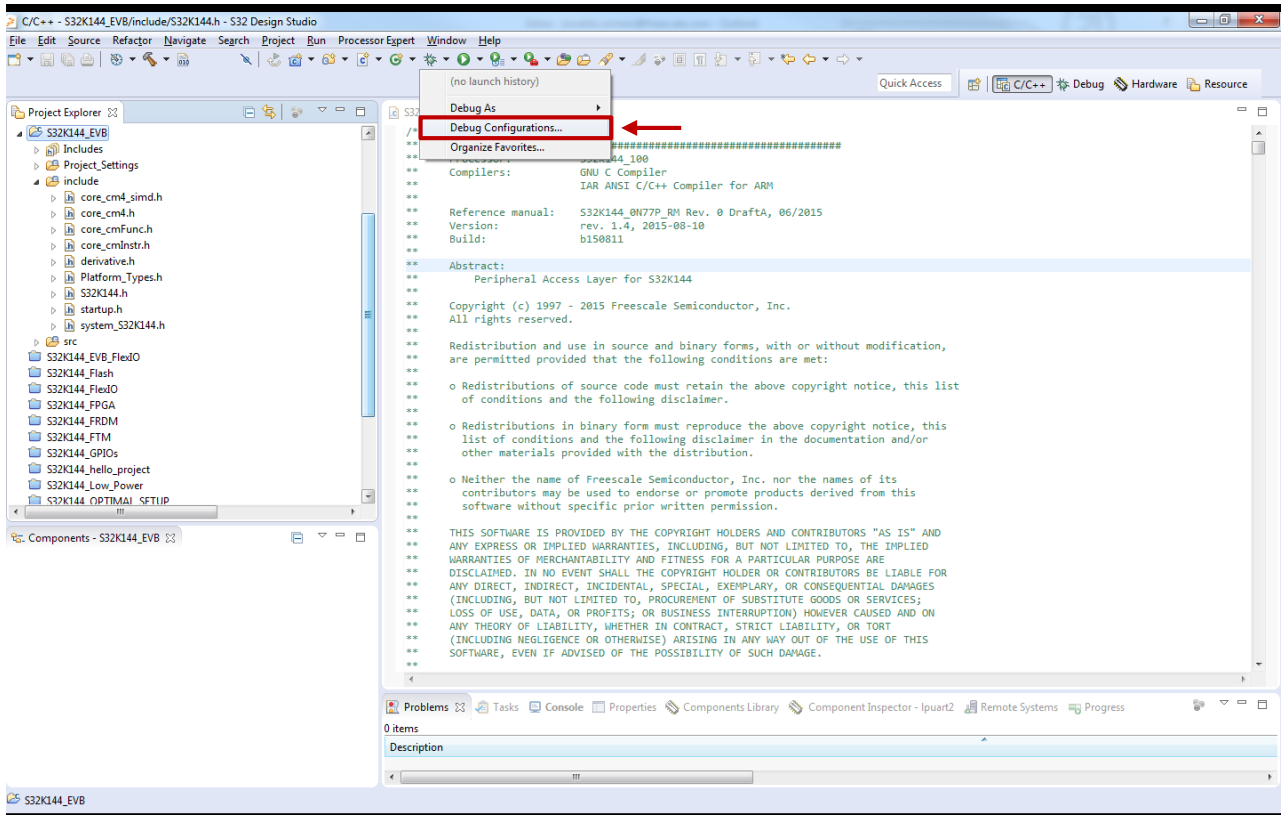
S32DS – OpenSDA Debug Configuration / OpenSDA调试配置

- To Debug your project with OpenSDA, it is necessary to select the OpenSDA in the Debug Configuration.

  使用OpenSDA调试工程，需要在调试配置界面中选择OpenSDA

- Select your project, and click on debug configuration

  选择对应工程，点击Debug Configuration按钮



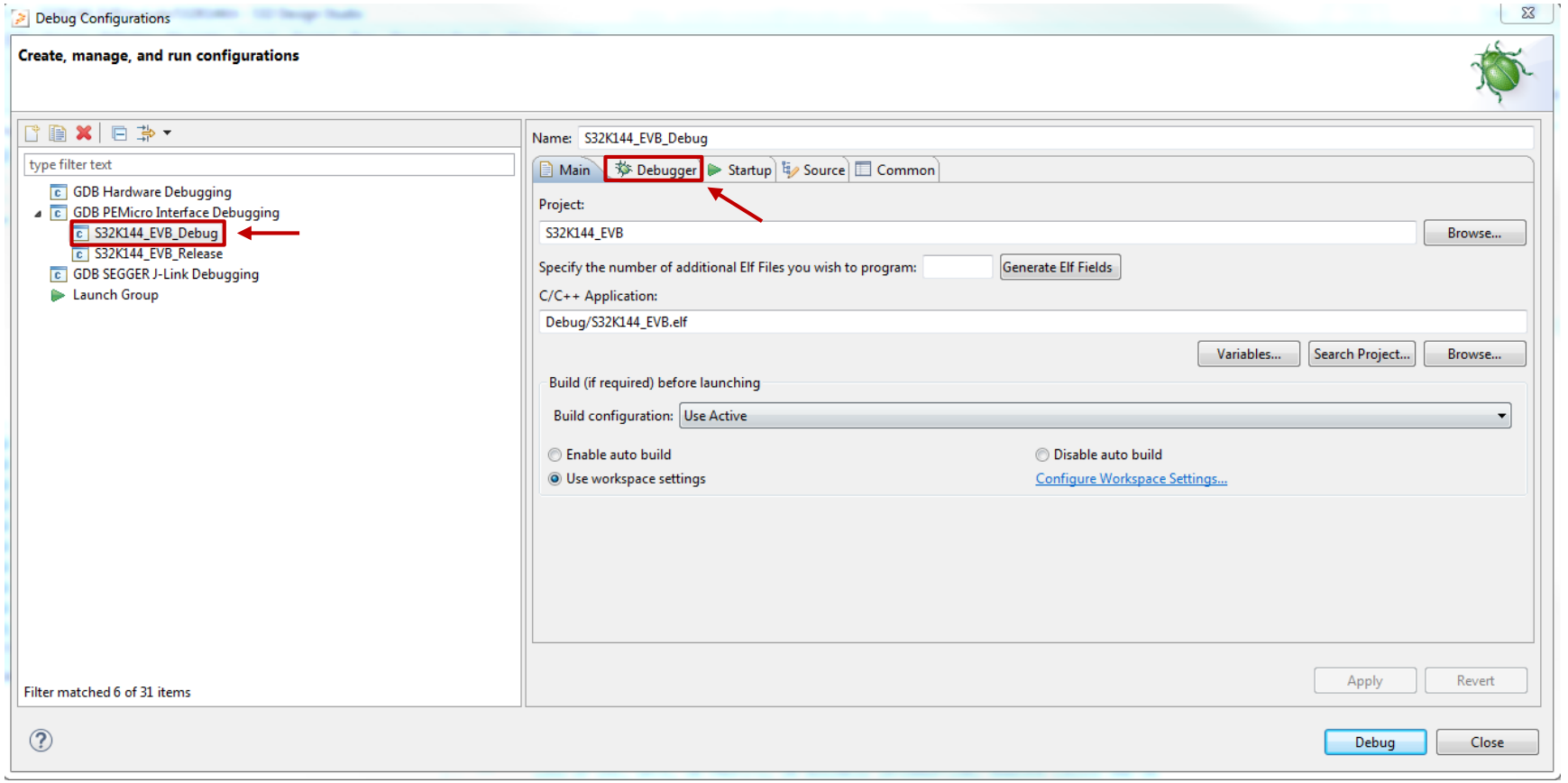
S32DS – OpenSDA Debug Configuration / OpenSDA调试配置

- Select the Debug configuration under GDB PEMicro Interface Debugging

  在GDB PEMicro Interface Debugging选项卡下，选择对应工程配置

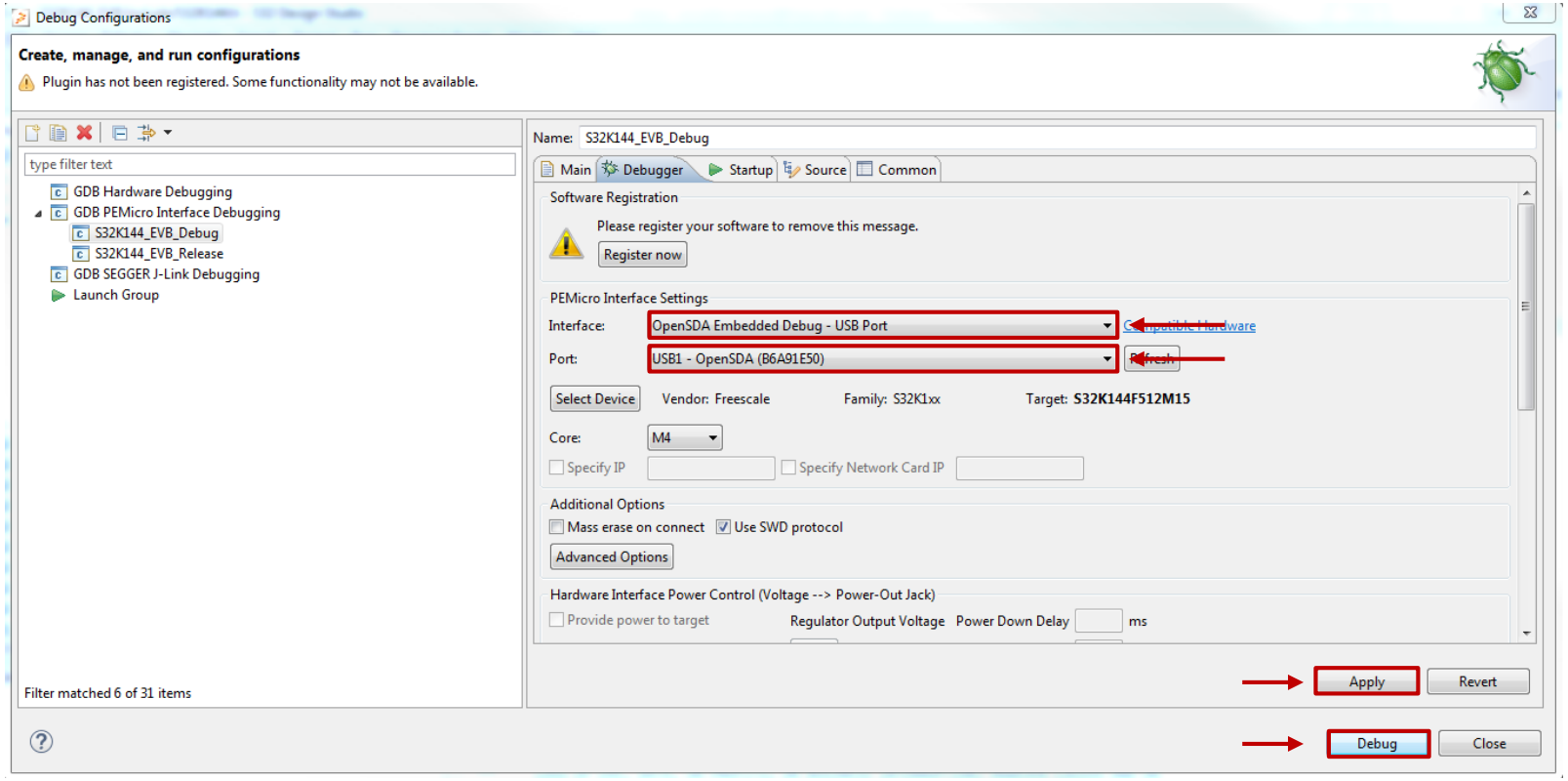
- Click on Debugger tab

  点击Debugger标签栏



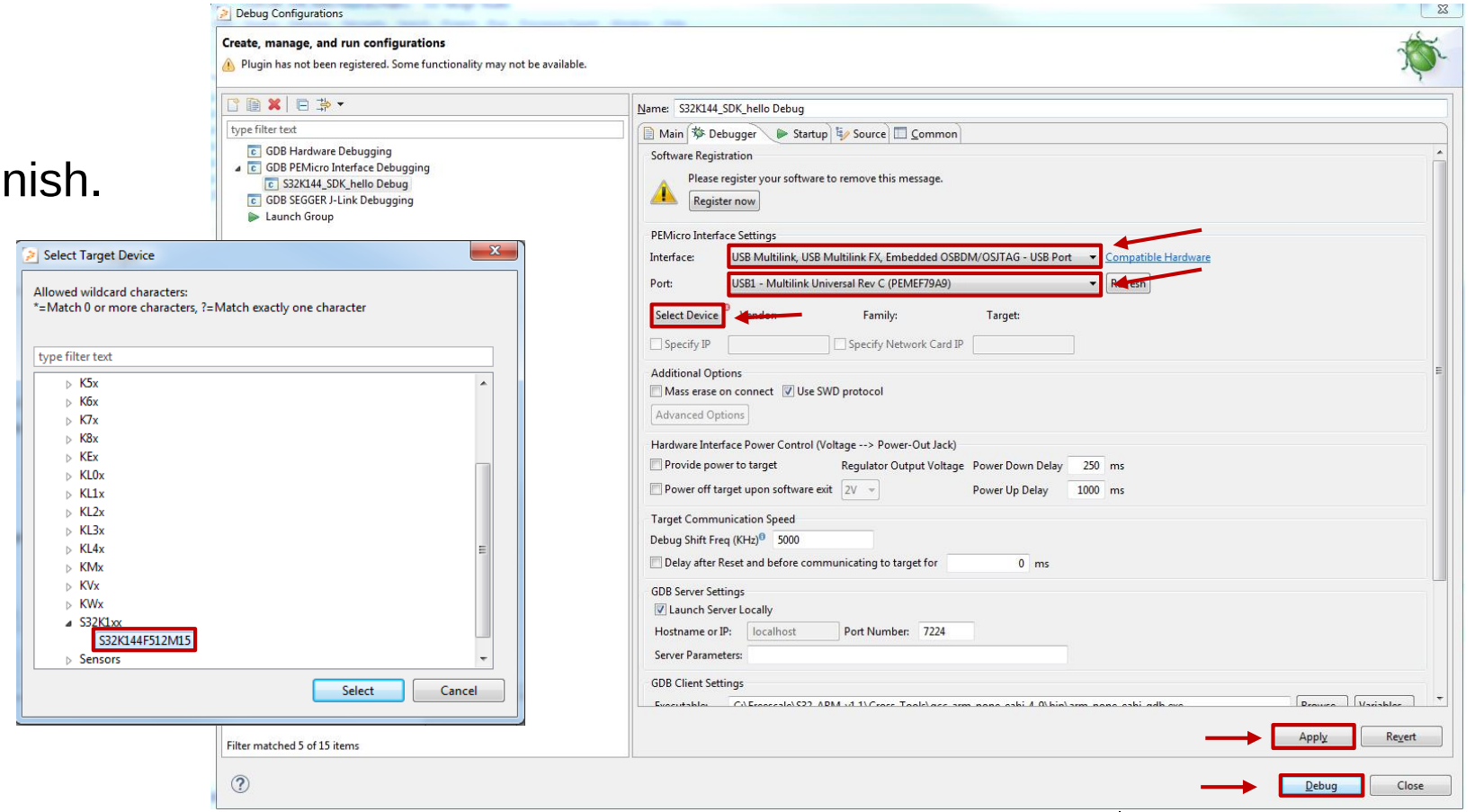
S32DS – OpenSDA Debug Configuration / OpenSDA调试配置

- Select OpenSDA as the interface, if your board is plugged should appear in the Port field.
  选择OpenSDA作为调试借口，将MicroUSB插在开发板上后，会在Port栏显示相应接口
- Click Apply and debug to finish.
  点击Apply-Debug



S32DS – P&E Multilink Debug Configuration / Multilink调试配置

- Select USB Multilink as the interface, if your board is plugged should appear in the Port field.
  选择USB Multilink作为调试借口，将调试器插在开发板上后，会在Port栏显示相应接口
- Select S32K144 Device
  选择芯片型号
- Click Apply and debug to finish.
  点击Apply-Debug



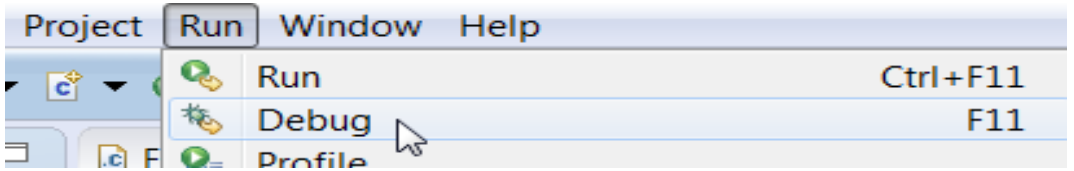
S32DS – Starting the Debugger / 调试入门

- Debug configuration is only required once. Subsequent starting of debugger does not require those steps.

  Debug Configuration只需配置一次，后续的调试无需配置。
- Three options to start debugger 启动调试器的三种方法
  - If the “Debug Configuration” has not been closed, click on “Debug” button on bottom right

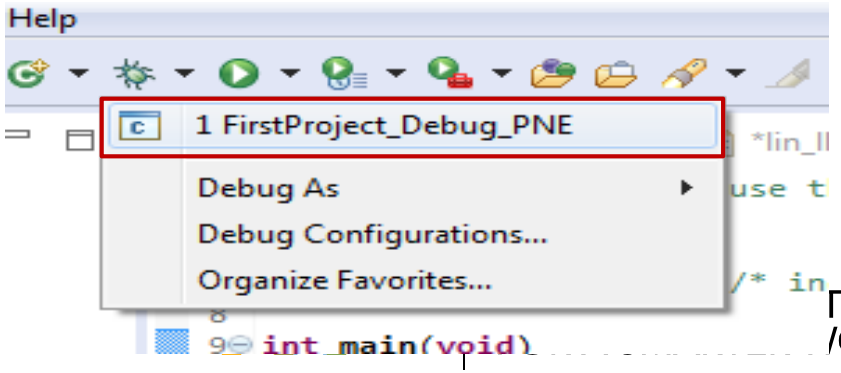
    如Debug Configuration窗口未关闭，直接点击右下角Debug按钮
  - Select Run – Debug (or hit F11)

    选择菜单按钮Run-Debug（或者按F11）



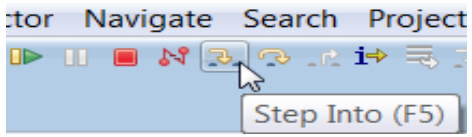
  - *Recommended Method*: Click on pull down arrow for bug icon and select …_debug.elf target

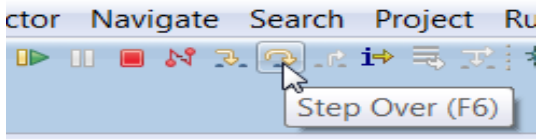
    推荐方法，点击 “小虫子”按钮下拉菜单对应的调试目标

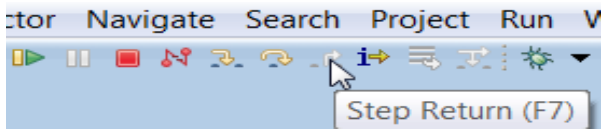


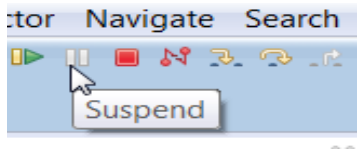
Debug Basics: Step, Run, Suspend, Resume / 调试基本操作

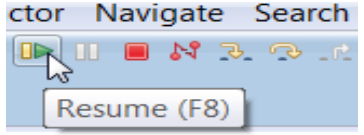
- Step Into (F5)
  单步运行，进入该语句对应函数
- Step Over (F6)
  逐程序运行，不进入该语句对应函数
- Step Return (F7)
  逐程序返回，返回至下一函数
- Suspend
  暂停
- Resume (F8)
  全速运行/继续
- Terminate (Ctl+F2)
  停止

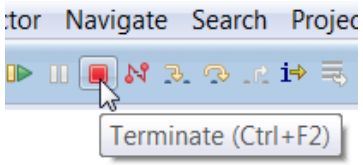






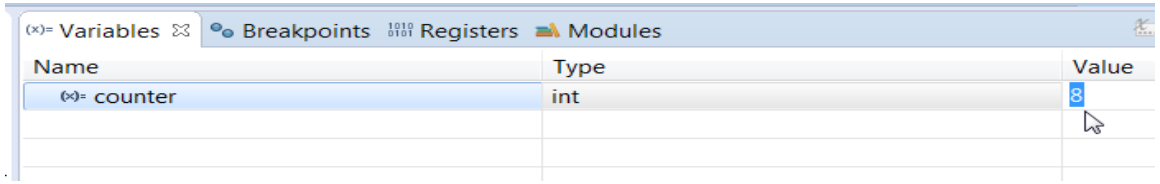


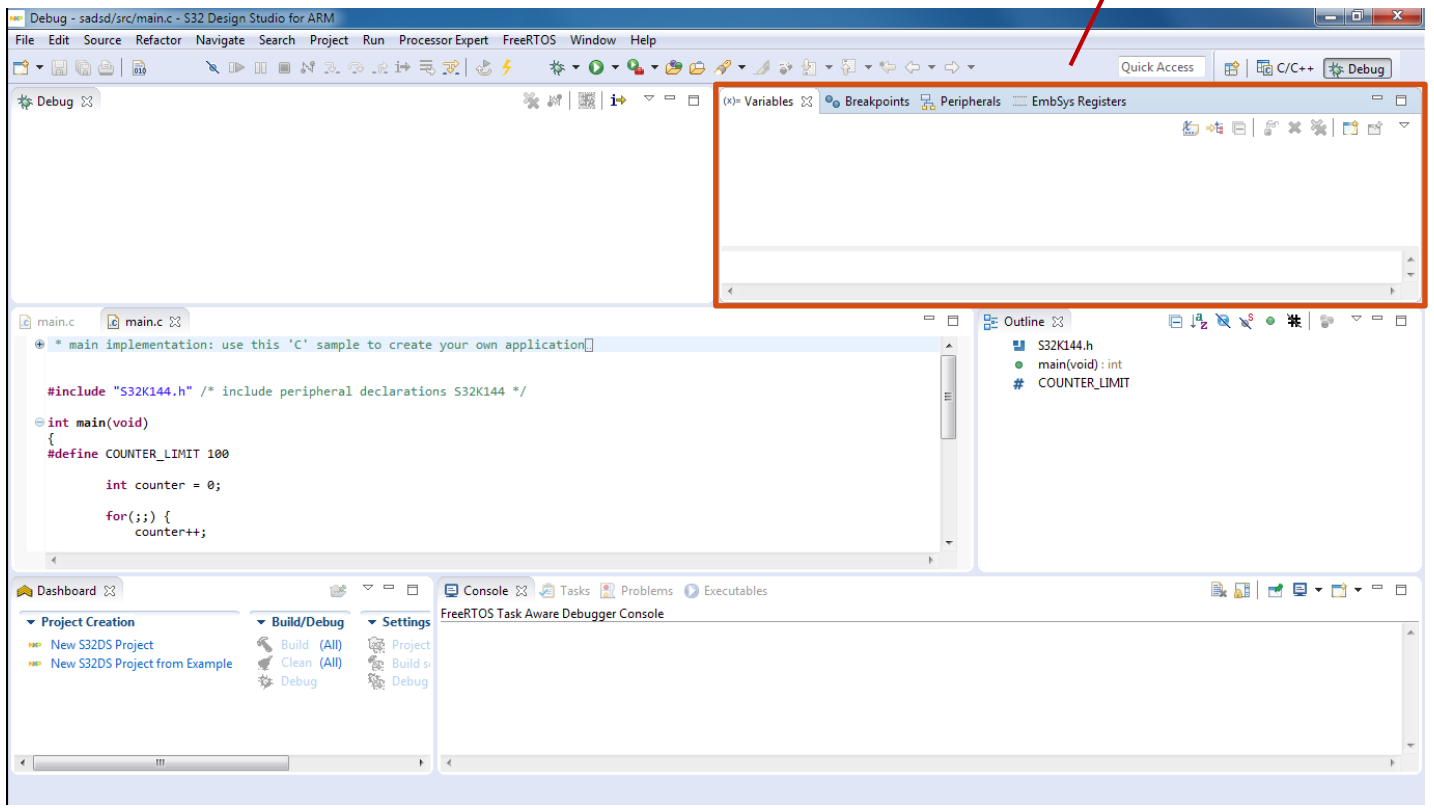




Debug Basics: View & Alter Variables / 查看&修改变量

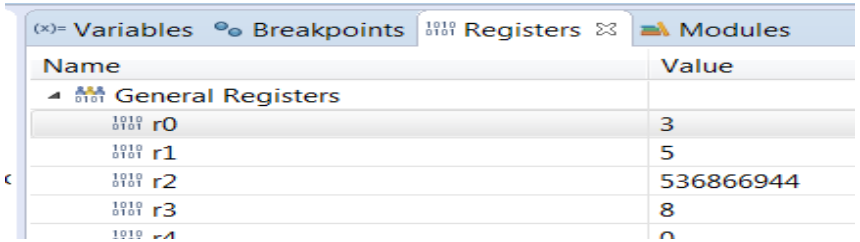
- View variables in “Variables” tab.
  在“Variables”标签栏查看变量的值
- Click on a value to allow typing in a different value.
  点击Value可以对变量行进修改

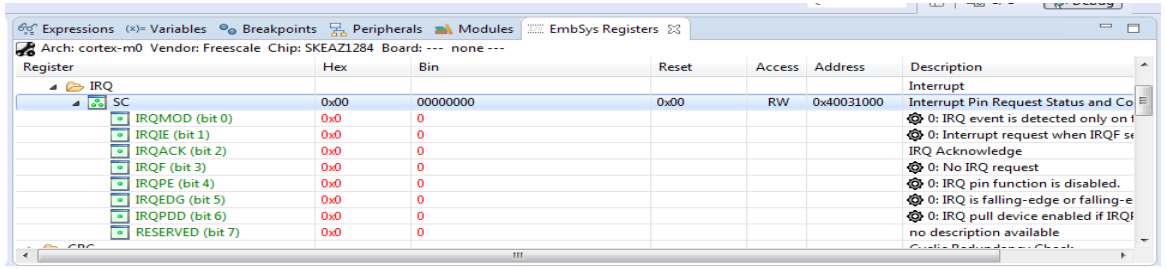


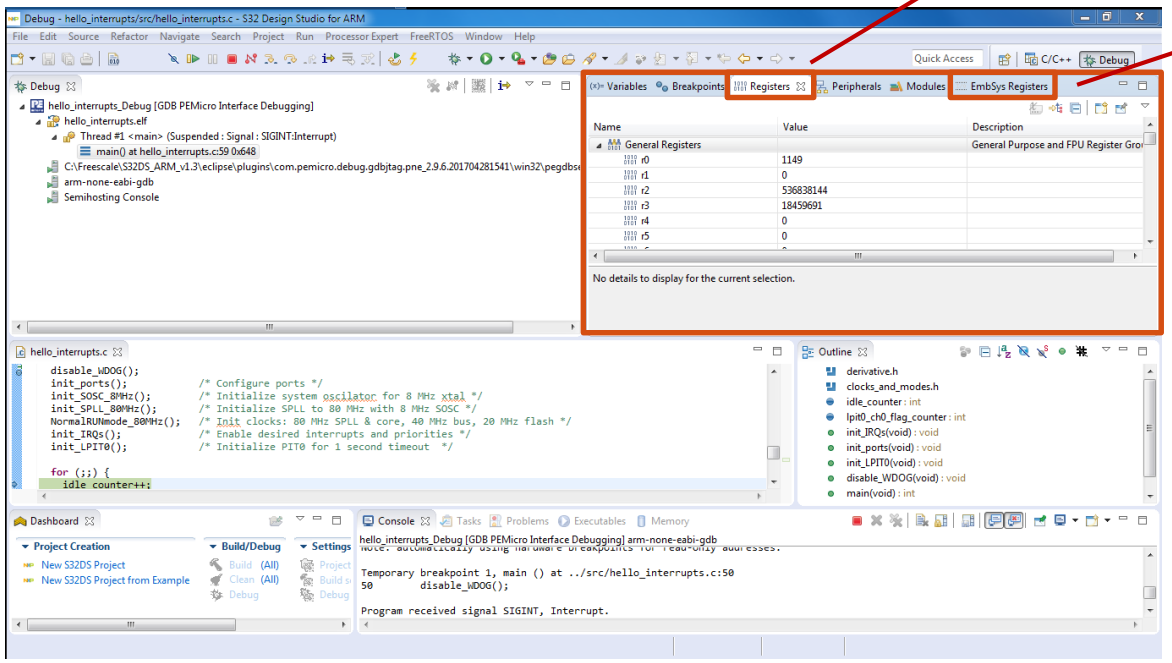


Debug Basics: View & Alter Registers / 查看&修改寄存器

- View CPU registers in the “Registers” tab
  在“Registers”标签栏查看CPU通用寄存器
- Click on a value to allow typing in a different value
  点击Value可以修改寄存器的值
- View peripheral registers in the “EmbSys Registers” tab
  在“EmbSys Registers”标签栏中查看每个外设寄存器的值







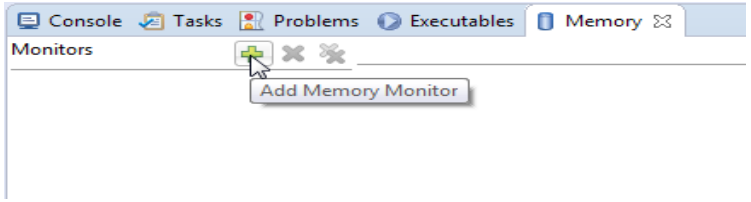
Debug Basics: View & Alter Memory / 查看&修改内存中的值

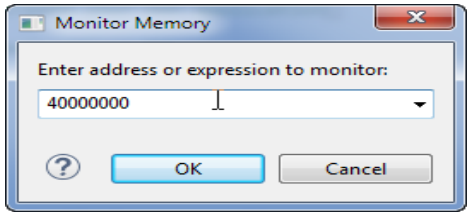
- Add Memory Monitor

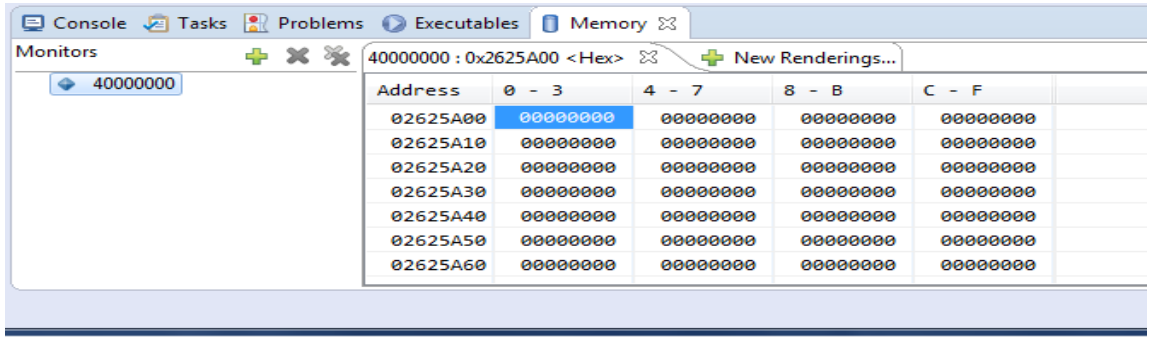
  添加Memory监控器窗口
- Select Base Address to Start at : 40000000

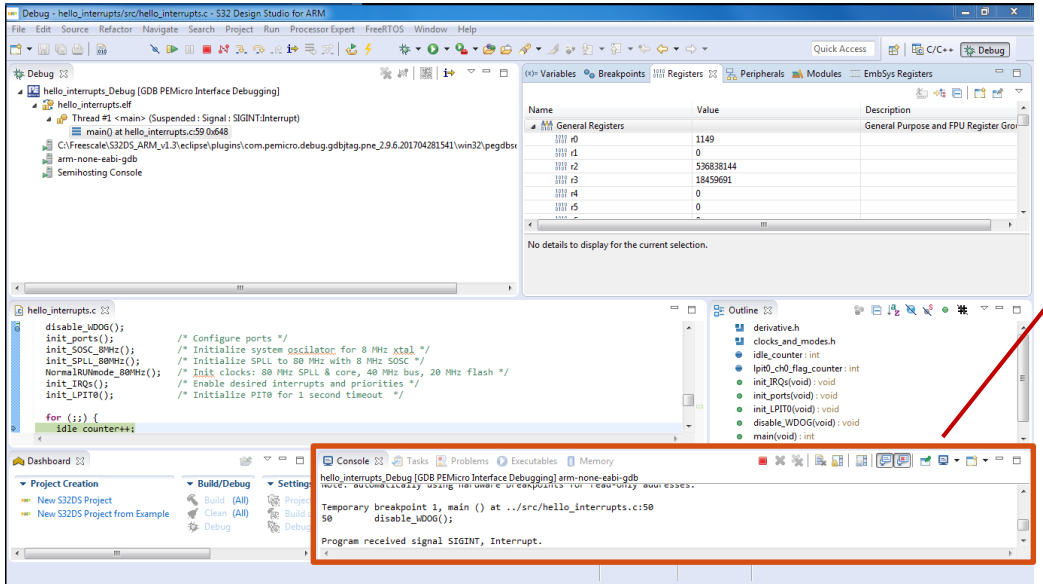
  选择基地址，从0x40000000开始
- View Memory

  查看Memory







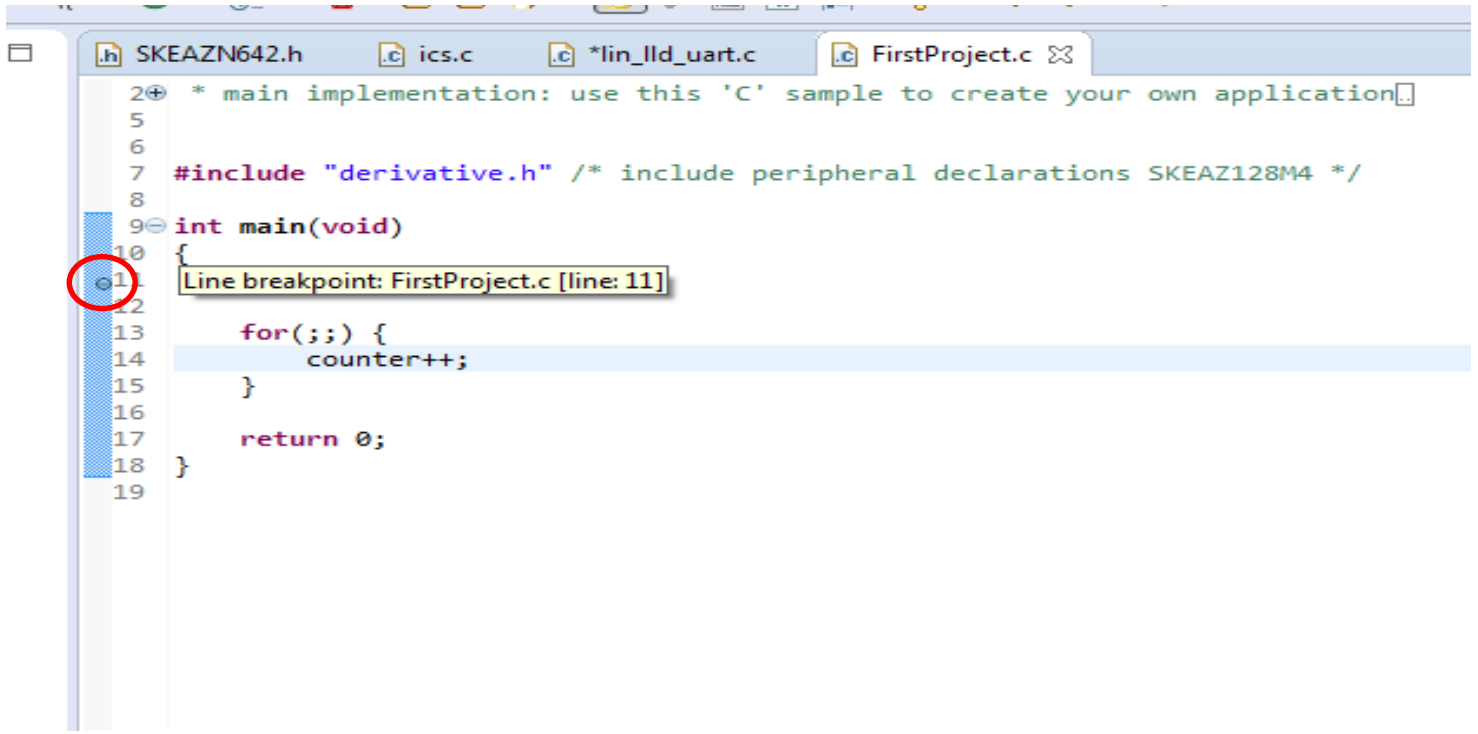


Debug Basics: Breakpoints / 设置断点

- Add Breakpoint: Point and Click

  设置断点，在相应代码所在行左侧双击
- light blue dot represents debugger breakpoint

  蓝色原点表示断点



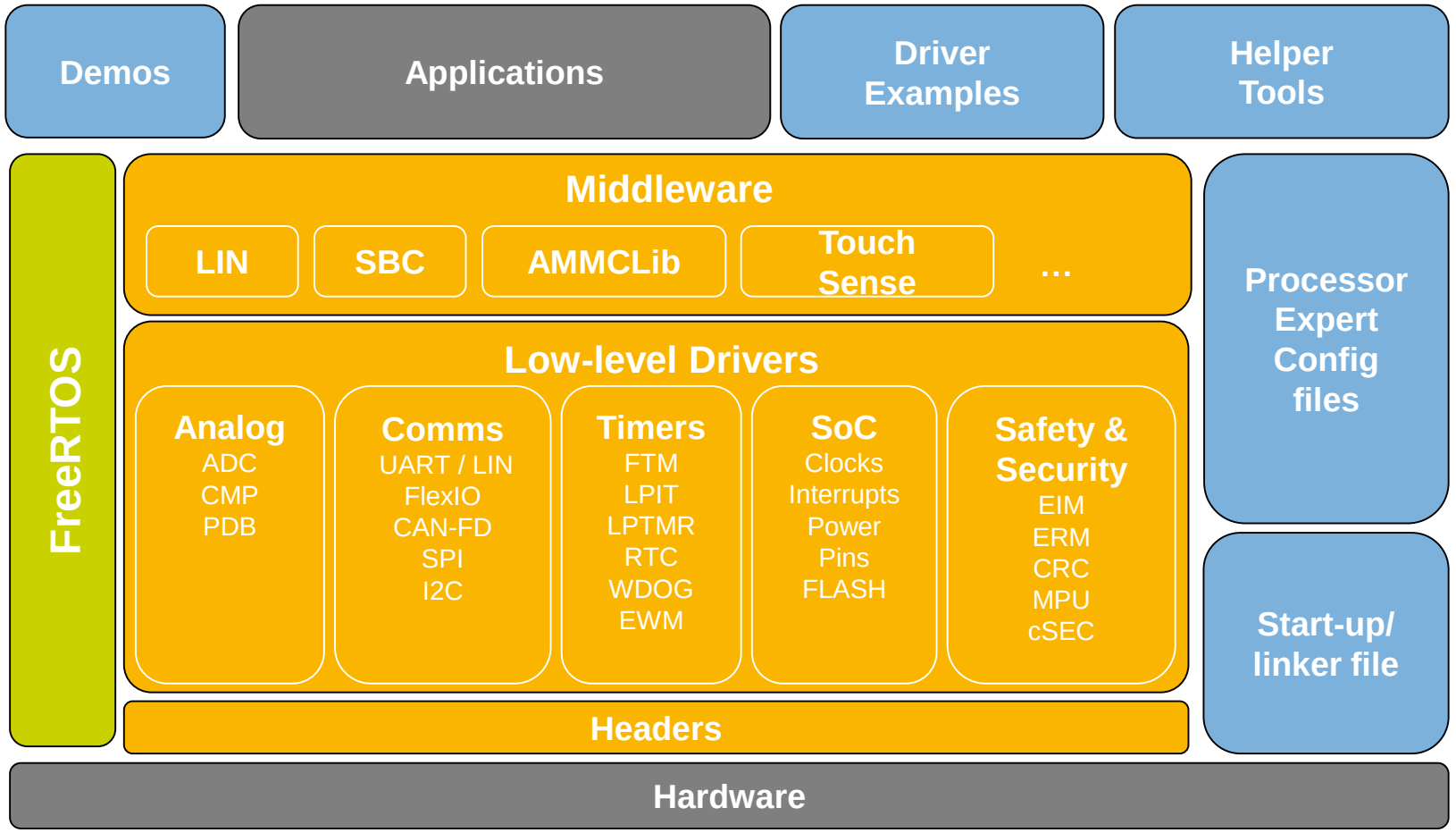
S32K SDK HANDS-ON
S32K SDK使用操作样例

SDK Product Overview / 软件开发包

Class B — SPICE Compliant Products

Class C — Min. Quality Compliant (ISO/TS16949)

Class D — Open Source Software



Features

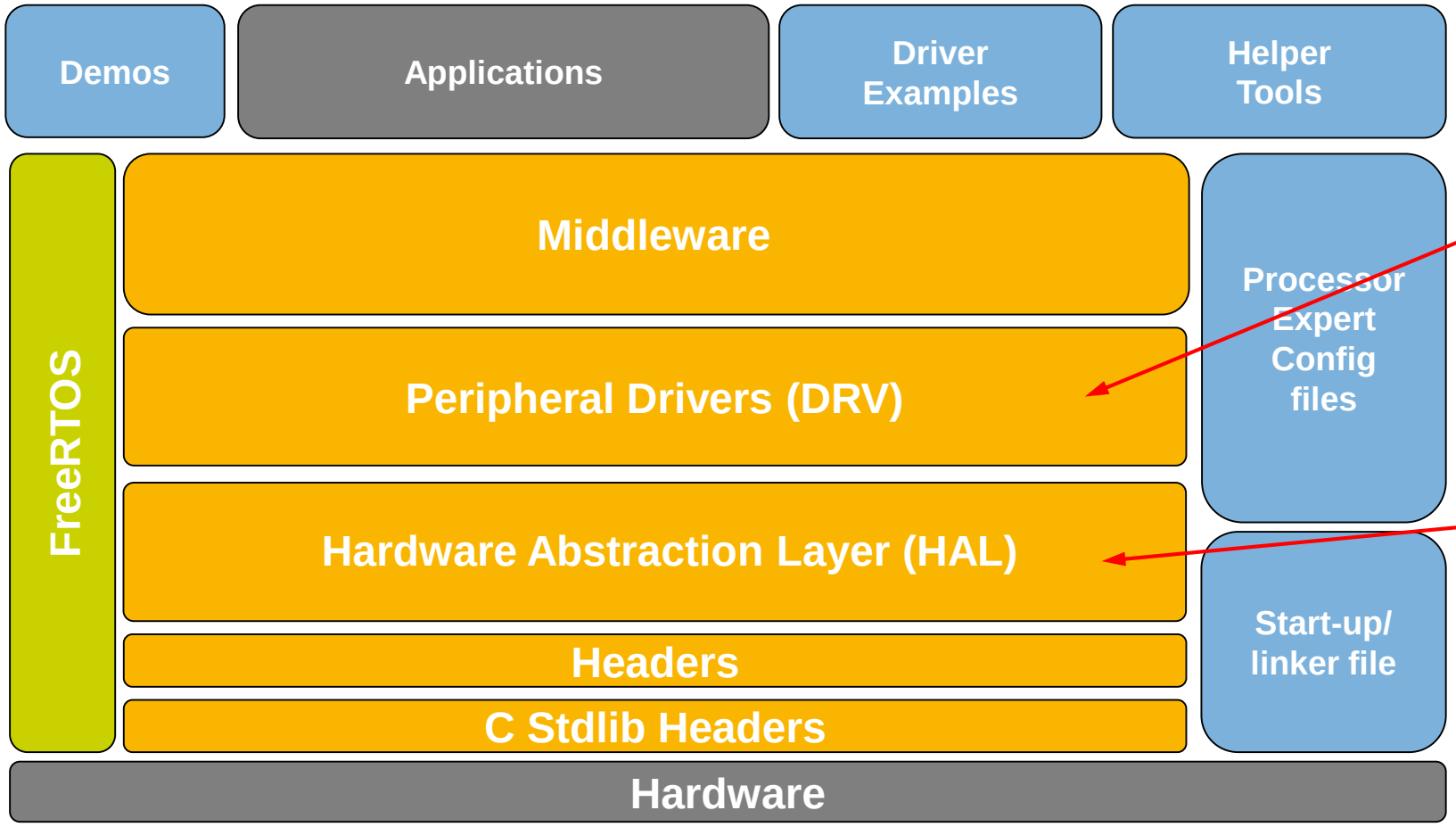
- Integrated Non-Autosar SW package
  集成的非AUTOSAR架构的软件包
- Graphical-based configuration
  图形化配置界面
- Layered SW architecture
  分层的软件架构
- Documented Source code and examples
  源代码和样例的完备文档描述
- Integrated with S32 Design Studio
  集成在S32DS开发环境中
- Featuring various Middleware (sold separately, demo as binary)
  丰富多样的中间件软件（需单独购买）
- FreeRTOS integration
  集成FreeRTOS实时操作系统
- Multiple toolchains supported
  支持多种工具链
- Several examples and demos
  完整的样例代码

SDK SW Architect / SDK软件架构

Class B — SPICE Compliant Products

Class C — Min. Quality Compliant (ISO/TS16949)

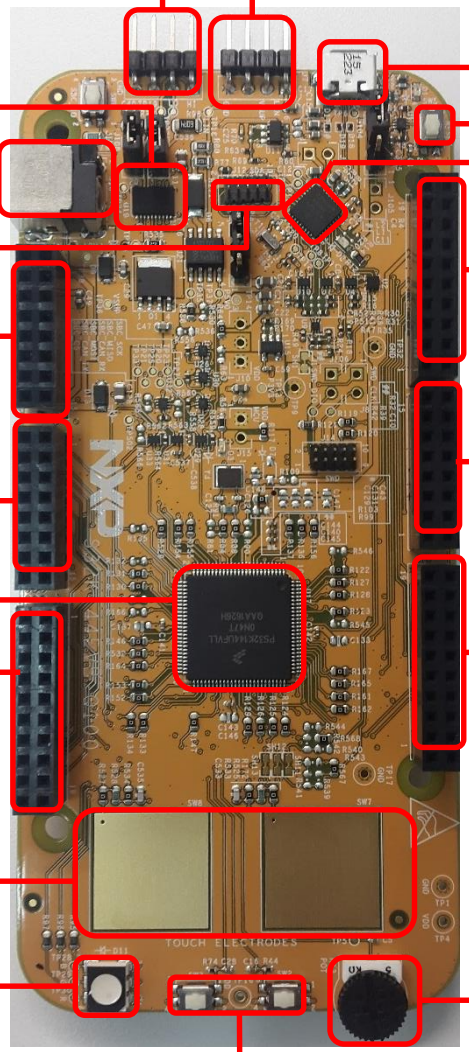
Class D — Open Source Software



Peripheral Drivers
- High-level drivers
  底层驱动
- Handle entire functionality
  针对功能的实现
- Based on HAL/other peripheral drivers/system services
  基于HAL/其他外设驱动/系统服务函数
- E.g, ADC_DRV_ConfigConverter()

Low-level drivers
- Abstract hardware peripherals register access
  访问硬件外设寄存器的API
- E.g, ADC_HAL_SetClockDivide()

Get to know S32K EVB / S32K开发板资源说明



Jumper	Configuration	Description
J104	1-2	Reset signal to OpenSDA, use to enter into OpenSDA Bootloader mode
	2-3 (Default)	Reset signal direct to the MCU, use to reset S32K144.
J107	1-2	S32K144 powered by 12V power source.
	2-3 (Default)	S32K144 powered by USB micro connector.
J109/J108	1-2 (Default)	Removes CAN termination resistor

Component	S32K144
Red LED	PTD15 (FTM0 CH0)
Blue LED	PTD0(FTM0 CH2)
Green LED	PTD16(FTM0 CH1)
Potentiometer	PTC14 (ADC0_SE12)
SW2	PTC12
SW3	PTC13
OpenSDA UART TX	PTC7(LPUART1_TX)
OpenSDA UART RX	PTC6(LPUART1_RX)

Component	S32K144
CAN TX	PTE5(CAN0_TX)
CAN RX	PTE4 (CAN0_RX)
LIN TX	PTD7(LPUART2_TX)
LIN RX	PTD6 (LPUART2_RX)
SBC_SCK	PTB14 (LPSPI1_SCK)
SBC_MISO	PTB15(LPSPI1_SIN)
SBC_MOSI	PTB16(LPSPI1_SOUT)
SBC_CS	PTB17(LPSPI1_PCS3)

Lab 01.

S32K144 CLOCKING

- In this lab you will learn:

  本实验你将学习以下内容

  - About the clock tree in S32K144

    S32K144时钟树分布

  - How to create a new SDK project with S32DS.

    使用S32DS新建SDK工程

  - How to setup S32K144 in the following clock configuration

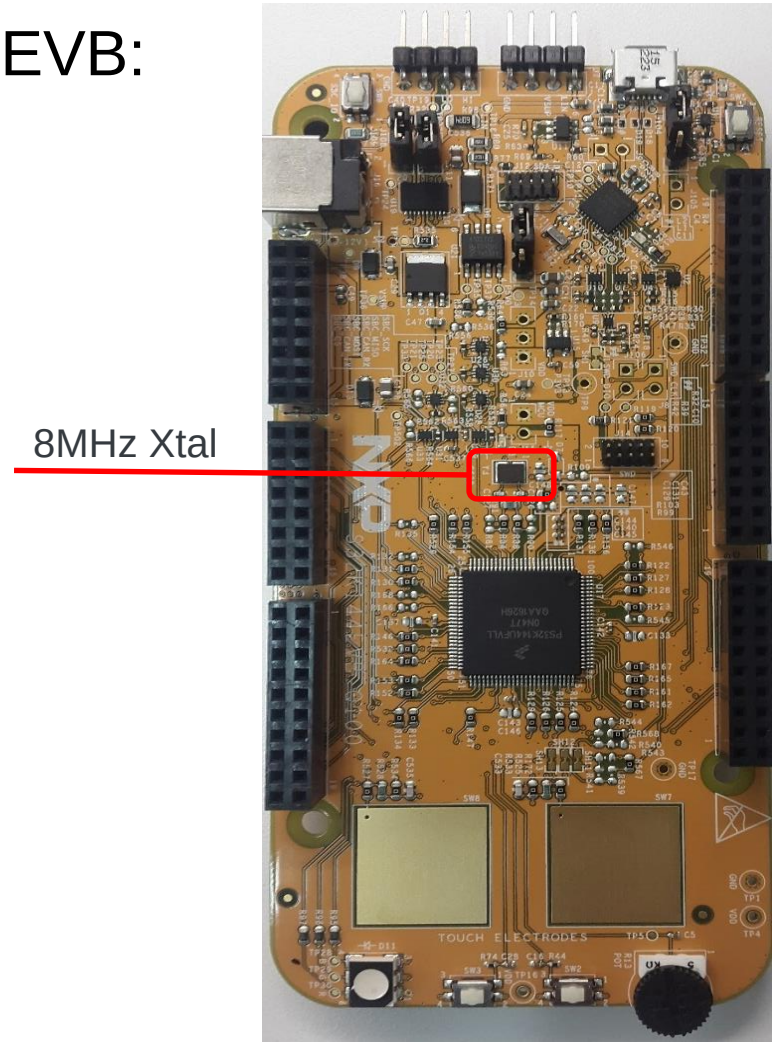
    如何对S32K144进行如下时钟配置

    - Core Clock 内核时钟: 50MHz
    - Bus Clock 总线时钟: 25MHz
    - Flash Clock Flash时钟: 25MHz

S32K144 Clocks Lab: Resources to be used

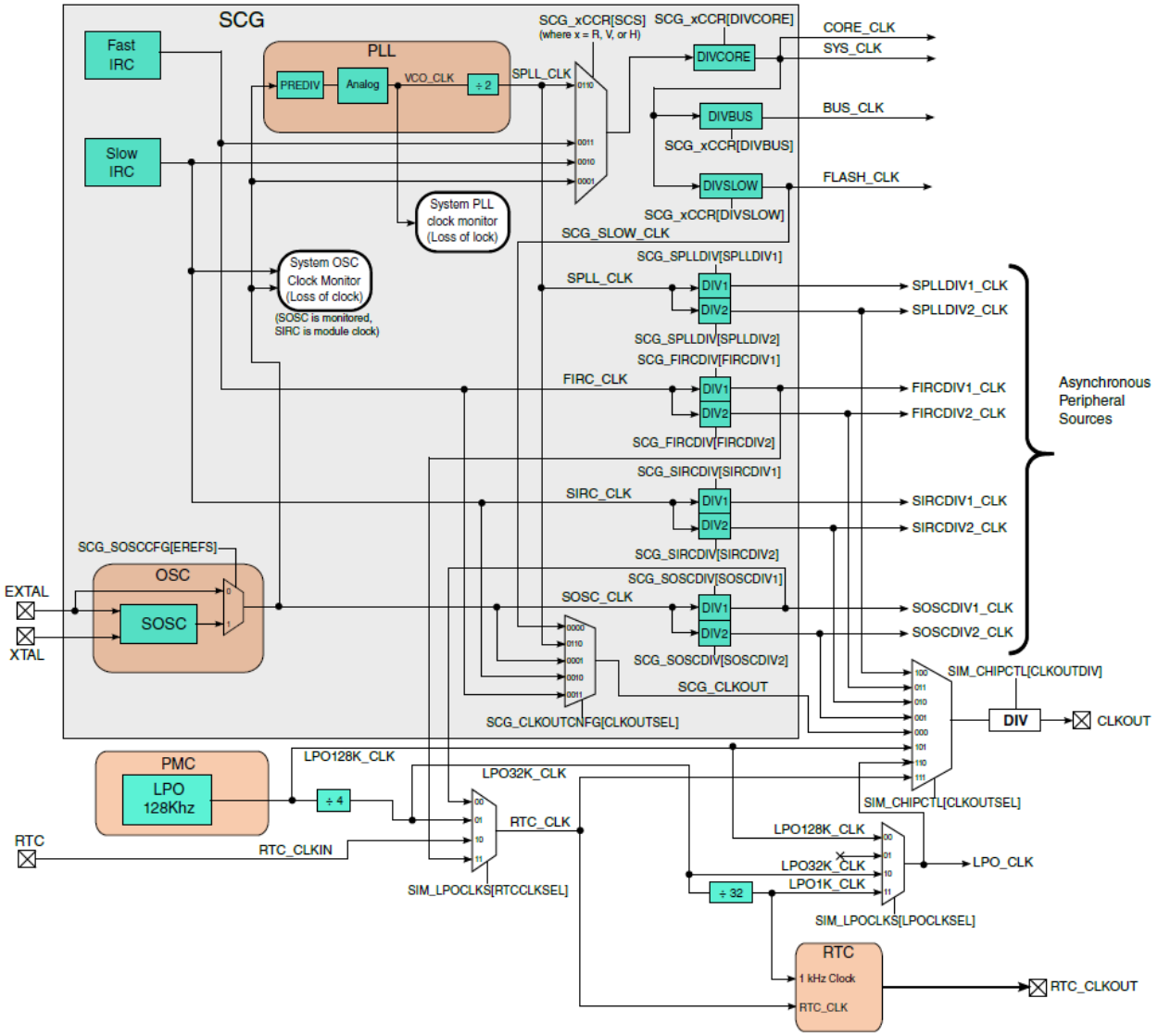
- In this lab will be used the following components of the EVB:
  本实验中，你将使用以下EVB资源
  - 8 MHz Xtal 晶振

XTAL	S32K144 PIN
1	PTB7
2	PTB6



S32K144 Clocks Lab: Clock Distribution / 时钟树分布

- Clock source for the core:
  内核时钟源
  - FIRC (48MHz)
  - SIRC (8 MHz)
  - PLL (up to 112 MHz)
  - SOSC (8 – 40 MHz)
- PLL Flexible multiplier (16 – 47)
  PLL灵活的倍频系数（16-47）
- PLL source is external oscillator
  外部晶振作为PLL时钟源
- By default device clock is FIRC. Core Frequency is 48 MHz.
  缺省状态芯片时钟为FIRC，48MHz
- Multiple clock source for peripherals
  每个外设有多种时钟源



S32K144 Clocks Lab: Clock Description / 时钟描述

Clock Name	Description
Core Clock	Clocks the ARM core
System Clock	Clocks the Crossbar, NVIC, Flash controller, FTM, PDB, and so on. Same as Core Clock
Bus Clock	Clocks the chip Peripherals
Flash Clock	Clocks the flash module
SPLL DIVx Clock	Optional divided PLL source for peripherals
SIRC DIVx Clock	Optional divided SIRC source for peripherals
FIRC DIVx Clock	Optional divided FIRC source for peripherals
OSC DIVx Clock	Optional divided System Oscillator clock for peripherals
LPO Clock	Low power oscillator clock inside PMC
RTC Clock Out	RTC Oscillator output driving external pin
Clock Out	Optional output clock source for external devices

S32K144 Clocks Lab: Clock in power mode / 功耗模式的时钟配置

- For each power mode (HSRUN,RUN,VLPR) there are three dividers that are controlled independently:
  针对每个功耗模式（HSRUN,RUN,VLPR）都有相应的3个分频器，可单独对每个功耗模式下的时钟进行配置
  - DIVCORE: Core Clock divider 内核时钟分频系数
  - DIVBUS: Bus Clock divider 总线时钟分频系数
  - DIVSLOW: Flash Clock divider Flash时钟分频系数
- Each of these dividers bit fields inside the Clock Control Register of each mode
  分频系数的比特域在对应模式的Clock Control Register寄存器中（SCG_RCCR，SCG_VCCR，SCG_HCCR）
- The source clock for each mode is selected with the SCS bits:
  通过SCS比特域可以选择每个功耗模式下的时钟源
  - FIRC
  - SIRC
  - PLL
  - OSC

26.3.4 Run Clock Control Register (SCG_RCCR)

This register controls the system clock source and the system clock dividers for the core, platform, external and bus clock domains when in Run mode only. This register can only be written using a 32-bit write. Selecting a different clock source when in RUN requires that clock source to be enabled first and be valid before system clocks switch to that clock source. If system clock divide ratios also change when selecting a different clock mode when in RUN, new system clock divide ratios will not take affect until new clock source is valid.

Address: 0h base + 14h offset = 14h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0				SCS				0				DIVCORE				Reserved				0				DIVBUS				DIVSLOW			
W																																
Reset	0	0	0	0	0	0	0	1	1	0	0	0	*	*	*	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

* Notes:
- DIVCORE field: The reset value is controlled by user FOPT bits that get uploaded during reset. The two option reset values are div-by-1 and div-by-2

19–16 DIVCORE	Core Clock Divide Ratio 0000 Divide-by-1 0001 Divide-by-2 0010 Divide-by-3 0011 Divide-by-4 0100 Divide-by-5 0101 Divide-by-6 0110 Divide-by-7 0111 Divide-by-8 1000 Divide-by-9

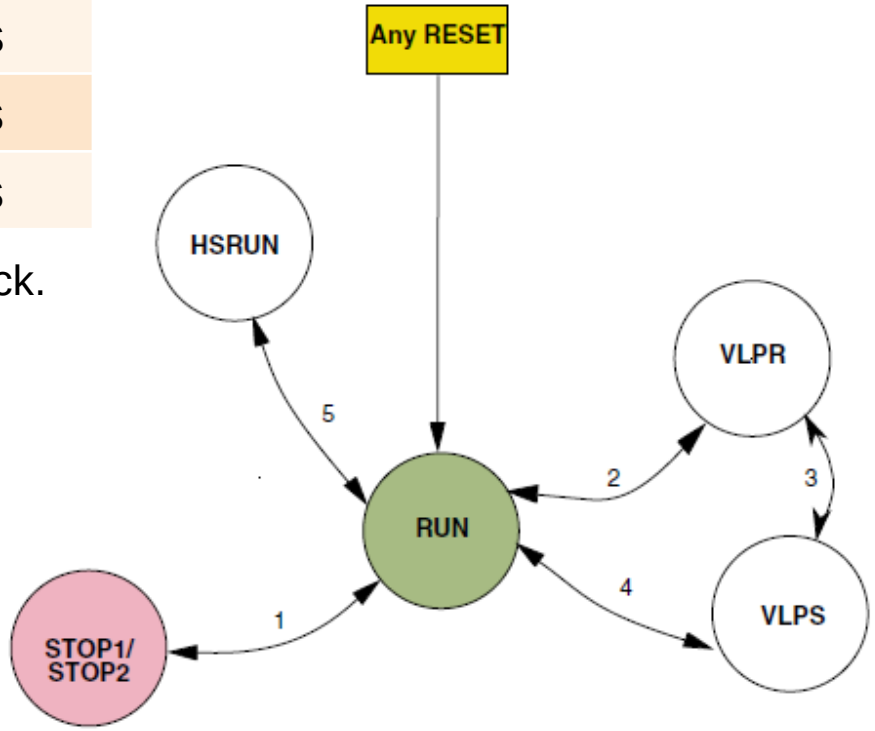
27–24 SCS	System Clock Source Selects the clock source generating the system clock in Run mode. Attempting to select a clock that is not valid will be ignored. Selecting a different clock source when in Run mode requires that clock source to be enabled first and be valid before system clocks are allowed to switch to that clock source. 0000 Reserved 0001 System OSC (SOSC_CLK) 0010 Slow IRC (SIRC_CLK) 0011 Fast IRC (FIRC_CLK) 0100 Reserved 0101 Reserved 0110 System PLL (SPLL_CLK) 0111 Reserved

S32K144 Clocks Lab: Clock Requirement / 时钟需求

	HSRUN	RUN	VLPR
Core Clock*	112MHz or less	80MHz or less	4MHz or less
System Clock*	112MHz or less	80MHz or less	4MHz or less
Bus Clock*	56MHz or less	48MHz or less	4MHz or less
Flash Clock	28MHz or less	26.67MHz or less	1MHz or less

NOTE: System clock and core clock must not be configured to less than bus clock.

Please see Chapter 25.4 of reference manual for more details.
每个模块的详细内容请参见Reference Manual 25.4章节。



S32K144 Clocks Lab: Peripherals Clock Configuration / 外设时钟配置

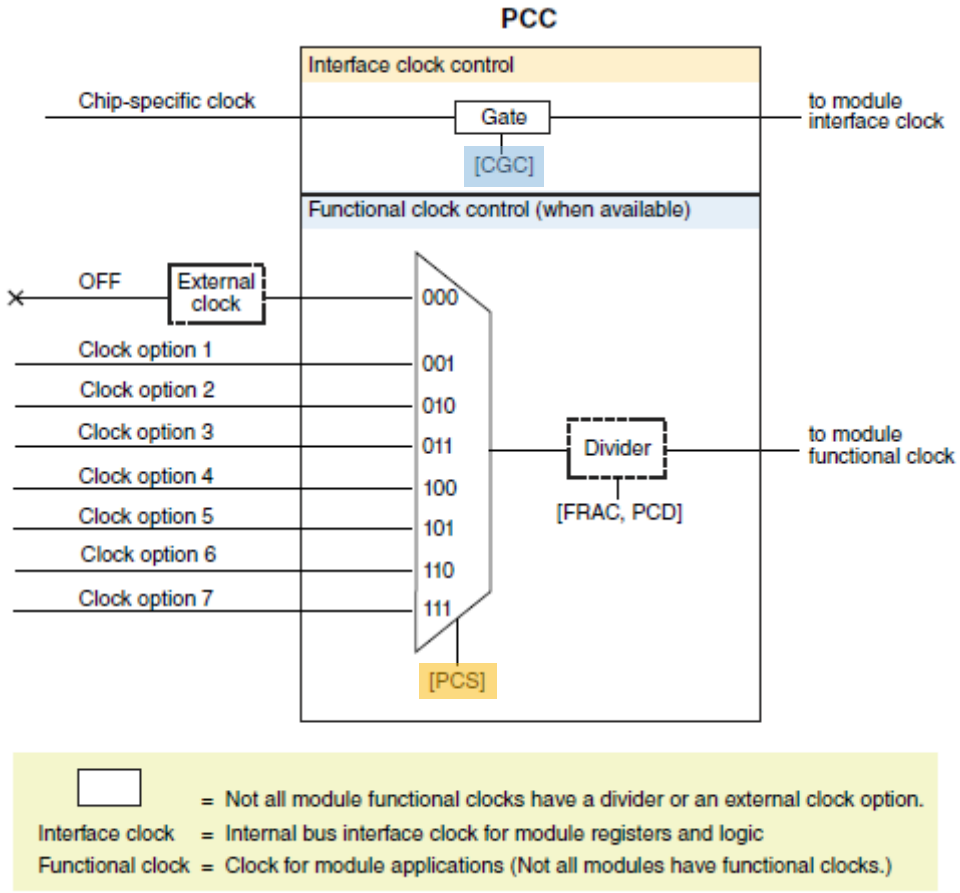
- Modules can be individually turn on or off using the PCC module.
  通过PCC模块，外设可以独立打开或关闭
- Clock source for each peripheral can be selected from multiple sources.
  每个外设时钟源都有多种可选
- Before using a peripheral, turn on its clock.
  使用某外设之前需要打开其时钟

PCC Register

27.6.19.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PR	CGC	-	0		PCS			0							
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0												0		0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



S32K144 Clocks Lab: Peripheral Clock Summary / 外设时钟总结

Module name	Bus interface clock	Bus interface clock cating	Peripheral functional clock	Additonal clocks	Comments and maximum frequencies
		Gated by [CGC] of PCC	Clocks controlled by [PCS] of PCC		
Communications					
LPUART[0:3]	BUS_CLK	Yes	SPLLDIV2_CLK, FIRCDIV2_CLK, SIRCDIV2_CLK, SOSCDIV2_CLK	—	Maximum frequency governed by BUS_CLK
LPSPI[0:2]	BUS_CLK	Yes	SPLLDIV2_CLK, FIRCDIV2_CLK, SIRCDIV2_CLK, SOSCDIV2_CLK	—	Maximum frequency governed by BUS_CLK
LPI2C0	BUS_CLK	Yes	SPLLDIV2_CLK, FIRCDIV2_CLK, SIRCDIV2_CLK, SOSCDIV2_CLK	—	Maximum frequency governed by BUS_CLK
FlexIO	BUS_CLK	Yes	SPLLDIV2_CLK, FIRCDIV2_CLK, SIRCDIV2_CLK, SOSCDIV2_CLK	—	Maximum frequency governed by BUS_CLK
FlexCAN[0:2]	SYS_CLK	Yes	—	BUS_CLK, SOSCDIV2_CLK	Support 40 MHz from OSC; BUS_CLK must be >1.5x the protocol clock; while synchronous operation (when protocol clock is selected to BUS_CLK) can be done at 1:1 clock frequency.
Timers					
LPTMR	BUS_CLK	Yes	SPLLDIV2_CLK, FIRCDIV2_CLK, SIRCDIV2_CLK, SOSCDIV2_CLK	CLK32K[1], SIRCDIV2_CLK, LPO1K_CLK	Maximum frequency governed by BUS_CLK
LPIT	BUS_CLK	Yes	SPLLDIV2_CLK, FIRCDIV2_CLK, SIRCDIV2_CLK, SOSCDIV2_CLK	—	Maximum frequency governed by BUS_CLK
RTC	BUS_CLK	Yes	—	CLK32K[1], LPO1K_CLK	—
PDB[0:1]	SYS_CLK	Yes	—	—	—

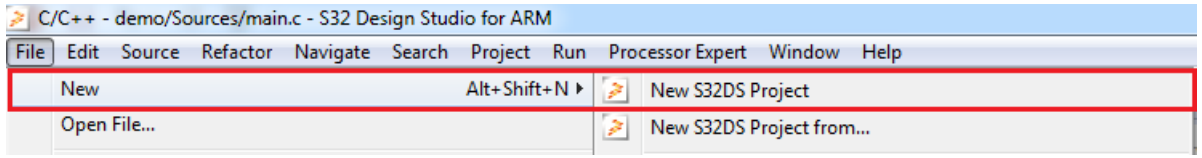
Module name	Bus interface clock [1]	Bus interface clock[1] gating	Peripheral functional clock	Additonal clocks	Comments and maximum frequencies
		Gated by [CGC] of PCC	Clocks controlled by [PCS] of PCC		
TRGMUX	BUS_CLK	No	—	—	Maximum frequency governed by BUS_CLK
DMAMUX0	BUS_CLK	Yes	—	—	Maximum frequency governed by BUS_CLK
DMA	SYS_CLK	No	—	—	Maximum frequency governed by SYS_CLK
MPU	SYS_CLK	No	—	—	Maximum frequency governed by SYS_CLK
EIM	SYS_CLK	No	—	—	Maximum frequency governed by SYS_CLK
ERM	SYS_CLK	No	—	—	Maximum frequency governed by SYS_CLK
MSCM	SYS_CLK	No	—	—	Maximum frequency governed by SYS_CLK
Memory Modules					
FTFC	FLASH_CLK	Yes	—	—	Maximum frequency governed by FLASH_CLK
System RAM	SYS_CLK	No	—	—	Maximum frequency governed by SYS_CLK
Analog Modules					
ADC[0:1]	BUS_CLK	Yes	SPLLDIV2_CLK, FIRCDIV2_CLK, SIRCDIV2_CLK, SOSCDIV2_CLK	—	50 MHz[5]
CMP0	BUS_CLK	Yes	—	—	Maximum frequency governed by BUS_CLK

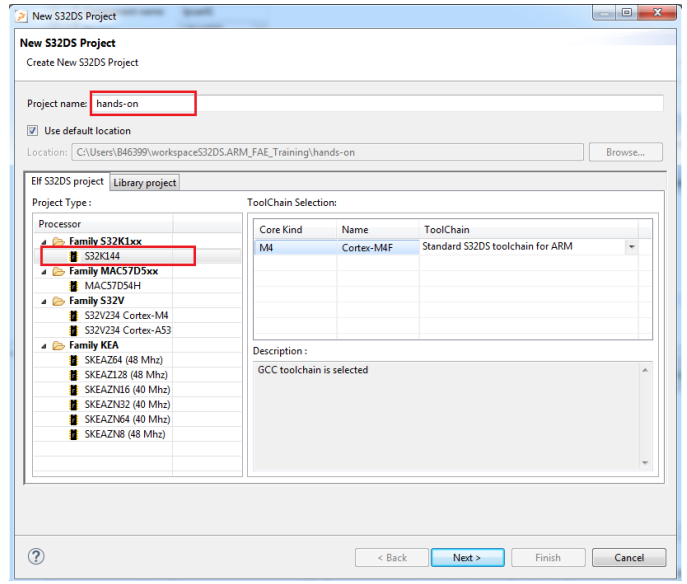
Please see Chapter 25.6 of reference manual for more details.
每个模块的详细内容请参见Reference Manual 25.6章节。

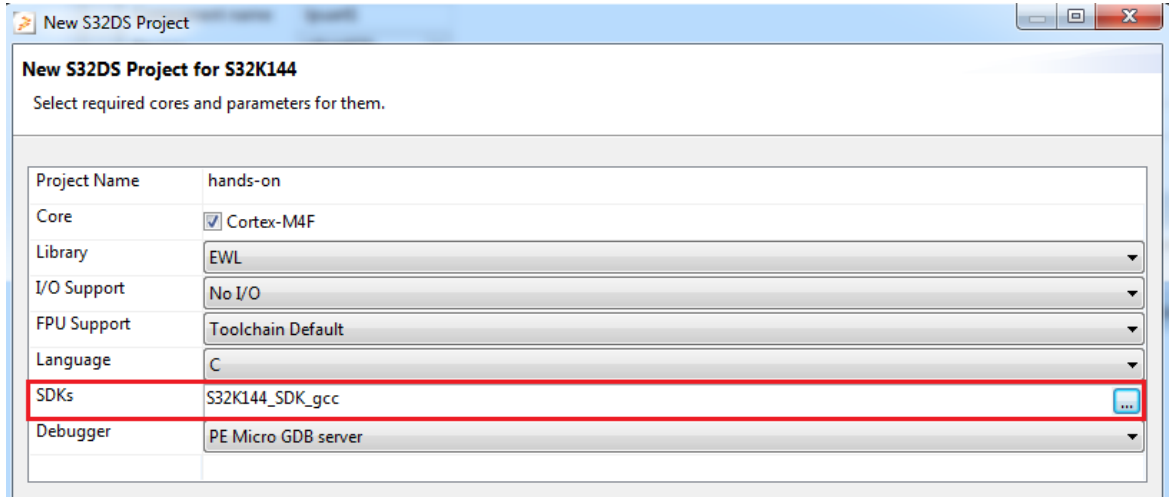
S32K144 Clocks Lab: Previous Steps / 准备步骤

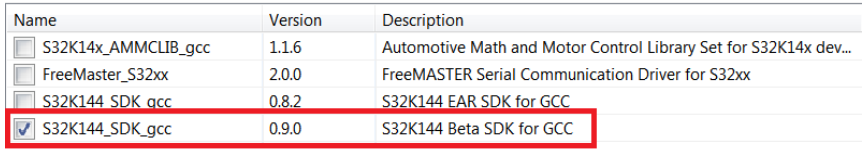
- Create a new S32DS Project

  新建S32DS工程



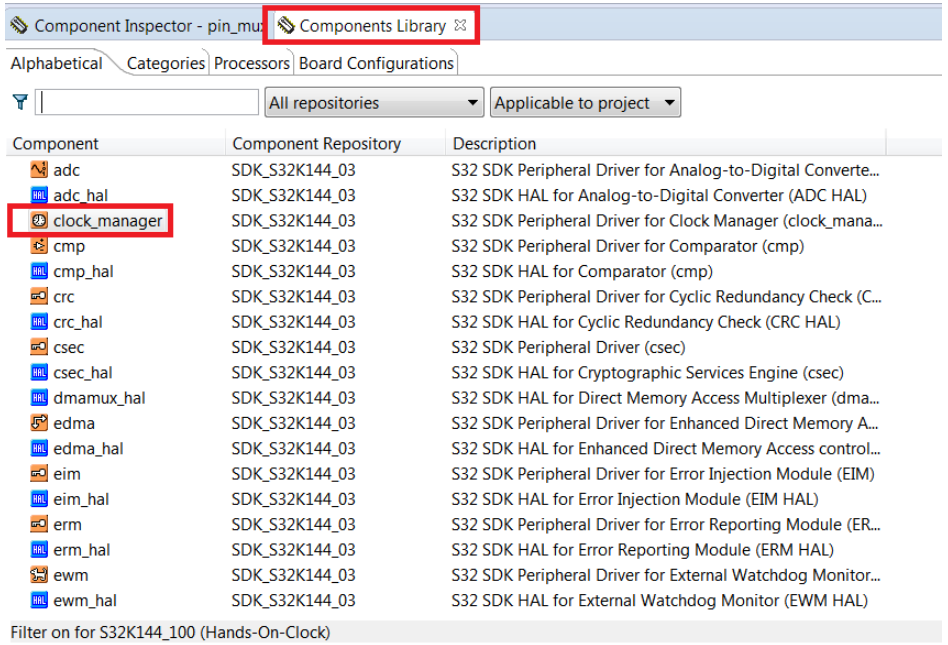


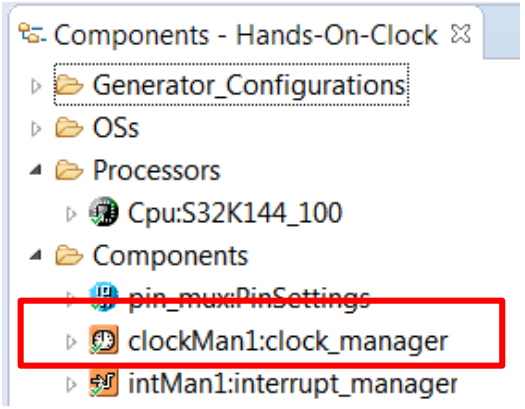




S32K144 Clocks Lab: Add Clock Manager / 添加Clock部件至工程

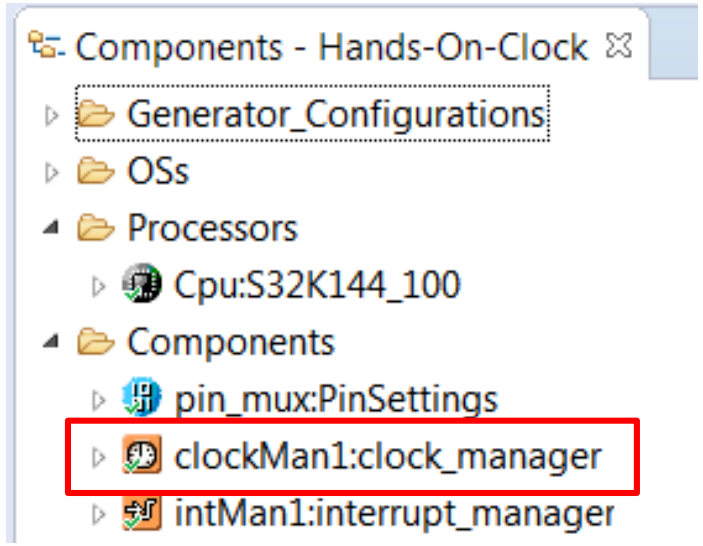
- Go to **Components Library** window
  进入Components Library视图
- Select the **clock_manager** in the **Alphabetical** Tab
  在Alphabetical标签栏选择Clock Manager
- Double click **clock_manager** to add to your project.
  双击Clock_Manager添加进当前工程
- Clock component should appear on the component window.
  Clock部件即显示在Component窗口中
- This component is added by default to your project.
  Clock_Manager默认已添加进工程中





S32K144 Clocks Lab: Enable Clock

- Select the **clock_manager** component in the **Components** window



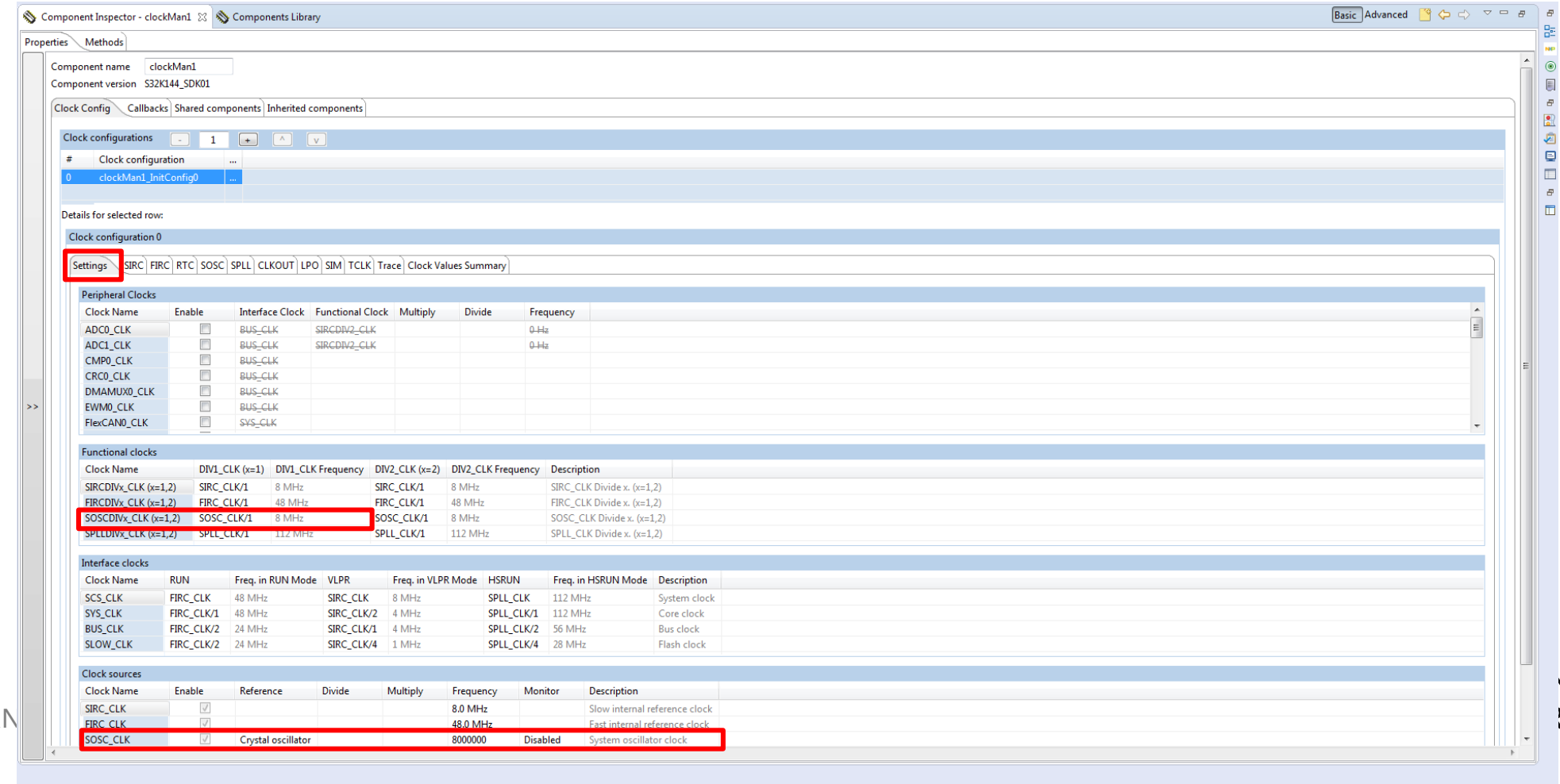
S32K144 Clocks Lab: Select SOSC configuration / 配置SOSC

- In the **Clock sources -> SOSC_CLK->Frequency** field write 8000000(corresponding to 8MHz)

  在Clock Sources区域中，填写SOCC_CLK的频率8000000（即8MHz）

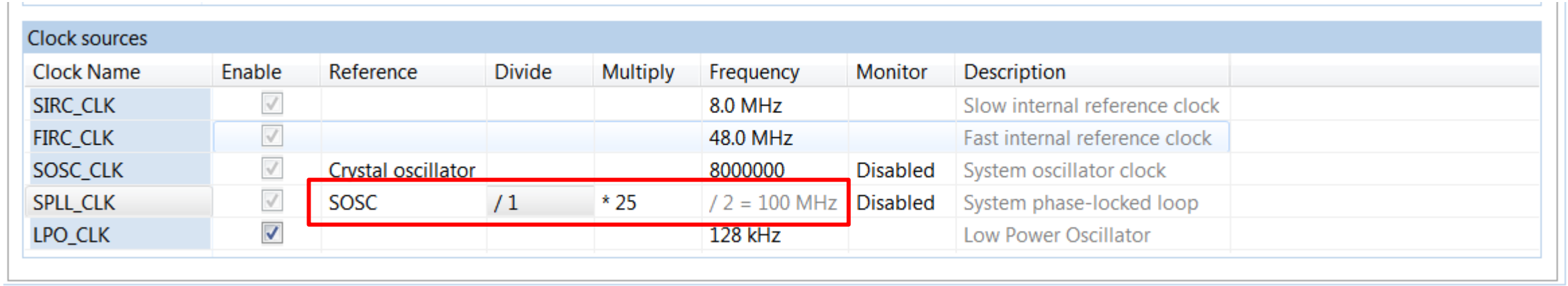
- In the **Functional Clock-> SOSCDIVx_CLK->DIV1_CLK** select SOSC_CLK/1

  在Functional Clock -> SOSCDIVx_CLK -> DIV1_CLK 选择SOSC_CLK/1



S32K144 Clocks Lab: Select PLL configuration / 配置PLL

- In the **Clock sources -> SPLL_CLK->Reference** select SOSC
  设置PLL的时钟源：SOSC
- In the **Clock sources -> SPLL_CLK->Divide** select /1
  设置分频系数：/1
- In the **Clock sources -> SPLL_CLK->Multiplicity** field select *25
  设置倍频系数：x25



Clock sources

Clock Name	Enable	Reference	Divide	Multiply	Frequency	Monitor	Description
SIRC_CLK	☑				8.0 MHz		Slow internal reference clock
FIRC_CLK	☑				48.0 MHz		Fast internal reference clock
SOSC_CLK	☑	Crystal oscillator			8000000	Disabled	System oscillator clock
SPLL_CLK	☑	SOSC	/ 1	* 25	/ 2 = 100 MHz	Disabled	System phase-locked loop
LPO_CLK	☑				128 kHz		Low Power Oscillator

S32K144 Clocks Lab: Select PLL configuration / 配置PLL

- In the **Functional clocks->SPLLDIVx_CLK->DIV1_CLK** select SPLL_CLK/1.
  设置SPLLDIV1_CLK的分频系数
- In the **Functional clocks->SPLLDIVx_CLK->DIV2_CLK** select SPLL_CLK/2.
  设置SPLLDIV2_CLK的分频系数

Functional clocks					
Clock Name	DIV1_CLK (x=1)	DIV1_CLK Frequency	DIV2_CLK (x=2)	DIV2_CLK Frequency	Description
SIRCDIVx_CLK (x=1,2)	SIRC_CLK/1	8 MHz	SIRC_CLK/1	8 MHz	SIRC_CLK Divide x. (x=1,2)
FIRCDIVx_CLK (x=1,2)	FIRC_CLK/1	48 MHz	FIRC_CLK/1	48 MHz	FIRC_CLK Divide x. (x=1,2)
SOSCDIVx_CLK (x=1,2)	SOSC_CLK/1	8 MHz	SOSC_CLK/1	8 MHz	SOSC_CLK Divide x. (x=1,2)
SPLLDIVx_CLK (x=1,2)	SPLL_CLK/1	100 MHz	SPLL_CLK/2	50 MHz	SPLL_CLK Divide x. (x=1,2)

S32K144 Clocks Lab: Select PLL configuration / 配置PLL

- In the **Interface clocks section** select the following RUN configuration**:**
  对RUN模式进行时钟配置
  - **System Clock Source:** SPLL
  - **Core Clock Divide Ratio:** SPLL_CLK/2
  - **Platform Clock Divide Ratio:** SPLL_CLK/2
  - **Bus Clock Divide Ratio:** SPLL_CLK/2
  - **Slow Clock(Flash Clock) Divide Ratio:** SPLL_CLK/2

Interface clocks

Clock Name	RUN	Freq. in RUN Mode	VLPR	Freq. in VLPR Mode	HSRUN	Freq. in HSRUN Mode	Description
SCS_CLK	SPLL_CLK	100 MHz	SIRC_CLK	8 MHz	SPLL_CLK	100 MHz	System clock
SYS_CLK	SPLL_CLK/2	50 MHz	SIRC_CLK/2	4 MHz	SPLL_CLK/1	100 MHz	Core clock
BUS_CLK	SPLL_CLK/2	25 MHz	SIRC_CLK/1	4 MHz	SPLL_CLK/2	50 MHz	Bus clock
SLOW_CLK	SPLL_CLK/2	25 MHz	SIRC_CLK/4	1 MHz	SPLL_CLK/4	25 MHz	Flash clock

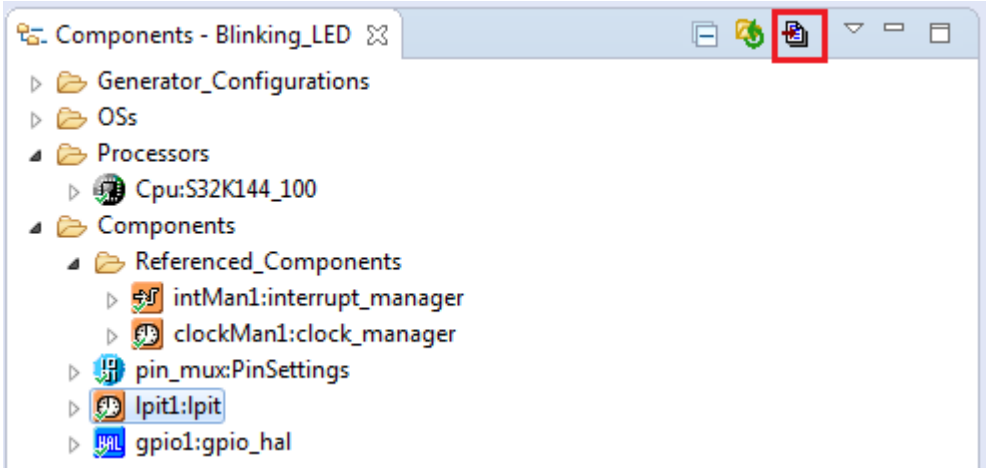
S32K144 Clock Lab: Generate the code / 代码生成

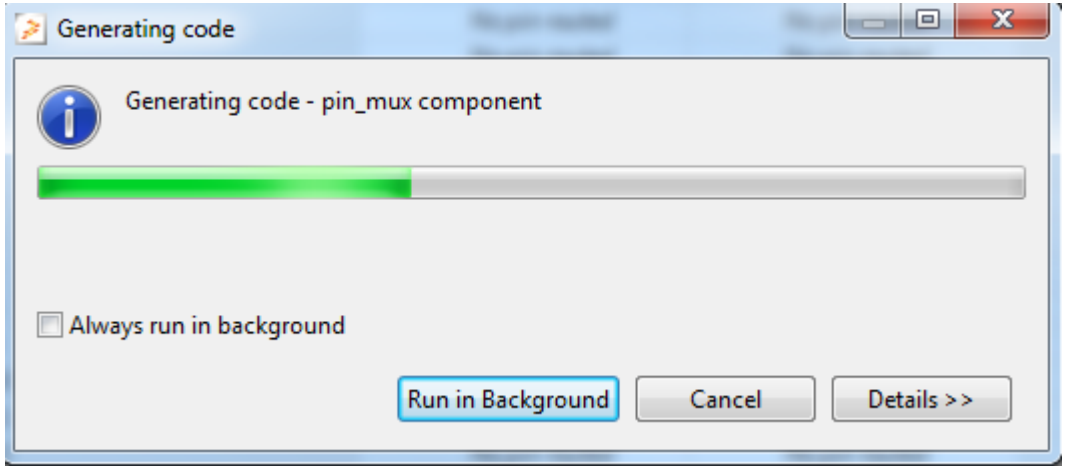
- To generate the code for the configuration select, click the **generate code** icon in the **Components** window.

  配置代码生成，点击Generate Code图标

- Wait for the code to be generated.

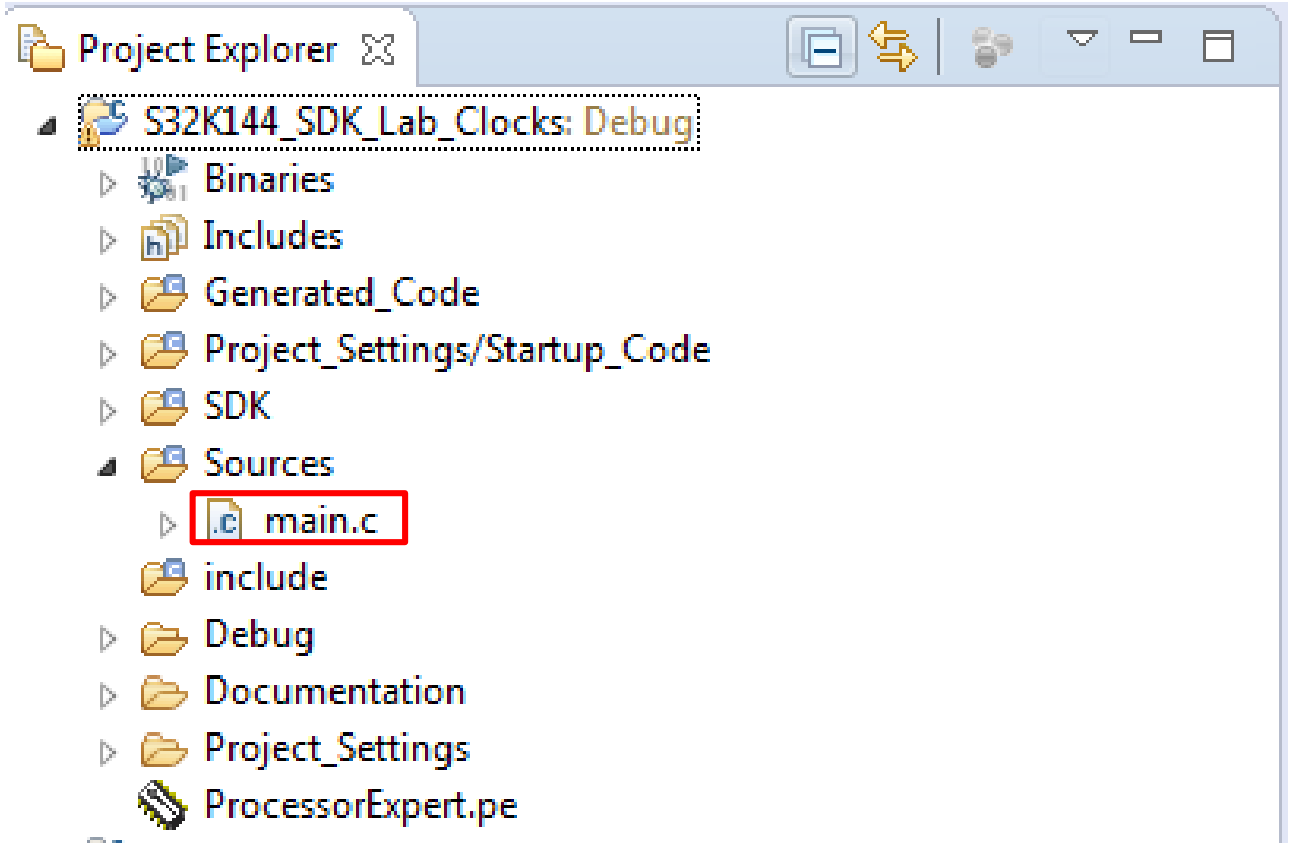
  等待进度条完成





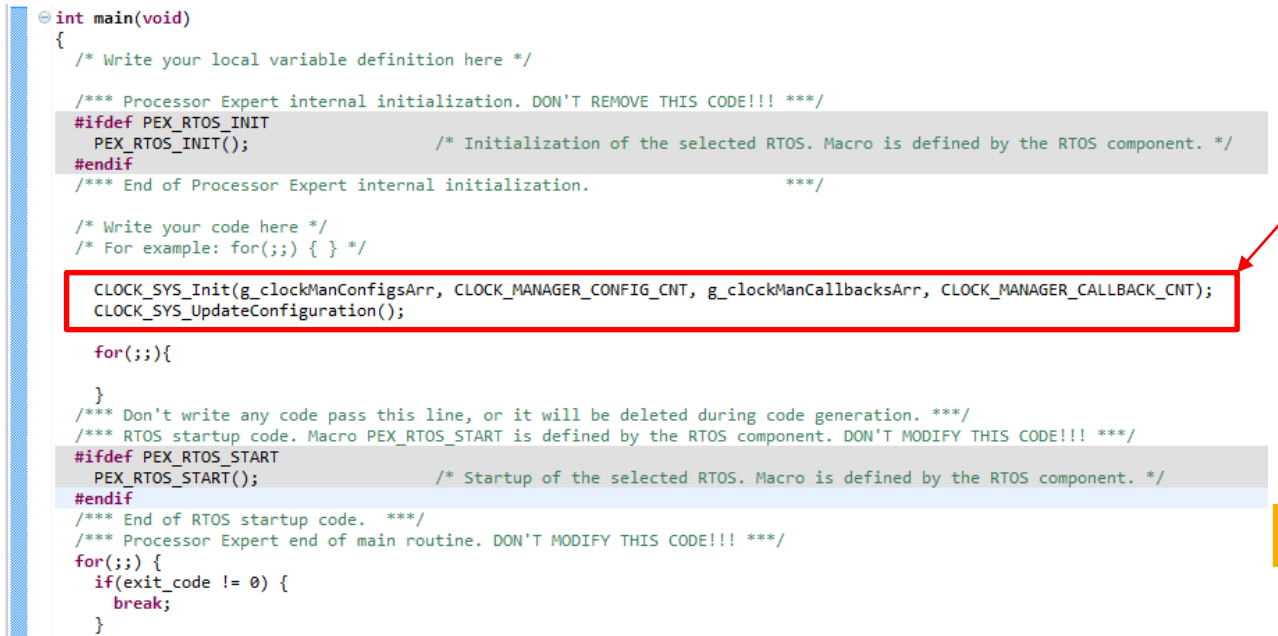
S32K144 GPIOs Lab: Open the main.c / 打开main.c文件

- In the project window double click the main.c file to open it
  在工程窗口，双击main.c文件打开

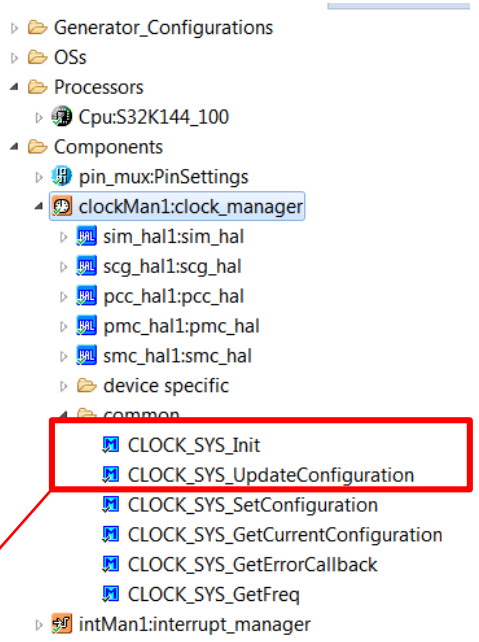


S32K144 Clocks Lab: Add Init and Update Configuration functions / 添加初始化和配置更新函数

- Expand the **clock_manager** component in the **Components** Window
  展开Clock_Manager部件
- Drag and drop the **CLOCK_SYS_Init** function into main.
  拖拽CLOCK_SYS_Init函数到Main函数中
- Drag and drop the **CLOCK_SYS_UpdateConfiguration** function into main.
  拖拽CLOCK_SYS_UpdateConfiguration函数到Main函数中
- Include an infinite loop after these functions. 添加一个主循环



```
int main(void)
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  #ifdef PEX_RTOS_INIT
    PEX_RTOS_INIT();                 /* Initialization of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */

    CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT, g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration();

    for(;;){

    }
  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;) {
    if(exit_code != 0) {
      break;
    }
```



S32K144 Clocks Lab: Add Init and Update Configuration functions / 添加初始化和配置更新函数

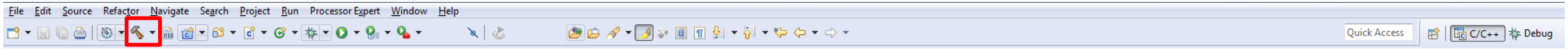
- In the **CLOCK_SYS_Init** function add the following parameters.
  填写CLOCK_SYS_Init函数的参数
  - g_clockManConfigsArr,
  - CLOCK_MANAGER_CONFIG_CNT,
  - g_clockManCallbacksArr,
  - CLOCK_MANAGER_CALLBACK_CNT
- In the **CLOCK_SYS_UpdateConfiguration** add the following parameters.
  填写CLOCK_SYS_UpdateConfiguratin函数的参数
  - 0U,
  - CLOCK_MANAGER_POLICY_AGREEMENT

```
CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                          g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_AGREEMENT);
```

S32K144 Clocks Lab: Build and debug the lab / 编译调试工程

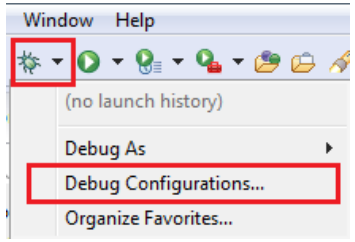
- Click on the build icon to make sure that there a no compiler errors.

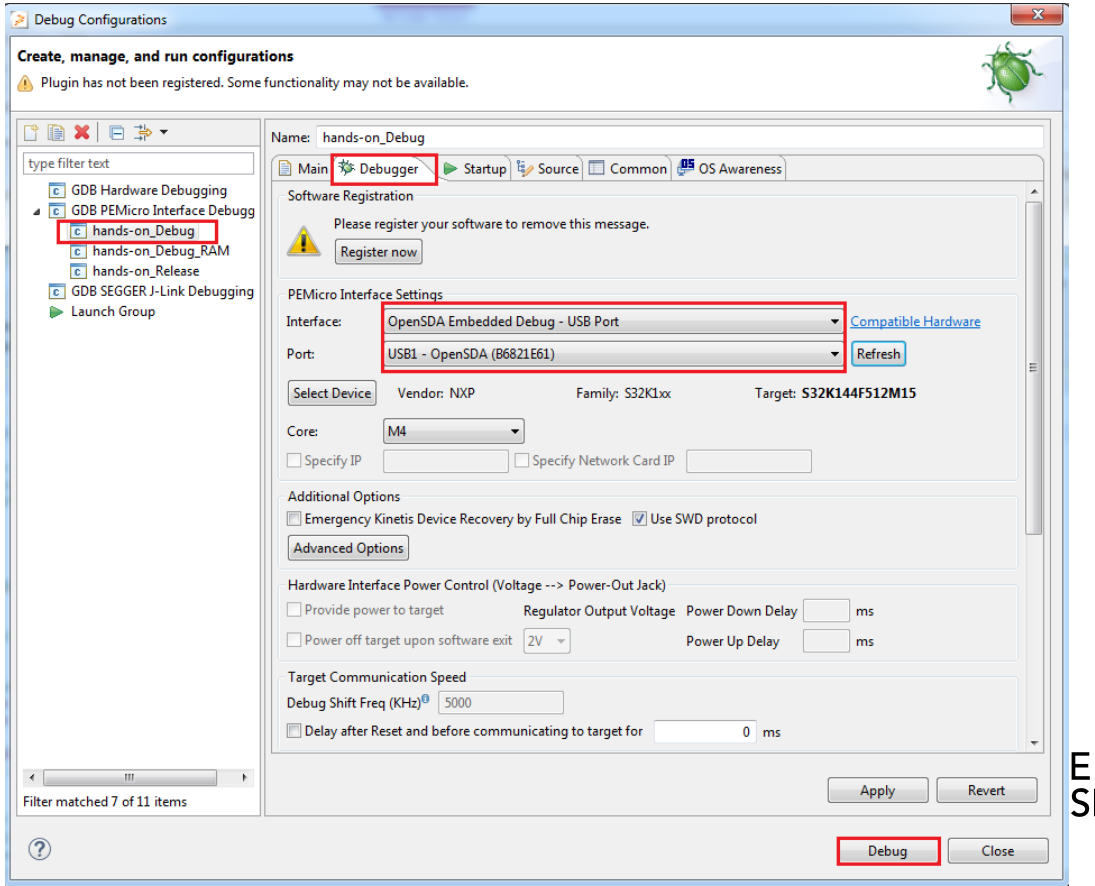
  编译工程，确认没有报错



- Configure the debug configuration start a new debug session

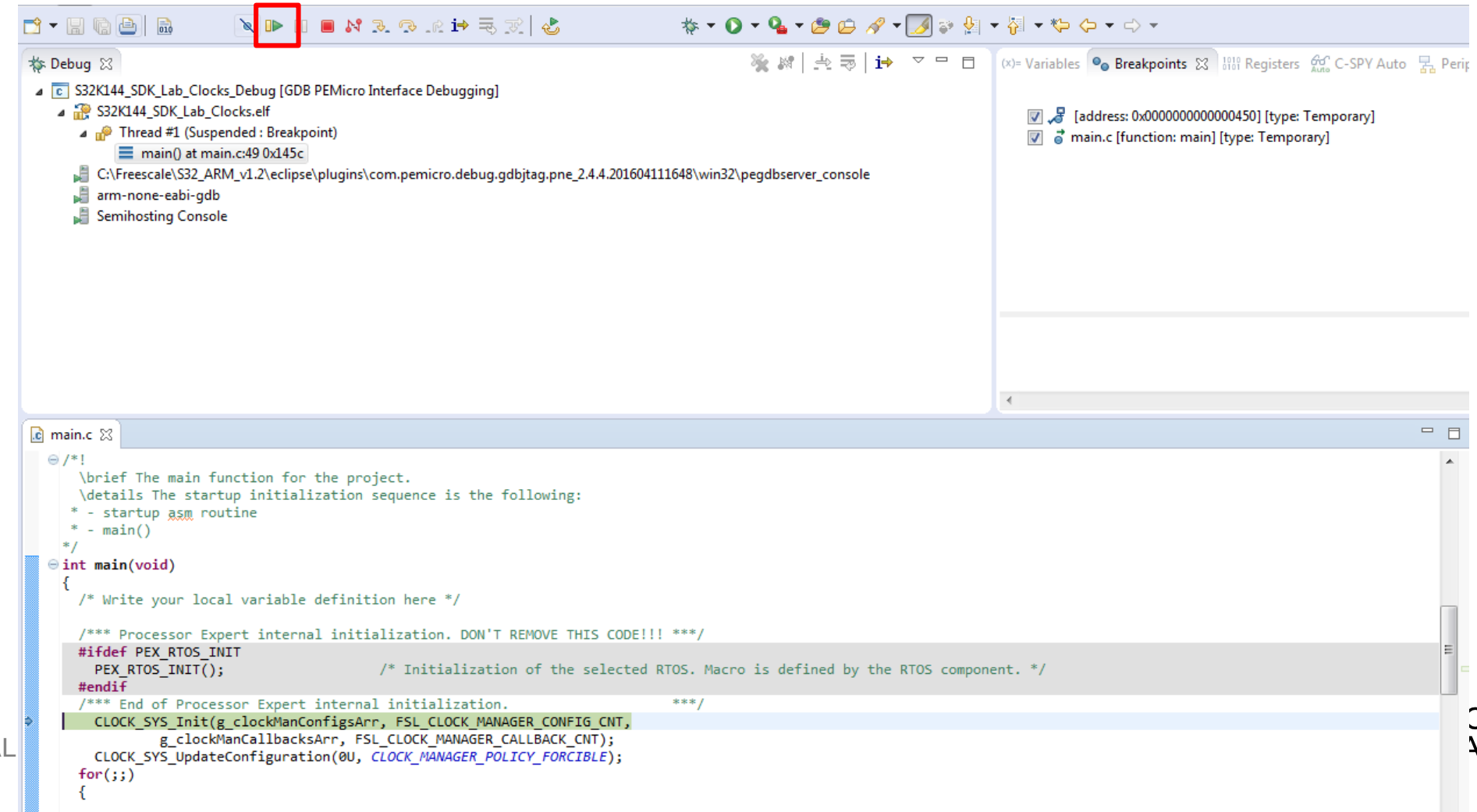
  配置调试口，开始调试





S32K144 Clocks Lab: Build and debug the lab / 编译调试工程

- In the debug perspective click the run icon to start the project.
  在调试界面，点击全速运行图标



Lab 02.

S32K144 GPIOS

- In this lab you will learn:

  本实验你将学习以下内容

  - About the GPIOs structure in S32K144

    S32K144 GPIOs结构

  - How to create a new SDK project with S32DS.

    使用S32DS新建SDK工程

  - How to set a pin as output/input with SDK

    如何使用SDK配置管脚成输入或输出

  - Configure GPIOs to turn ON Red LED every time SW2 is pressed

    配置GPIOs，在每次SW2按下时点亮绿色LED灯

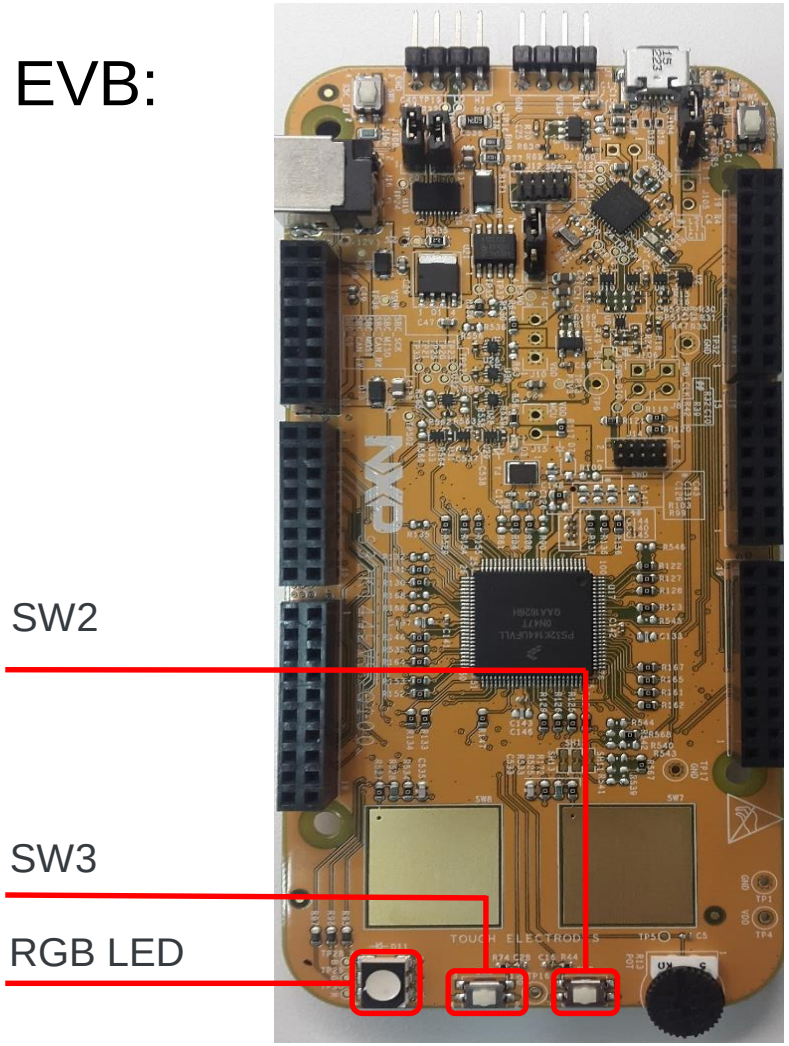
S32K144 GPIOs Lab: Resources to be used

- In this lab will be used the following components of the EVB:
  本实验中，你将使用以下EVB资源
  - RGB LED 绿色LED

LED	S32K144 PIN	Pull resistor
BLUE	PTD0	Pull up
RED	PTD15	Pull up
GREEN	PTD16	Pull up

  - SW2 and SW3 按键2和按键3

Button	S32K144 PIN	Pull resistor
SW2	PTC12	Pull down
SW3	PTC13	Pull down



S32K144 GPIOs Lab: Theory

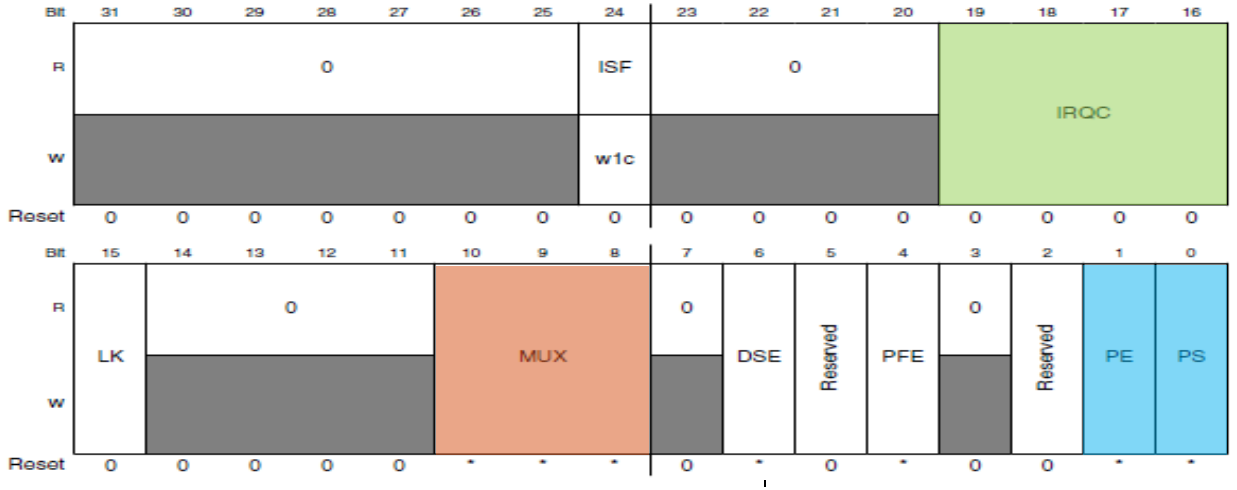
- There are up to 89 GPIOs in the S32K144
  S32K144芯片上最多支持89个GPIO
  – 5 PORTs (PTA, PTB, PTC, PTD, PTE)
- 11 high current pins (up to 20 mA each)
  11个大电流管脚（最高支持20mA）
- Each I/O is interrupt capable
  每个I/O管脚均可产生中断
- Each I/O is DMA capable
  每个I/O管脚均可产生DMA请求
- Support for edge or level sensitive
  支持边沿或电平敏感
- Each can wake up MCU from low power modes
  每个管脚均可从低功耗模式唤醒MCU
- Digital filter included for each I/O
  每个I/O管脚包含数字滤波器

Package	GPIOs	High current pins
100 LQFP/64 LQFP/ BGA	89/58	11 - PTD1 - PTD0 - PTD16 - PTD15 - PTB6 - PTB5 - PTB4 - PTE4 - PTE1 - PTE0 - PTA10

S32K144 GPIOs Lab: Theory

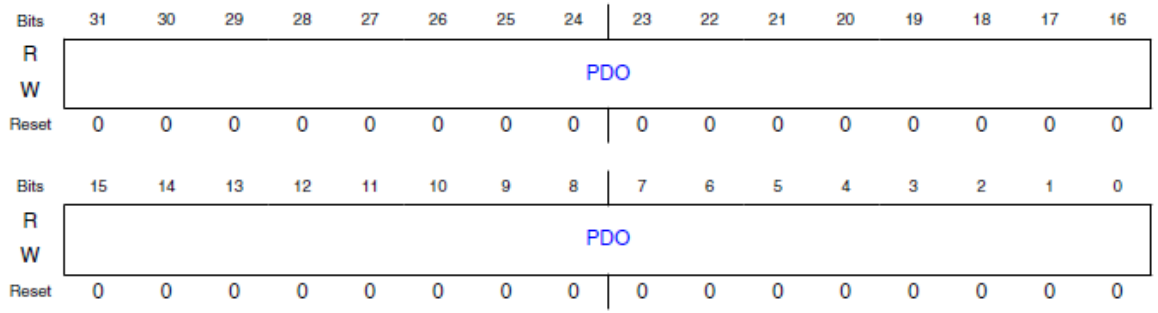
- Each I/O is multiplexed with different functionalities
  每个I/O可复用成多种功能
- I/O functionality is selected with PORTx register, MUX bits.
  I/O功能通过PORTx寄存器MUX比特域配置
  - Alternative 1 (MUX=0b001) is GPIO functionality for all I/OS
    所有I/O的GPIO功能都是Alternative1选项
- I/O interrupt configuration (IRQC bit flied) is controlled independently
  单独控制各个I/O的中断
- I/O Pull resistor (PE bit and PS bit) is controlled independently
  单独控制各个I/O的上下拉电阻

Pin Control Register n (PORT_PCR*n*)



S32K144 GPIOs Lab: Theory

- Each port pin is mapped to the following 32-bit GPIO registers, each bit represents a pin in the port x:
  每个端口均映射成32-bit GPIO寄存器，每比特位对应端口中的一个管脚



GPIO Register	Description
GPIOx->PDOR	Port Data Output Register
GPIOx->PSOR	Port Set Output Register
GPIOx->PCOR	Port Clear Output Register
GPIOx->PTOR	Port Toggle Output Register
GPIOx->PDIR	Port Data Input register
GPIOx->PDDR	Port Data Direction register
GPIOx->PIDR	Port Input disable register

S32K144 GPIOs Lab: Theory

- GPIO Direction selected with PDDR register.

  PDDR寄存器选择GPIO输入或者输出
- GPIO INPUT 输入
  - Logic state available in PDIR register

    逻辑状态锁存在PDIR寄存器中
- GPIO OUTPUT 输出
  - Logic state controlled via PDOR or PCOR,PSOR and PTOR.

    通过PDOR, PCOR, PSOR和PTOR寄存器控制

If	Then
A pin is configured for the GPIO function and the corresponding port data direction register bit is clear.	The pin is configured as an input.
A pin is configured for the GPIO function and the corresponding port data direction register bit is set.	The pin is configured as an output and and the logic state of the pin is equal to the corresponding port data output register.

S32K144 GPIOs Lab: Pin Configuration / 配置管脚

- Create a new S32DS Project

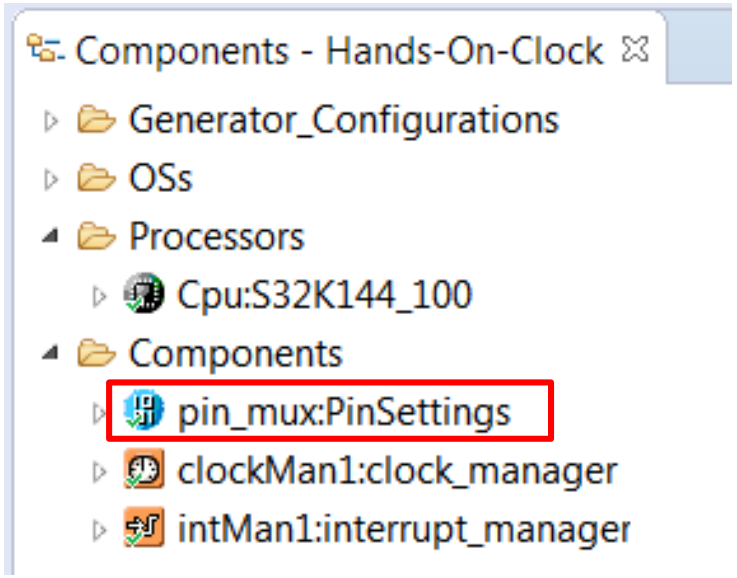
  新建S32DS工程
- Setup the clocks of the S32K144

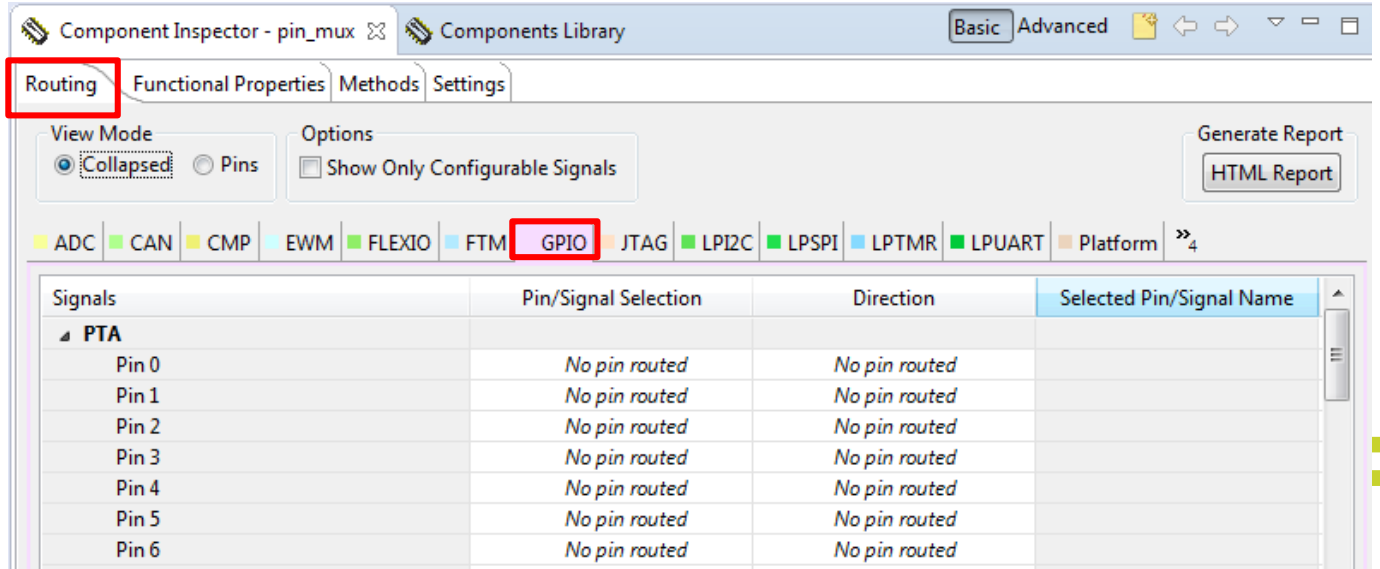
  设置系统时钟
- Select the **pin_mux** component in the **Components** window

  选择Pin_Mux部件

  – Select **GPIO** tab inside the **Routing** to select pin direction

  GPIO标签栏下，点击管脚的Routing下拉菜单，选择管方向

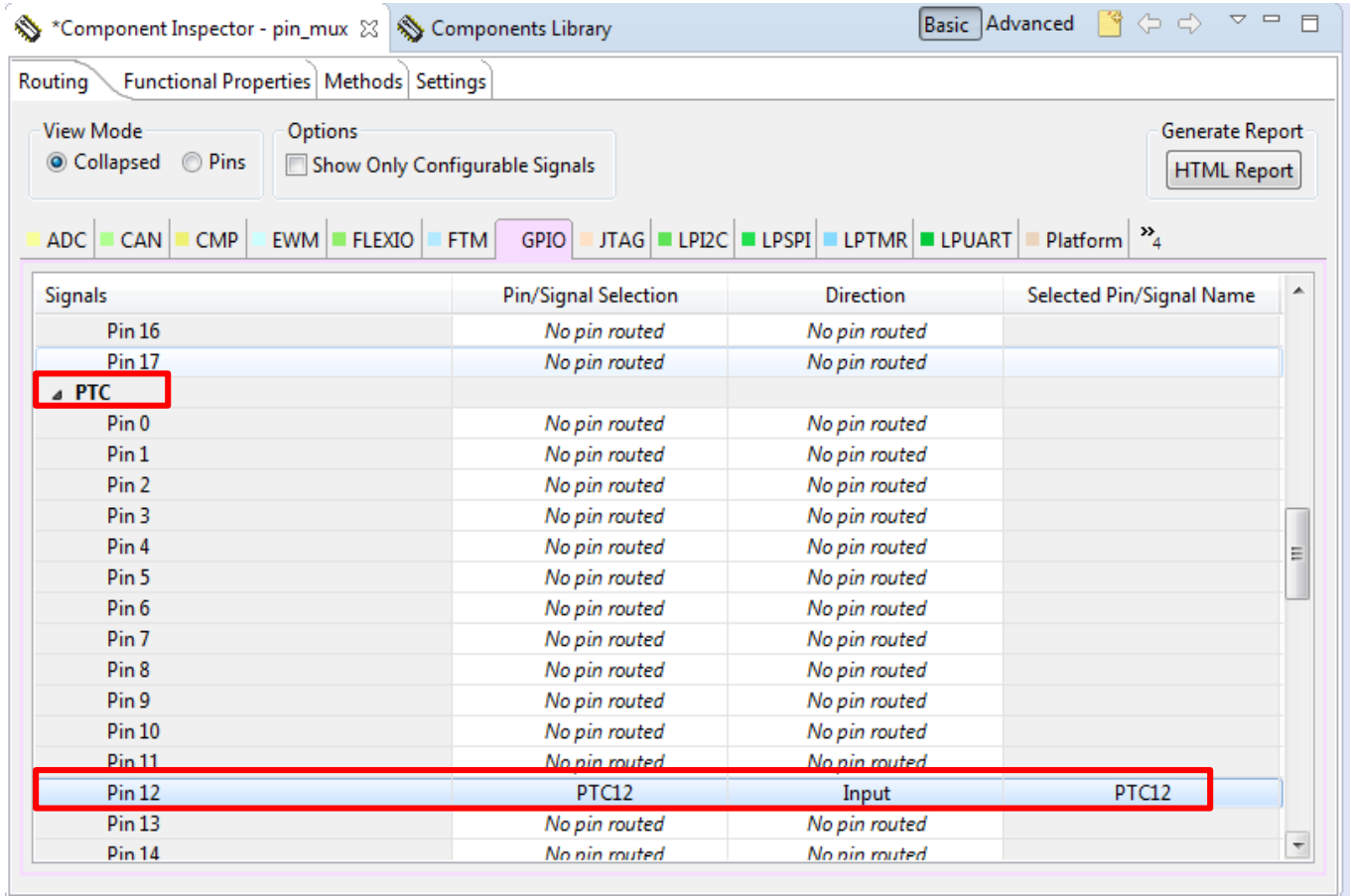




S32K144 GPIOs Lab: Select Input pin / 设置输入管脚

- Go to **PTC** and select pin **12**.
- In the **Pin/Signal Selection** Colum, select **PTC12**.
- In the **Direction** Colum, select **Input.**

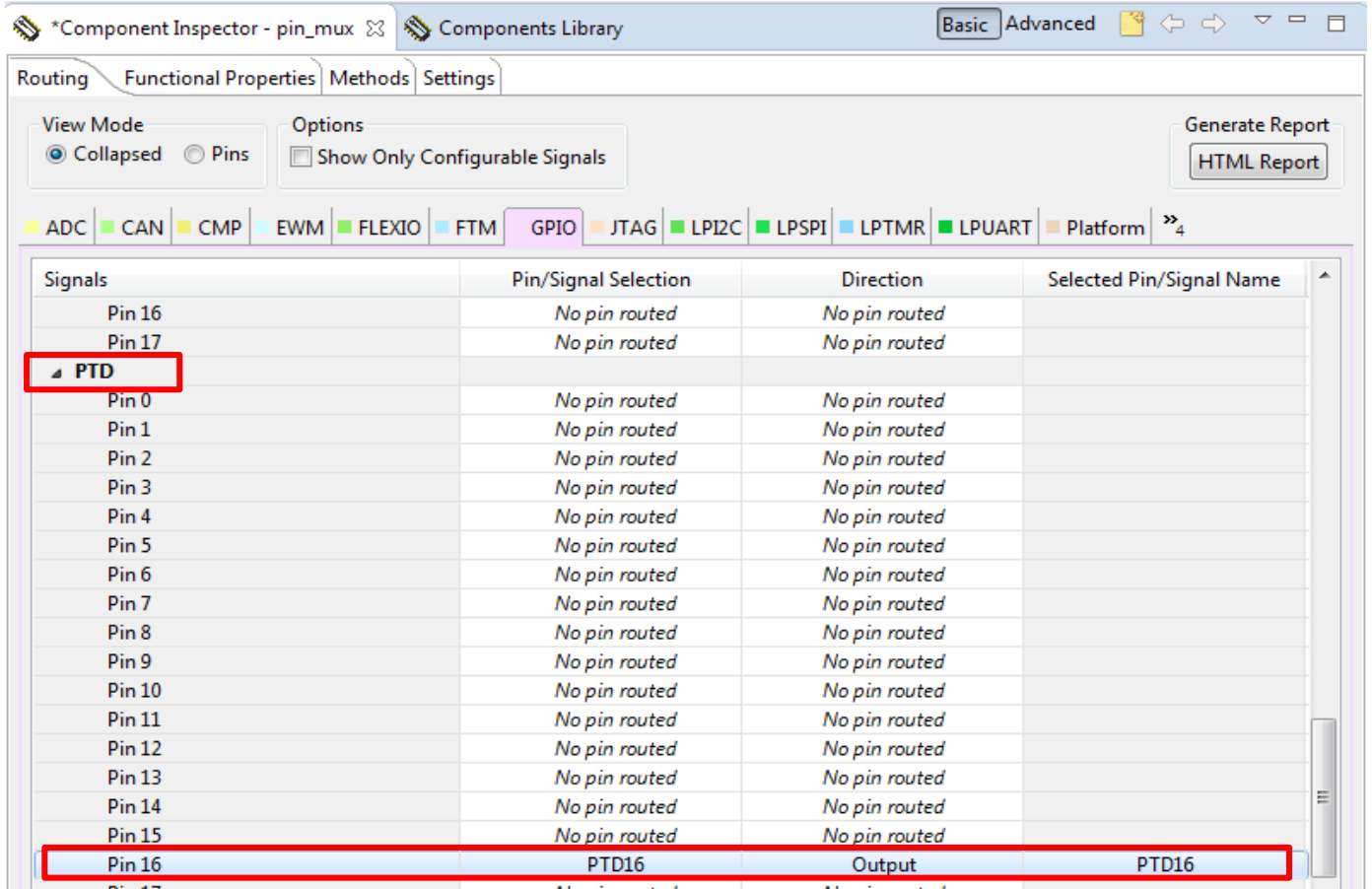
在PTC（端口C）下，选择管脚12，设置为Input



S32K144 GPIOs Lab: Select Output pin / 设置输出管脚

- Go to **PTD** and select pin **16**.
- In the **Pin/Signal Selection** Colum, select **PTD16**.
- In the **Direction** Colum, select **Output.**

在PTD（端口D）下，选择管脚16，设置为Output



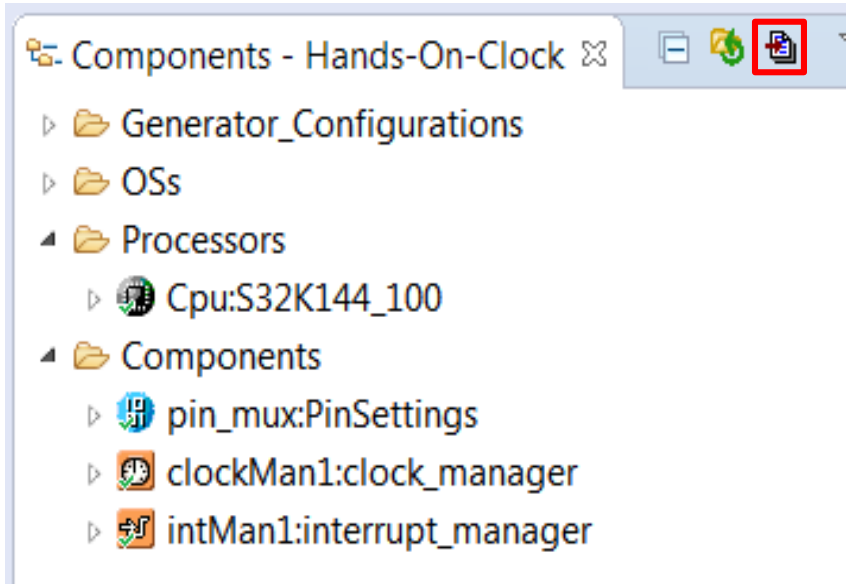
S32K144 GPIOs Lab: Generate the code / 代码生成

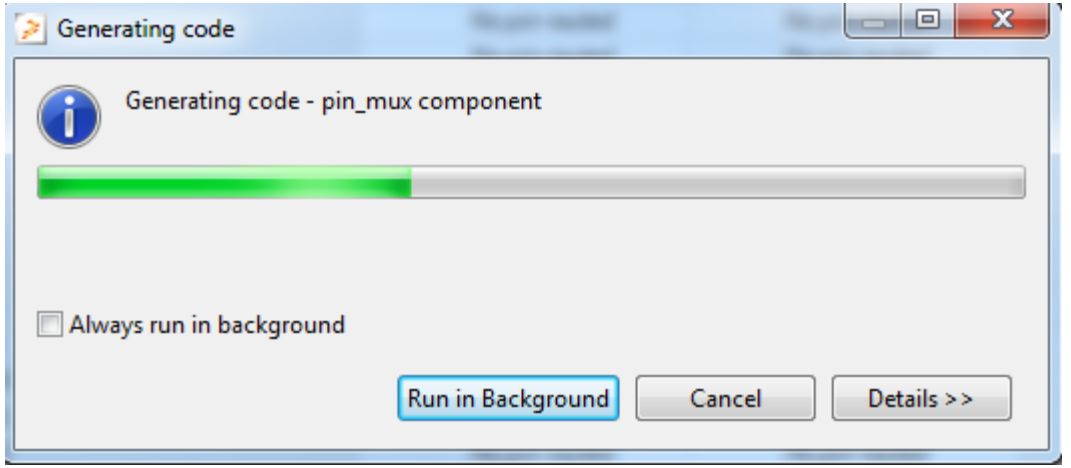
- To generate the code for the configuration select, click the **generate code** icon in the **Components** window.

  配置代码生成，点击Generate Code图标

- Wait for the code to be generated.

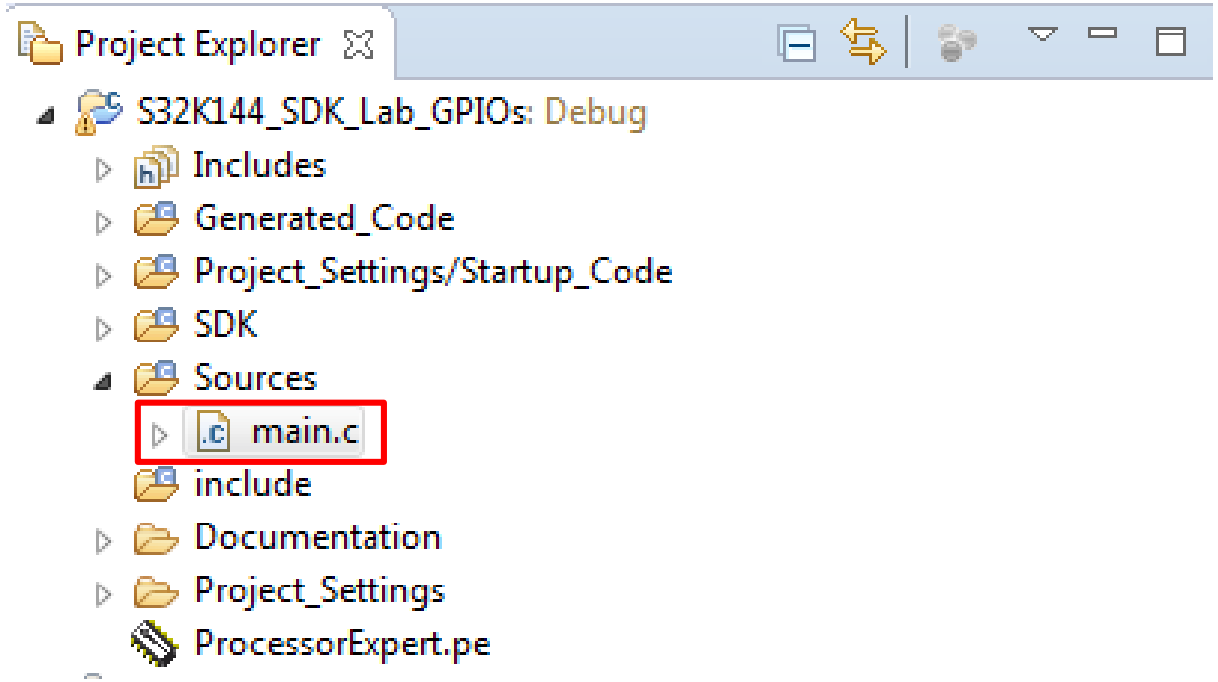
  等待进度条完成





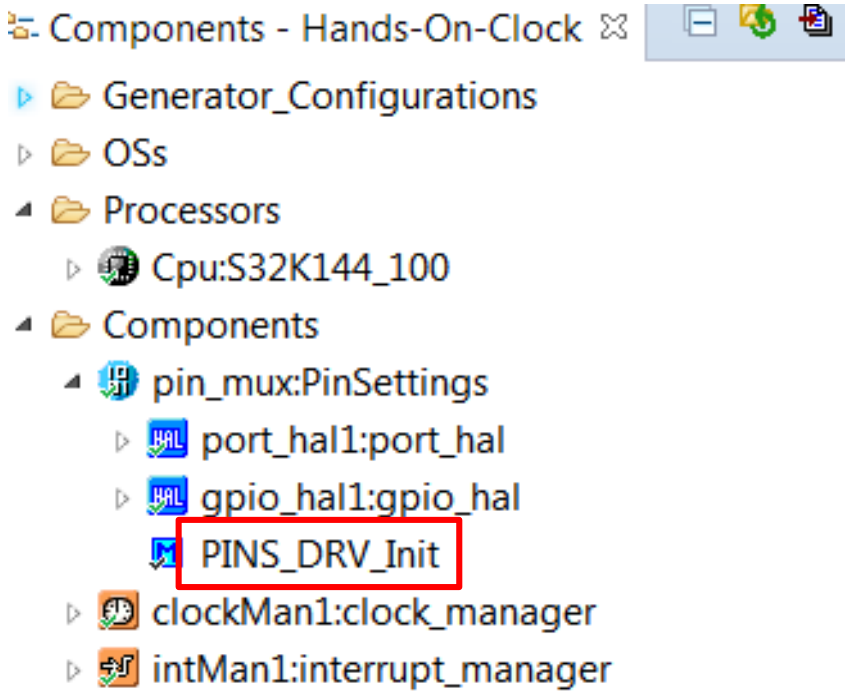
S32K144 GPIOs Lab: Open the main.c / 打开main.c文件

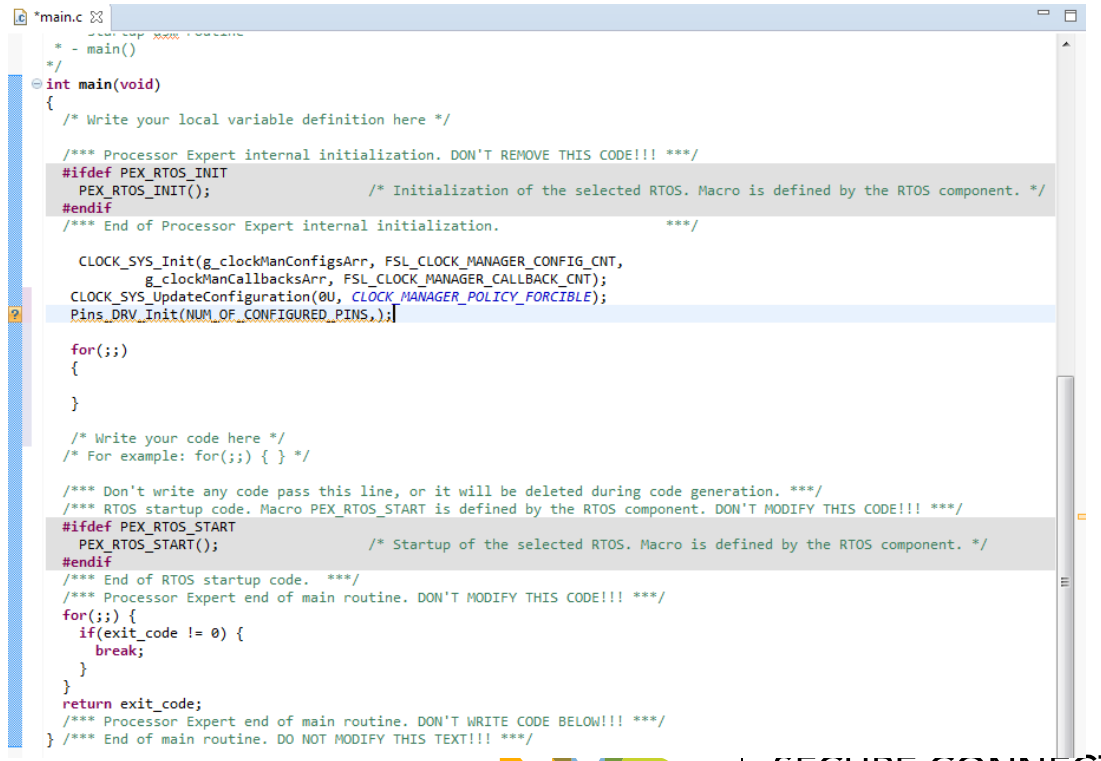
- In the project window double click the main.c file to open it
  在工程窗口，双击main.c文件打开



S32K144 GPIOs Lab: Initialize pins / 初始化管脚

- Expand the **pin_mux** component in the **Components** Window.
  展开Pin_Mux部件
- Drag and drop the **Pins_DRV_Init** function inside the, into main, below the clock configuration
  拖拽Pins_DRV_Init函数，至Main函数中，放在配置时钟函数之后





S32K144 GPIOs Lab: Initialize pins

- **Pins_DRV_Init** function receives two parameters:

  Pins_DRV_Init函数有两个参数

  – Number of pins to configure 已配置的管脚个数

  – Configuration structure 配置结构体
- The number of pins to configure is included by default

  参数“已配置的管脚个数” 默认已填好
- The configuration structure is already created, with the name **g_pin_mux_InitConfigArr**

  参数“配置结构体”已生成，名为“g_pin_mux_InitConfigArr”
- Add the configuration structure into the **Pins_DRV_Init** function

  将结构体放入Pins_DRV_Init第二个参数位置

```
Pins_DRV_Init(NUM_OF_CONFIGURED_PINS,g_pin_mux_InitConfigArr);
```

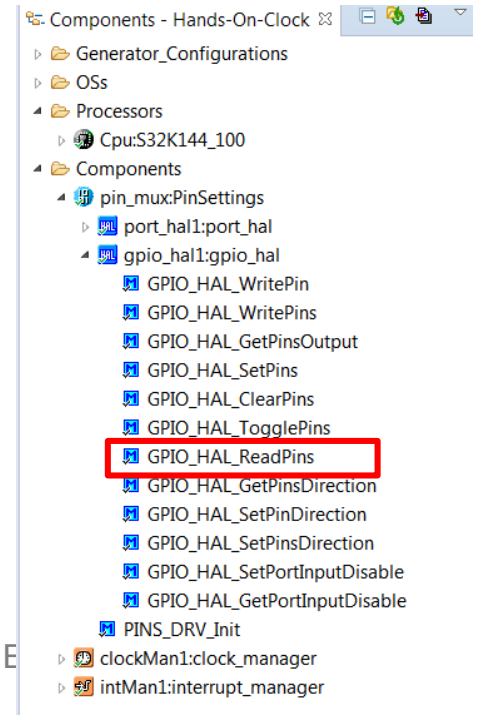
S32K144 GPIOs Lab: Read SW2(PTC12) input / 读出按键SW2的值

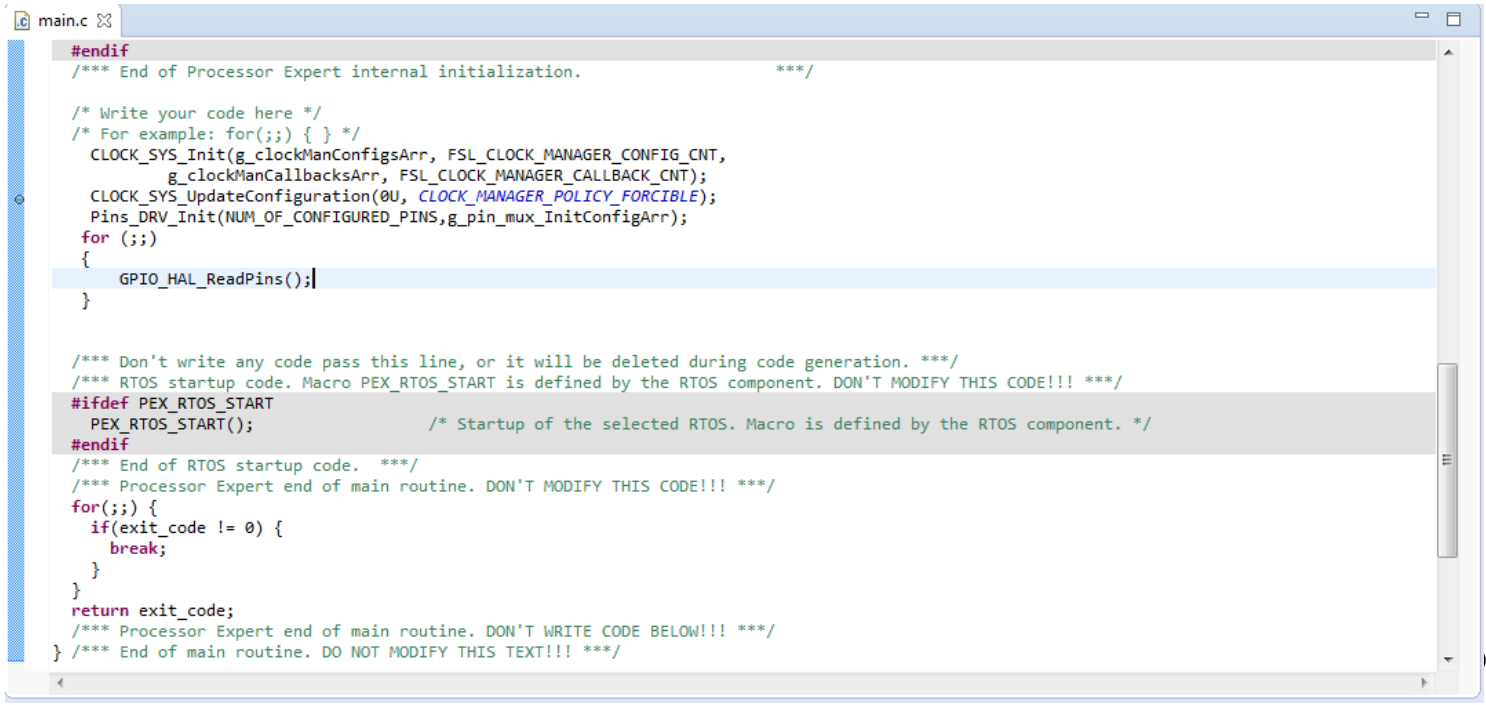
- Expand the **GPIO HAL** component in the **Components** Window

  展开GPIO_HAL1子部件

- Drag and drop the **GPIO_HAL_ReadPins** function in to main, and place it inside an infinite loop.

  拖拽GPIO_HAL_ReadPins函数至主函数，放在主循环里





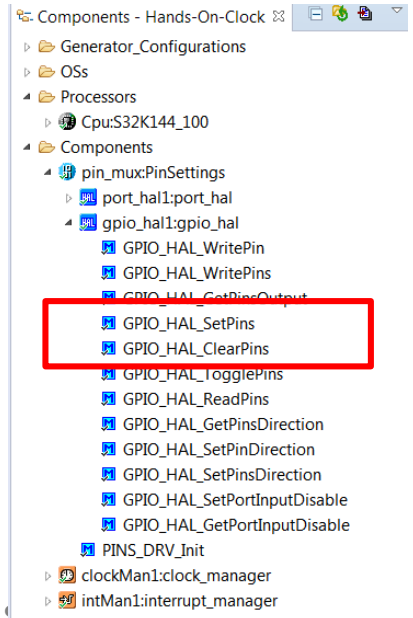
S32K144 GPIOs Lab: Read SW2(PTC12) input

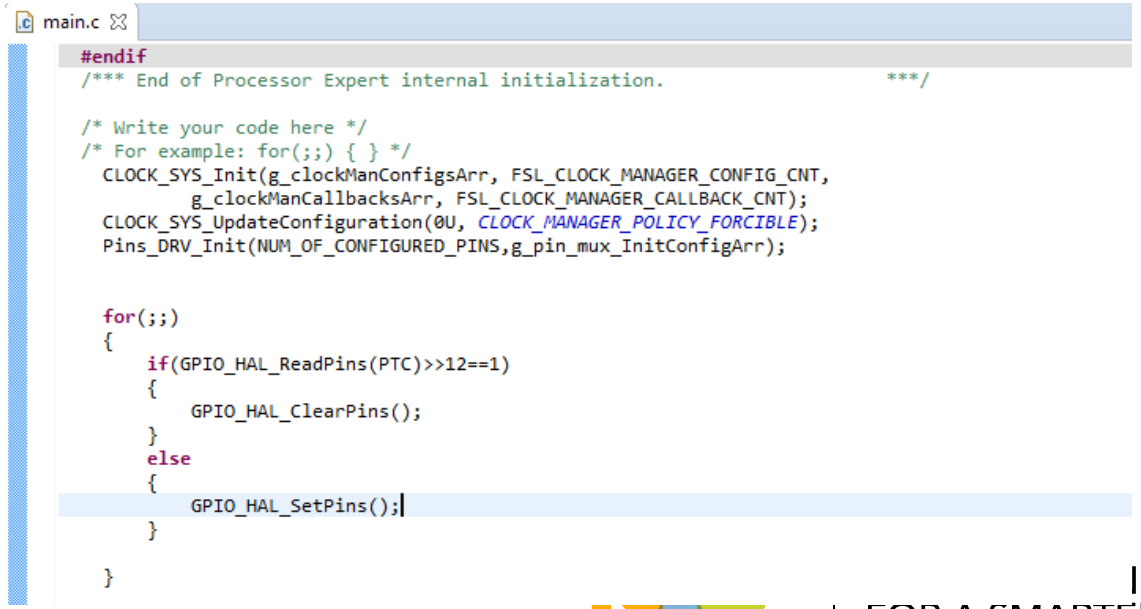
- **GPIO_HAL_ReadPins** function receives one parameter:
  GPIO_HAL_ReadPins函数需要一个参数，即端口
  - PORTx to read
- **GPIO_HAL_ReadPins** returns the value of the PIDR of PORTx
  GPIO_HAL_ReadPins返回对应Port的PIDR寄存器的值
- Use an if statement as follows to read SW2(PTC12)
  使用if-else方式读按键SW2的值

```
for(;;)
{
    if(GPIO_HAL_ReadPins(PTC)>>12==1)
    {

    }
    else
    {

    }

}
```

S32K144 GPIOs Lab: Turn on Green LED (PTD16)

- Expand the **GPIO HAL** component in the **Components** Window
  展开GPIO_HAL1子部件
- Drag and drop the **GPIO_HAL_ClearPins** function in to main, an place it inside the if statement
  拖拽GPIO_HAL_ClearPins至主函数，放在if分支
- Drag and drop the **GPIO_HAL_SetPins** function in to main, an place it inside the else statement
  拖拽GPIO_HAL_SetPins至主函数，放在else分支





S32K144 GPIOs Lab: Turn on Green LED (PTD16) / 点亮绿色LED

- Both functions receive two parameters:

  这两个函数都需要以下两个参数

  – PORTx to modify 端口

  – Bit mask 比特位

- In order to turn on/off the green LED, include the following parameters into the Clear and Set functions

  将参数放入Clear和Set函数中，点亮/灭绿色LED

```
for(;;)
{
    /* If button pressed*/
    if(GPIO_HAL_ReadPins(PTC)>>12==1)
    {
        /* Turn ON green LED */
        GPIO_HAL_ClearPins(PTD,1<<16);
    }

    else
    {
        /* Turn OFF green LED */
        GPIO_HAL_SetPins(PTD,1<<16);
    }
}
```

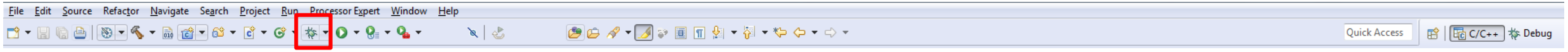
S32K144 GPIOs Lab: Build and debug the lab / 编译调试工程

- Click on the build icon to make sure that there a no compiler errors.

  编译工程，确认没有报错

- Configure the debug configuration start a new debug session

  配置调试口，开始调试

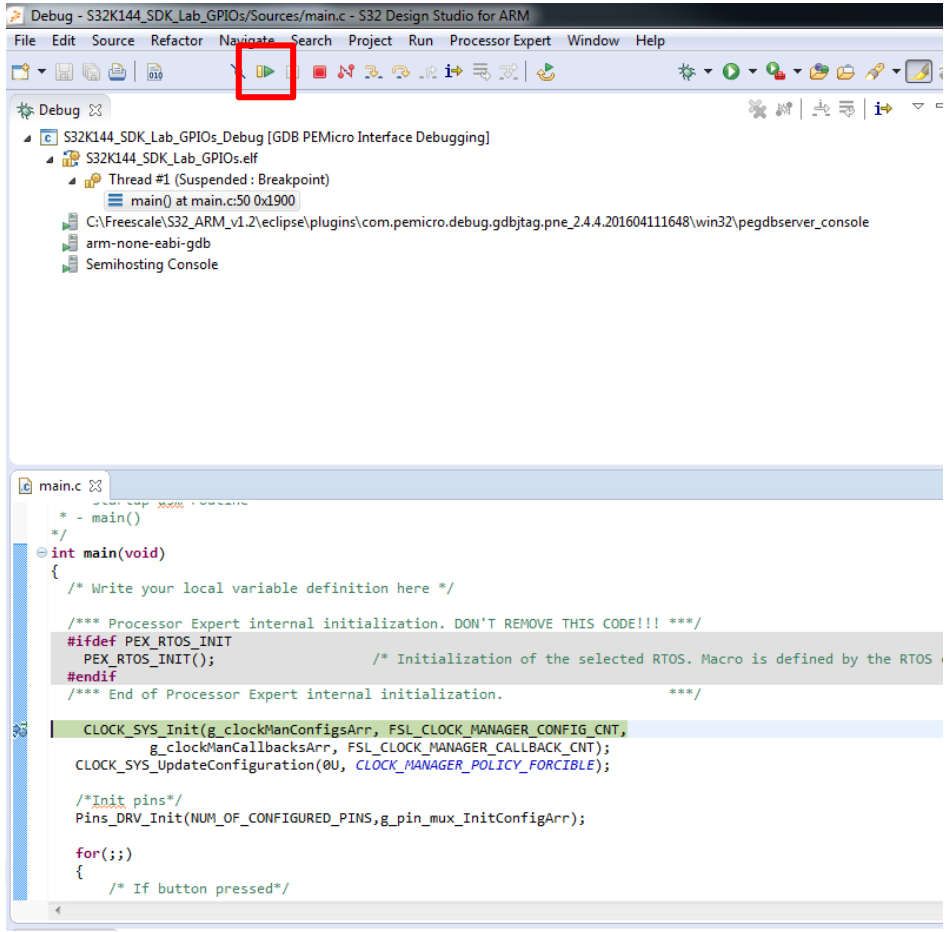


S32K144 GPIOs Lab: Build and debug the lab / 编译调试工程

- In the debug perspective click the run icon to start the project.

  Press SW2 and the Green LED should turn on.

  全速运行，按下SW2，绿色LED会亮起



Lab 03.

S32K144 INTERRUPTS

- In this lab you will learn:

  本实验你将学习以下内容

  - How interrupts works on S32K144

    S32K144中断机制

  - How the use the LPIT peripheral

    如何使用LPIT模块

  - Set up an interrupt in S32K144 using SDK

    使用SDK设置S32K144的中断

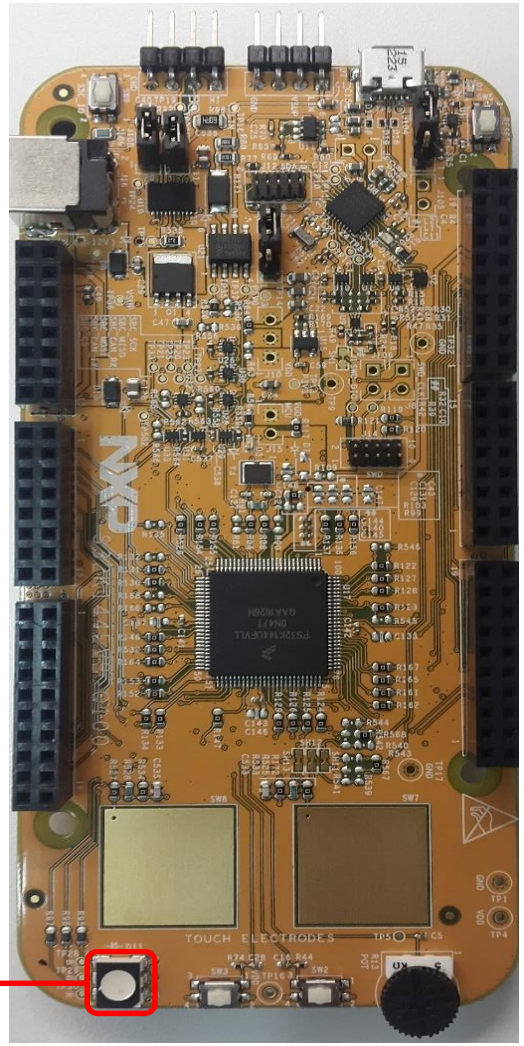
  - Blink an LED every 1 sec using the LPIT interrupt

    使用LPIT中断，每1秒钟闪烁LED

S32K144 Interrupts Lab: Resources to be used

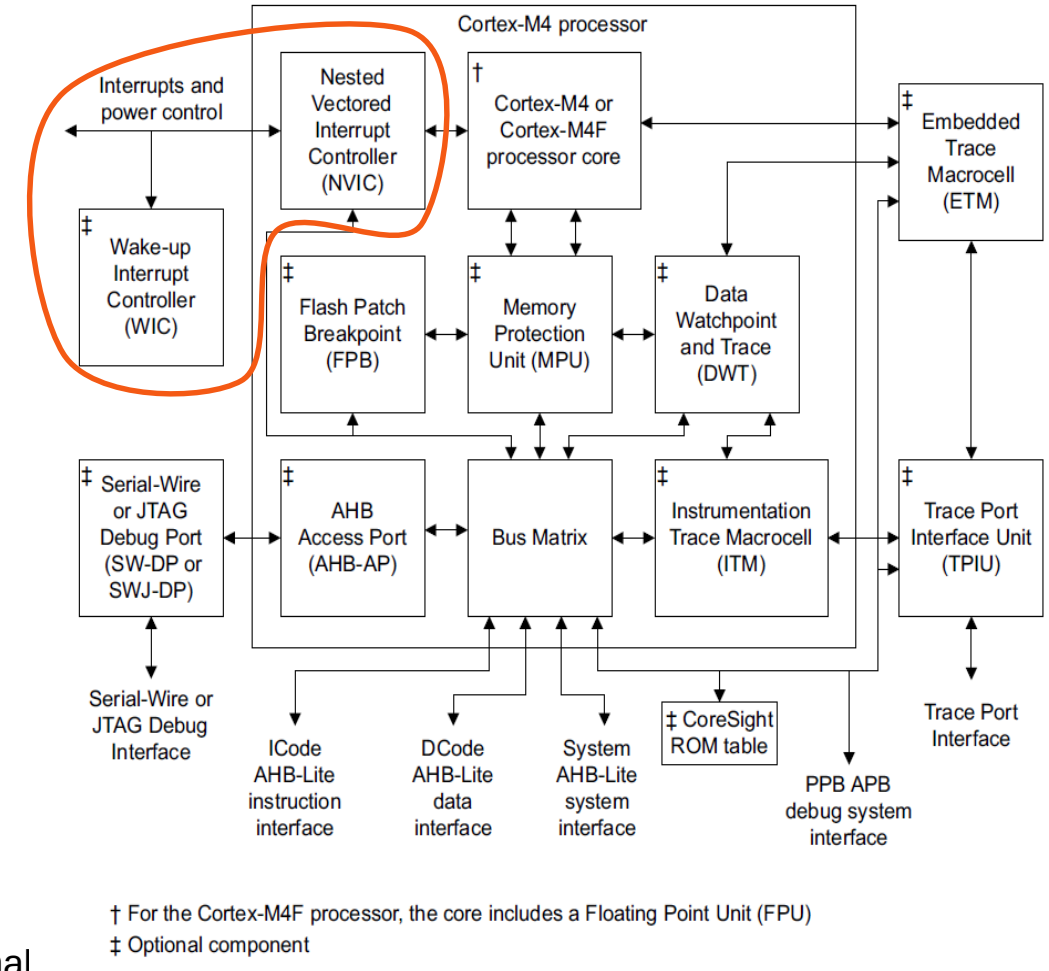
- In this lab will be used the following components of the EVB:
  本实验中，你将使用以下EVB资源
  - Green LED 绿色LED

LED	S32K144 PIN
BLUE	PTD0
RED	PTD15
GREEN	PTD16



S32K144 Interrupts Lab: Theory

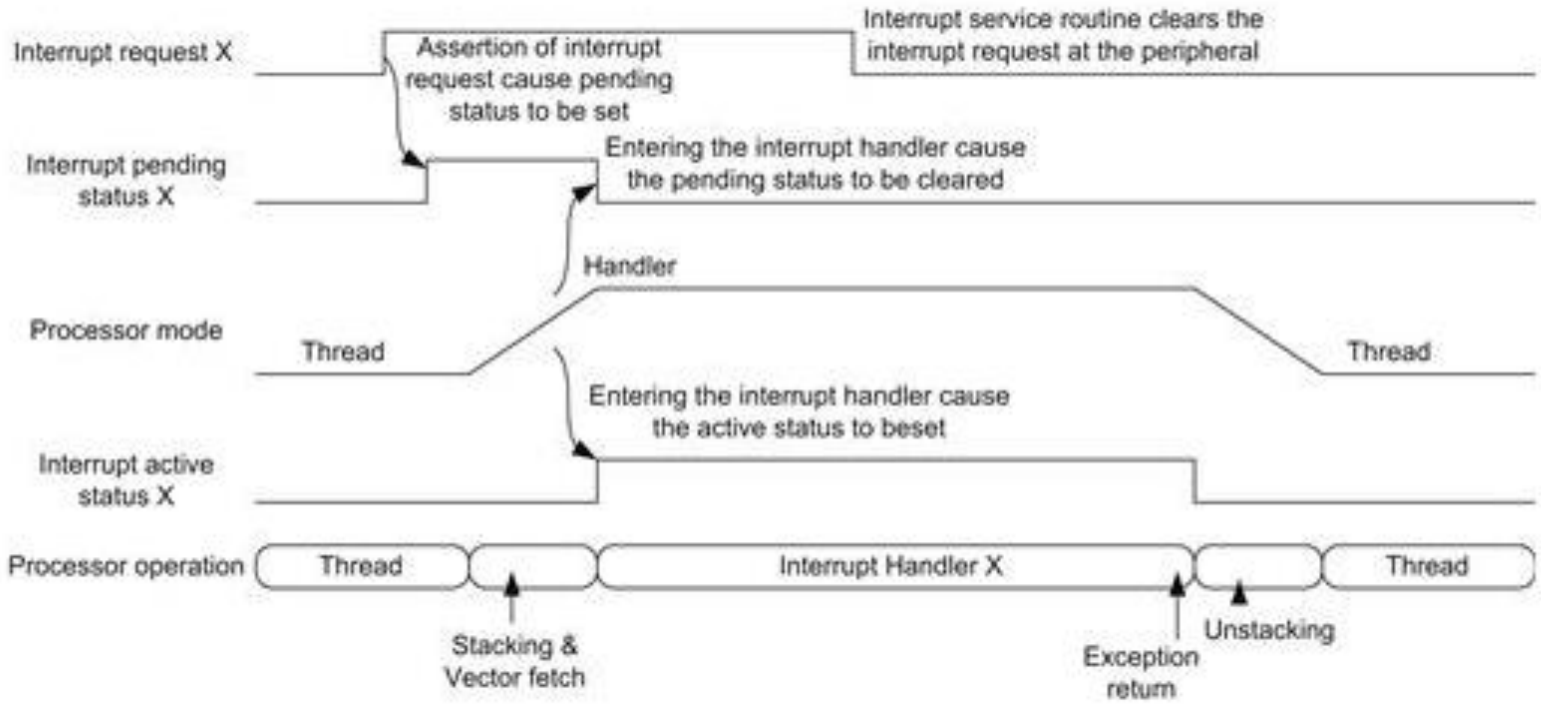
- **NVIC (Nested Vector Interrupt Controller)**
  嵌套中断向量控制器
  - Responsible of interrupt handling
    中断响应
  - Supports vector table relocation
    向量表重定位
  - Up to 240 vectored interrupts
    最高支持240个中断向量
  - 204 interrupts available in S32K14XX
    S32K14系列支持204个中断
- **Asynchronous Wake-up Interrupt Controller (AWIC)**
  异步唤醒中断控制器
  - Detect asynchronous wake-up events in stop modes
    在STOP模式下检测异步唤醒事件
  - Signal to clock control logic to resume system clocking
    发信号通知时钟控制逻辑，来唤醒系统时钟
  - After clock restart, NVIC observes the pending interrupt and performs normal
    时钟重启后，NVIC观测是否有等待状态的中断，执行常规的中断过程
  - Used during low power modes to generate an wake up signal
    用于在低功耗模式下产生唤醒信号



S32K144 Interrupts Lab: Theory

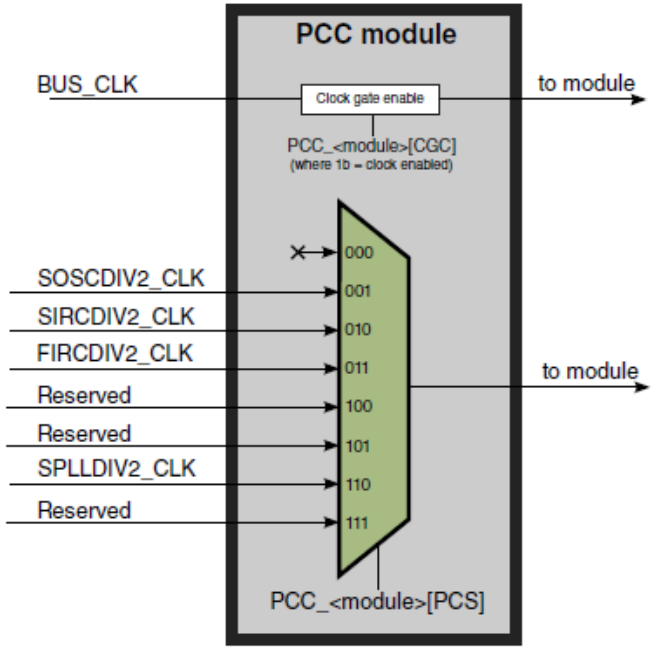
What happens when an interrupt occurs in an ARM Cortex M4?

ARM Cortex M4发生中断的反应时序



S32K144 Interrupts Lab: Theory

- LPIT (Low power interrupt timer) 中断定时器
  - 4 channels
  - Individual or chained channel operation
  - 32 bit counter per channel
  - 4 operation modes:
    - 32-bit Periodic Counter
    - Dual 16-bit Periodic Counter
    - 32-bit Trigger Accumulator
    - 32-bit Trigger Input Capture



Module	VLPR	VLPW	Stop	VLPS
LPIT	Full functionality	Full functionality	Async operation	Async operation

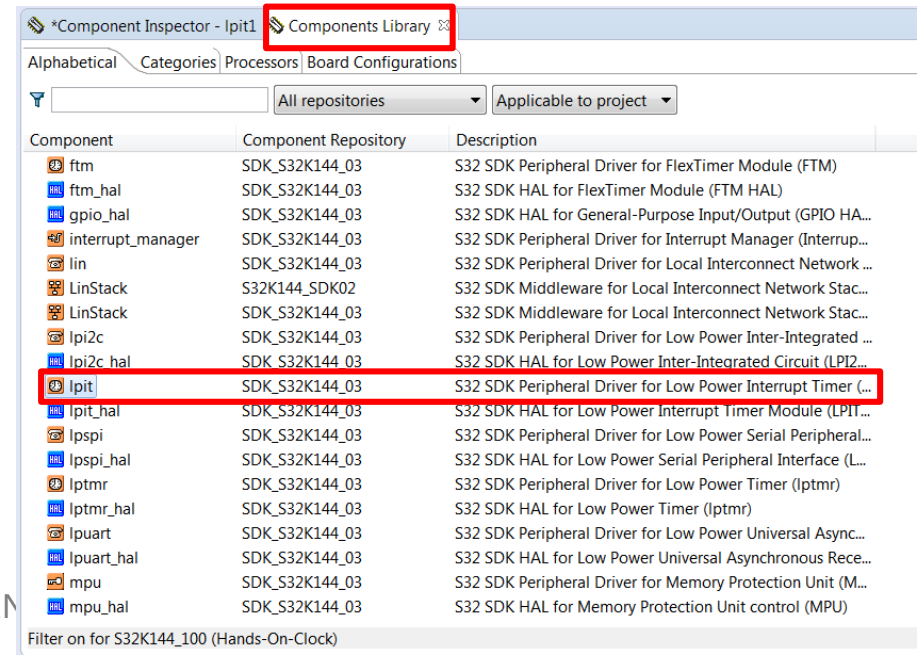
S32K144 Interrupts Lab: LPIT Configuration / LPIT配置

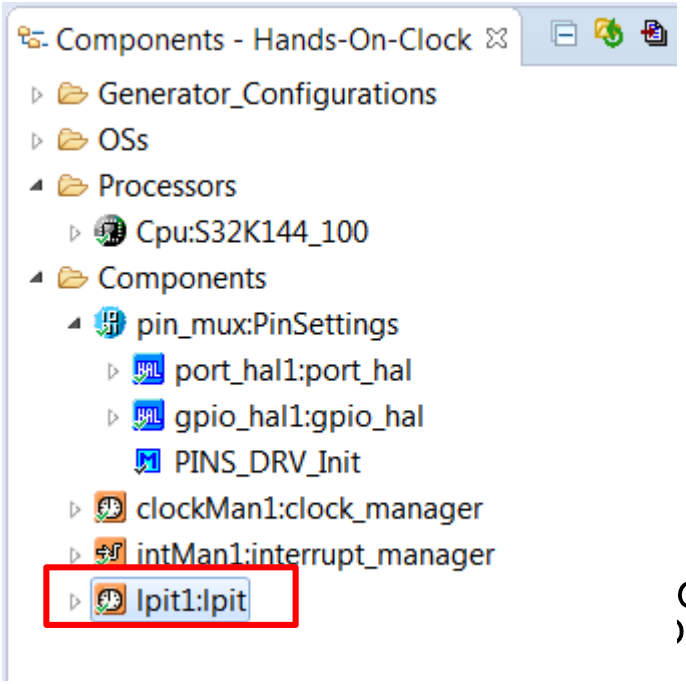
- Create a new S32DS Project

  新建S32DS工程
- Setup the clocks of the S32K144, Setup GPIOs of the S32K144 to enable EVB LED

  配置时钟和GPIO
- Go to **Component Library** window, select the **lpit** in the Alphabetical tab. Double click **lpit** to add to your project.

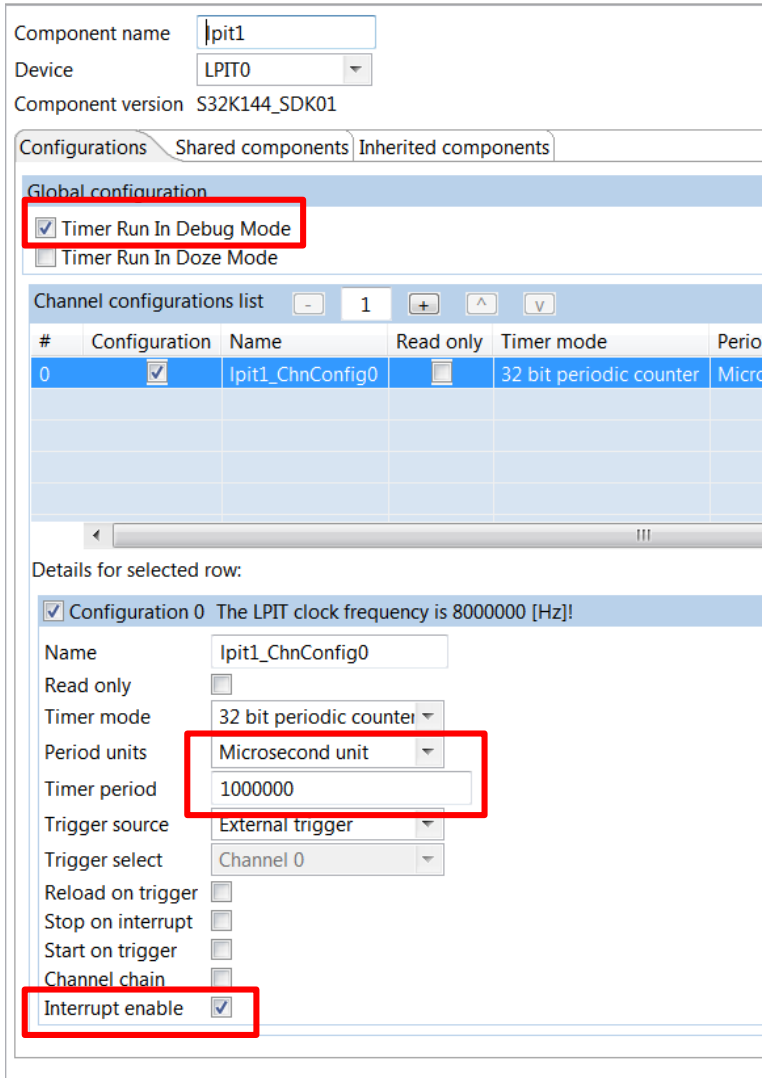
  选择LPIT部件，添加进工程





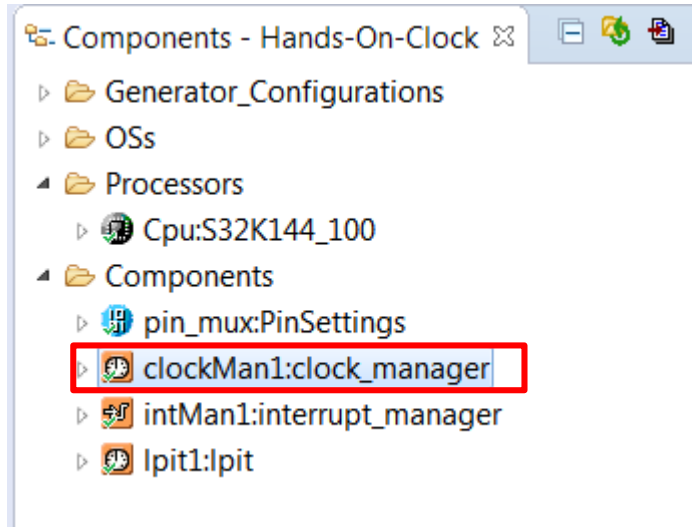
S32K144 Interrupts Lab: Select LPIT Clock / 设置LPIT时钟

- Go to **Components Inspector**.
  在LPIT部件视图下
- Check the **Timer Run In Debug Mode** box
  勾选Timer Run in Debug Mode选项
- Check the **Interrupt enable** box
  勾选Interrput enable选项
- Select in **Period units Microseconds unit**
  选择Period Units为us
- In the **Timer period** field type **1000000** counts for 1 sec
  在Timer Period中填写1000000，表示1s



S32K144 Interrupts Lab: Select LPIT Clock / 设置LPIT时钟

- In the **Components Window** select the **Clock Manager component**.
  选择Clock Manager部件
  - Select the Settings tab
    选择Setting标签栏
  - Check the Enable box for LPIT
    使能LPIT模块的时钟
  - Select the **SOSCDIV2_CLK** as the functional clock source
    选择SOSCDIV2_CLK作为LPIT的功能时钟源



Clock configuration 0

Settings | SIRC | FIRC | RTC | SOSC | SPLL | CLKOUT | LPO | SIM | TCLK | Trace | Clock Values Summary

Peripheral Clocks

Clock Name	Enable	Interface Clock	Functional Clock	Multiply	Divide	Frequency
FTM2_CLK	☐	~~SYS_CLK~~	~~SIRCDIV1_CLK~~			~~0 Hz~~
FTM3_CLK	☐	~~SYS_CLK~~	~~SIRCDIV1_CLK~~			~~0 Hz~~
LPI2C0_CLK	☐	~~BUS_CLK~~	~~SIRCDIV2_CLK~~			~~0 Hz~~
LPIT0_CLK	☑	BUS_CLK	SOSCDIV2_CLK			8 MHz
LPSPI0_CLK	☐	~~BUS_CLK~~	~~SIRCDIV2_CLK~~			~~0 Hz~~
LPSPI1_CLK	☐	~~BUS_CLK~~	~~SIRCDIV2_CLK~~			~~0 Hz~~
LPSPI2_CLK	☐	~~BUS_CLK~~	~~SIRCDIV2_CLK~~			~~0 Hz~~

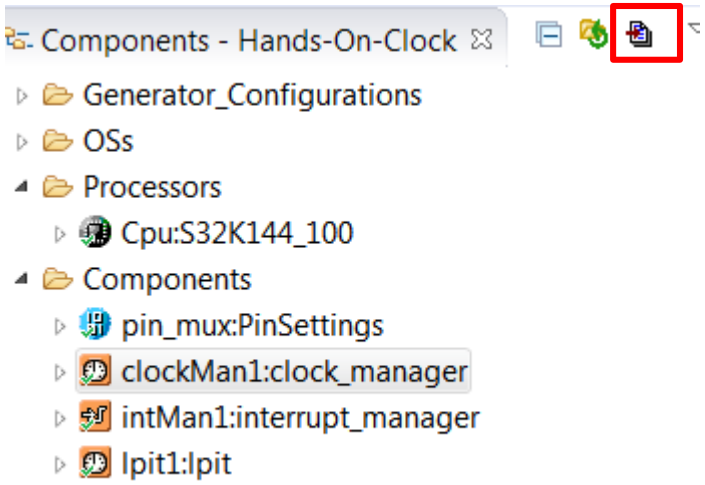
S32K144 GPIOs Lab: Generate the code

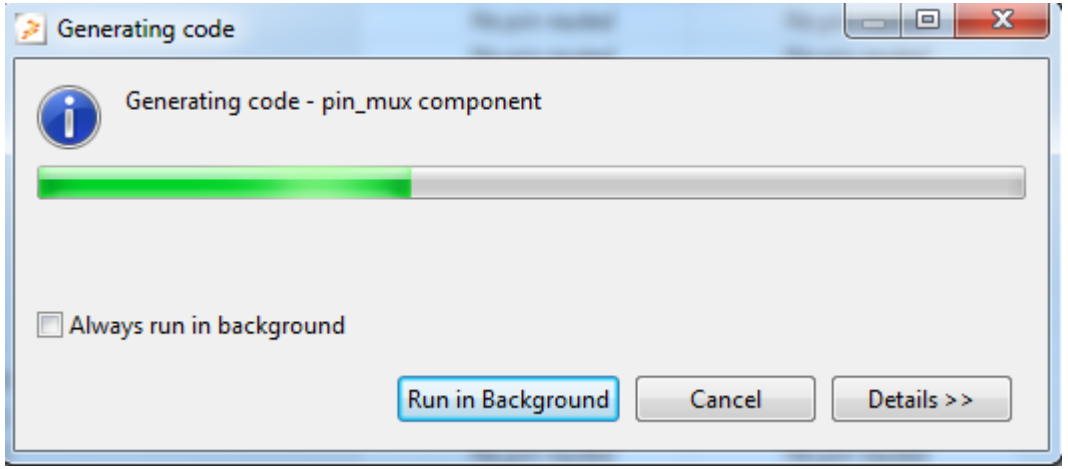
- To generate the code for the configuration select, click the **generate code** icon in the **Components** window.

  配置代码生成，点击Generate Code图标

- Wait for the code to be generated.

  等待进度条完成



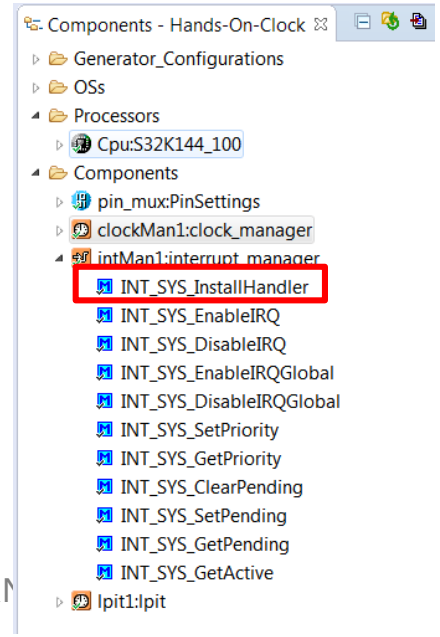


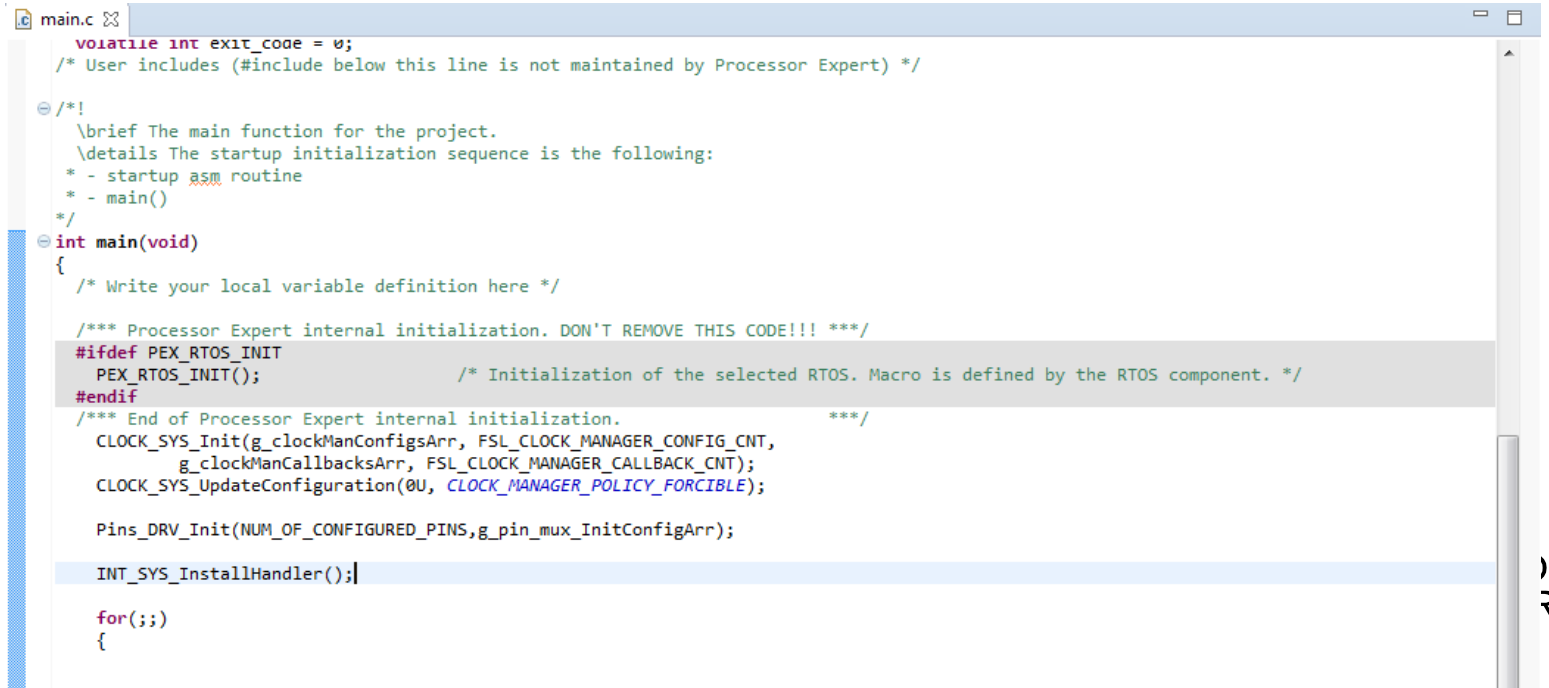
S32K144 Interrupts Lab: Install LPIT interrupt / 装载LPIT中断

- In the project window double click the **main.c** file to open it

  打开Main.C函数
- Expand the **interrupt_manager** component in the **Components** Window

  展开interrupt_manager部件
- Drag and drop the **INT_SYS_InstallHandle**r function. Placed it after the **Pins_DRV_Init** function in main.c
- 拖拽INT_SYS_InstallHander函数至Main函数，放在Pin初始化函数后面





S32K144 Interrupts Lab: Install LPIT interrupt / 装载LPIT中断

- In the **INT_SYS_InstallHandle**r function add the following parameters:

  INT_SYS_InstallHandler函数需要如下参数

  - LPIT0_Ch0_IRQn,
  - &LPIT_ISR,
  - (isr_t *)0

```
/* Install LPIT_ISR as LPIT interrupt handler */
    INT_SYS_InstallHandler(LPIT0_Ch0_IRQn, &LPIT_ISR, (isr_t *)0);
```

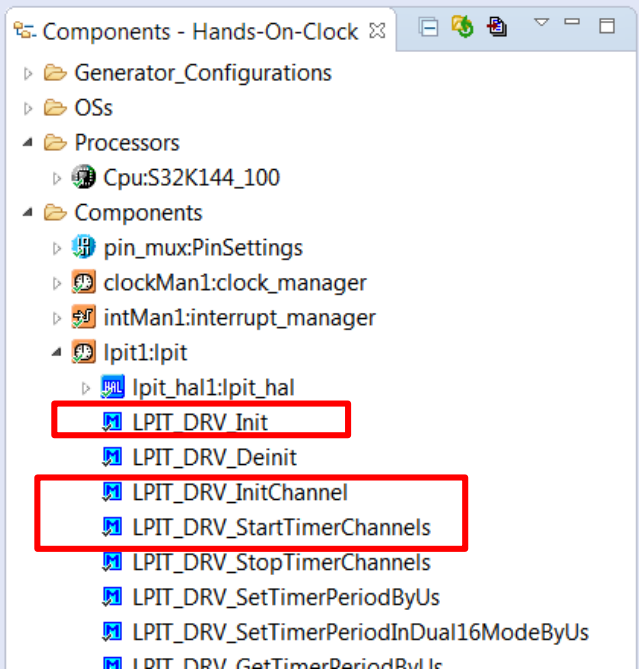
- Create a new function named LPIT_ISR and placed above main

  在Main函数前，新建一个函数，名为LPIT_ISR

```
void LPIT_ISR(void)
{

}


/*!
  \brief The main function for the project.
  \details The startup initialization sequence is the following:
 * - startup asm routine
 * - main()
*/
int main(void)
{
  /* Write your local variable definition here */
```

S32K144 Interrupts Lab: Initialize LPIT / 初始化LPIT

- Expand the **lpit** component in the **Components** Window
  展开LPIT部件
- Drag and drop the following functions in to main, place them after the **INT_SYS_InstallHandle**r function
  拖拽以下函数至Main函数，放在INT_SYS_InstallHandler函数
  - **LPIT_DRV_Init**
  - **LPIT_DRV_InitChannel**
  - **LPIT_DRV_StartTimerChannels**




```
/* Install LPIT_ISR as LPIT interrupt handler */
INT_SYS_InstallHandler(LPIT0_Ch0_IRQn, &LPIT_ISR, (isr_t *)0);

LPIT_DRV_Init(INST_LPIT1, &lpit1_InitConfig);
LPIT_DRV_InitChannel(INST_LPIT1, /*uint32_t channel*/,
                                        &lpit1_ChnConfig0);
LPIT_DRV_StartTimerChannels(INST_LPIT1,);
```

S32K144 Interrupts Lab: Initialize LPIT / 初始化LPIT

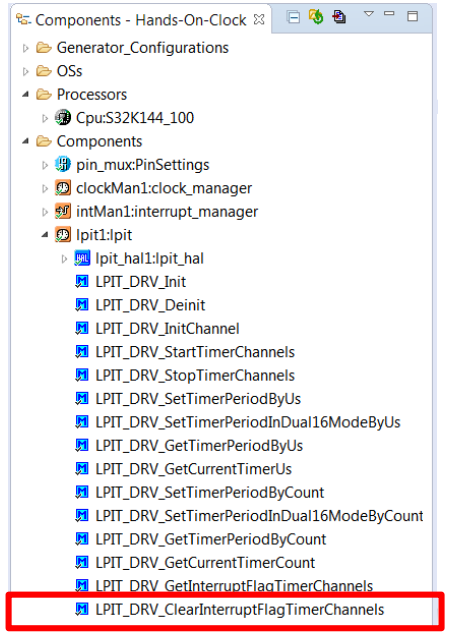
三个函数分别需要以下参数

- In the **LPIT_DRV_Init** function add the following parameters:.
  - FSL_LPIT1,
  - &lpit1_InitConfig
- In the **LPIT_DRV_InitChannel** function add the following parameters:.
  - FSL_LPIT1,
  - 0
  - &lpit1_ChnConfig0
- In the **LPIT_DRV_StartTimerChannels** function add the following parameters:.
  - FSL_LPIT1,
  - (1 << 0)

```
/* Install LPIT_ISR as LPIT interrupt handler */
INT_SYS_InstallHandler(LPIT0_Ch0_IRQn, &LPIT_ISR, (isr_t *)0);

LPIT_DRV_Init(INST_LPIT1, &lpit1_InitConfig);
LPIT_DRV_InitChannel(INST_LPIT1, 0, &lpit1_ChnConfig0);
LPIT_DRV_StartTimerChannels(INST_LPIT1, (1<<0));
```

S32K144 Interrupts Lab: Clear LPIT Flag in interrupt / 清中断标志

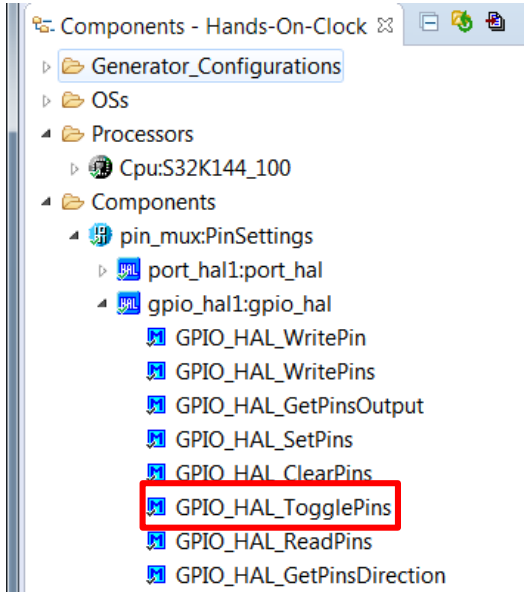
- Expand the **lpit** component in the **Components** Window
  展开LPIT部件
- Drag and drop the following function into **LPIT_ISR:**
  在LPIT_ISR子部件下，拖拽以下函数
  - **LPIT_DRV_ClearInterruptFlagTimerChannels**
- In the **LPIT_DRV_ClearInterruptFlagTimerChannels** function add the following parameters:
  函数参数如下
  - FSL_LPIT1
  - (1 << 0)




```
void LPIT_ISR(void)
{
    LPIT_DRV_ClearInterruptFlagTimerChannels(FSL_LPIT1,(1 << 0));


}
```

S32K144 Interrupts Lab: Toggle Green LED (PTD16) / 闪烁LED

- Expand the **gpio_hal** component in the **Components** Window
  展开GPIO_HAL子部件
- Drag and drop the **GPIO_HAL_TogglePins** function into LPIT_ISR
  拖拽GPIO_HAL_TogglePins函数，放在LPIT_ISR函数中
- Add the following parameters: 添加参数
  - PTD
  - (1<<16)




```
void LPIT_ISR(void)
{
    LPIT_DRV_ClearInterruptFlagTimerChannels(FSL_LPIT1,(1 << 0));
    GPIO_HAL_TogglePins(PTD,(1<<16));
}
```

- Remember to initialize LEDs to be OFF (low-side drive):
  初始化时先将LED熄灭
- GPIO_HAL_SetPins(PTD, 1<<16);

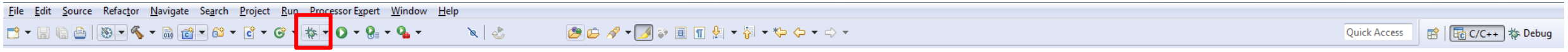
S32K144 Interrupts Lab: Build and debug the lab / 编译调试工程

- Click on the build icon to make sure that there a no compiler errors.

  编译工程，确认没有报错

- Configure the debug configuration start a new debug session

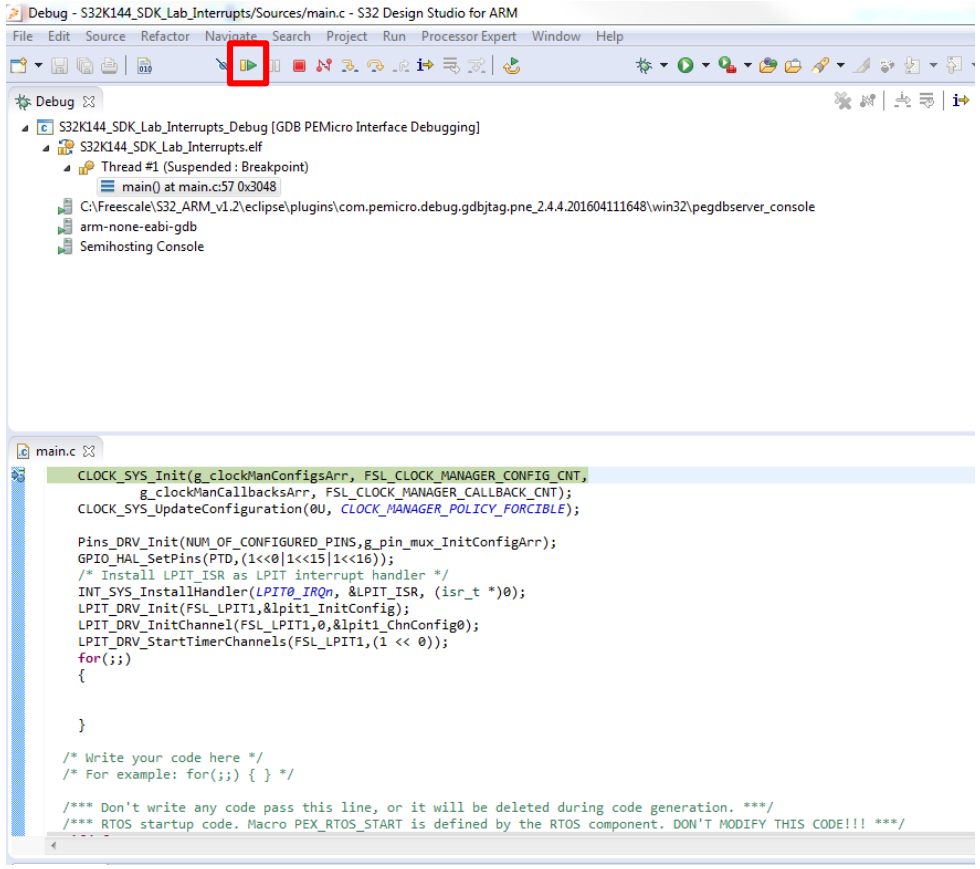
  配置调试口，开始调试

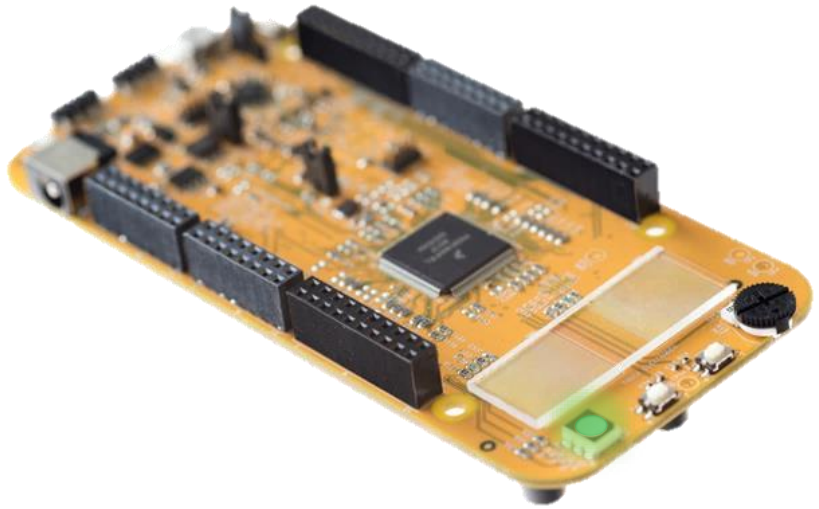


S32K144 Interrupts Lab: Build and debug the lab / 编译调试工程

- In the debug perspective click the run icon to start the project.
- Green LED should toggle every 1 sec.

  全速运行，观察到绿色LED闪烁





Lab 04.

S32K144 FLEXCAN

- In this lab you will learn:

  本实验你将学习以下内容

  - How to configure FlexCAN module for CAN transmission and Reception

    配置FlexCAN模块收发CAN报文

  - Use callbacks for Send and Receive operations

    使用回调函数来发送和接受

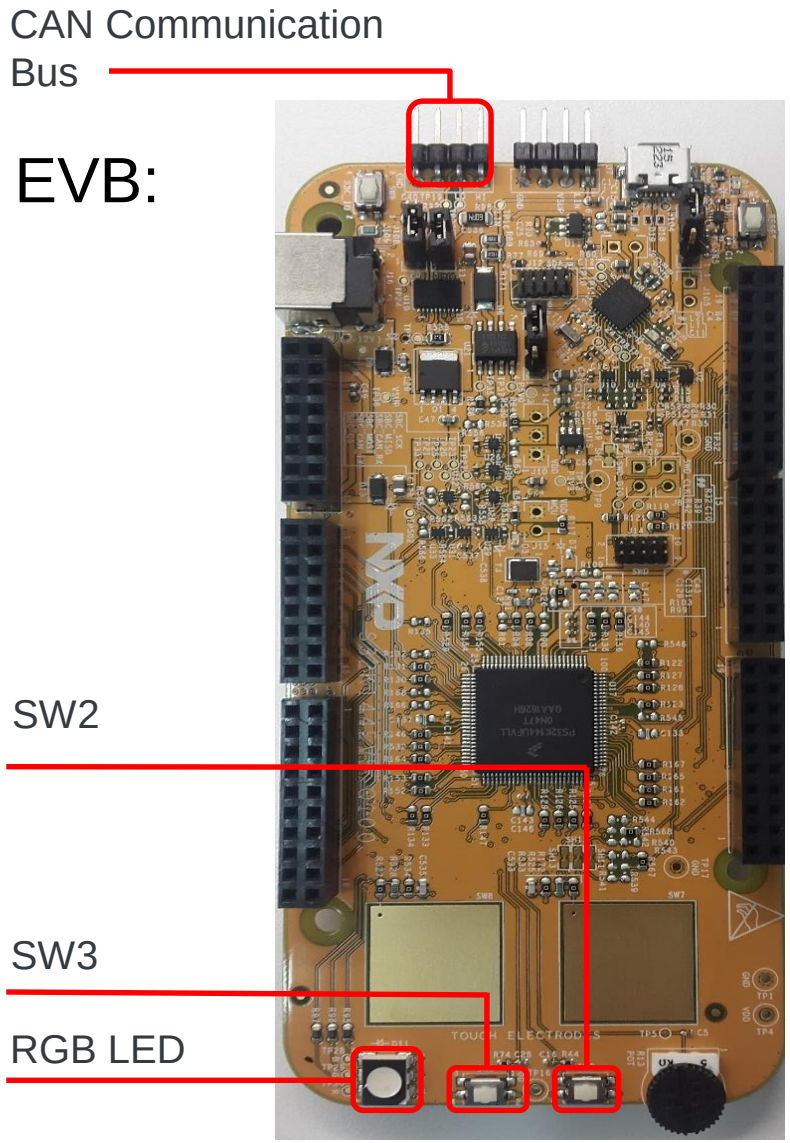
  - Use pin edge interrupt

    使用Pin边沿中断模式

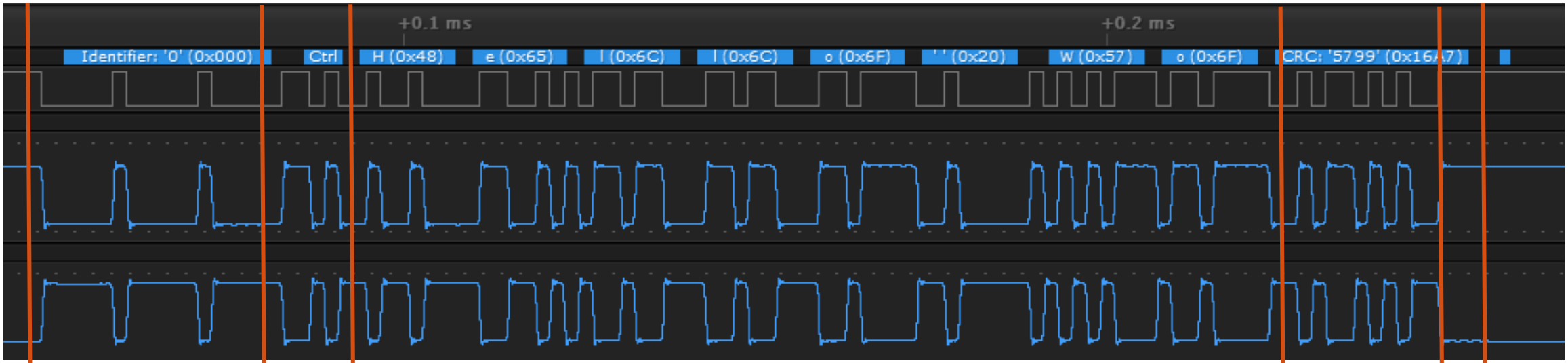
S32K144 FlexCAN Lab: Theory

- In this lab will be used the following components of the EVB:
  本实验中，你将使用以下EVB资源
  - RED and Green LEDs
    红色和绿色LED
  - CAN Low and CAN High
    CAN高、CAN低
  - SW2 and SW3
    按键2、按键3

Component	S32K144 PIN
BLUE LED	PTD0
RED LED	PTD15
GREEN LED	PTD16
SW2	PTC12
SW3	PTC13
CAN TX	PTE5
CAN RX	PTE4



S32K144 FlexCAN Lab: CAN Message / CAN报文格式



S32K144 FlexCAN Lab: Theory

- Provides full implementation for CAN FD and Standard CAN implementation (CAN 2.0B)

  支持全功能的CANFD协议和标准CAN协议
- Compliant with ISO 11899-1 standard

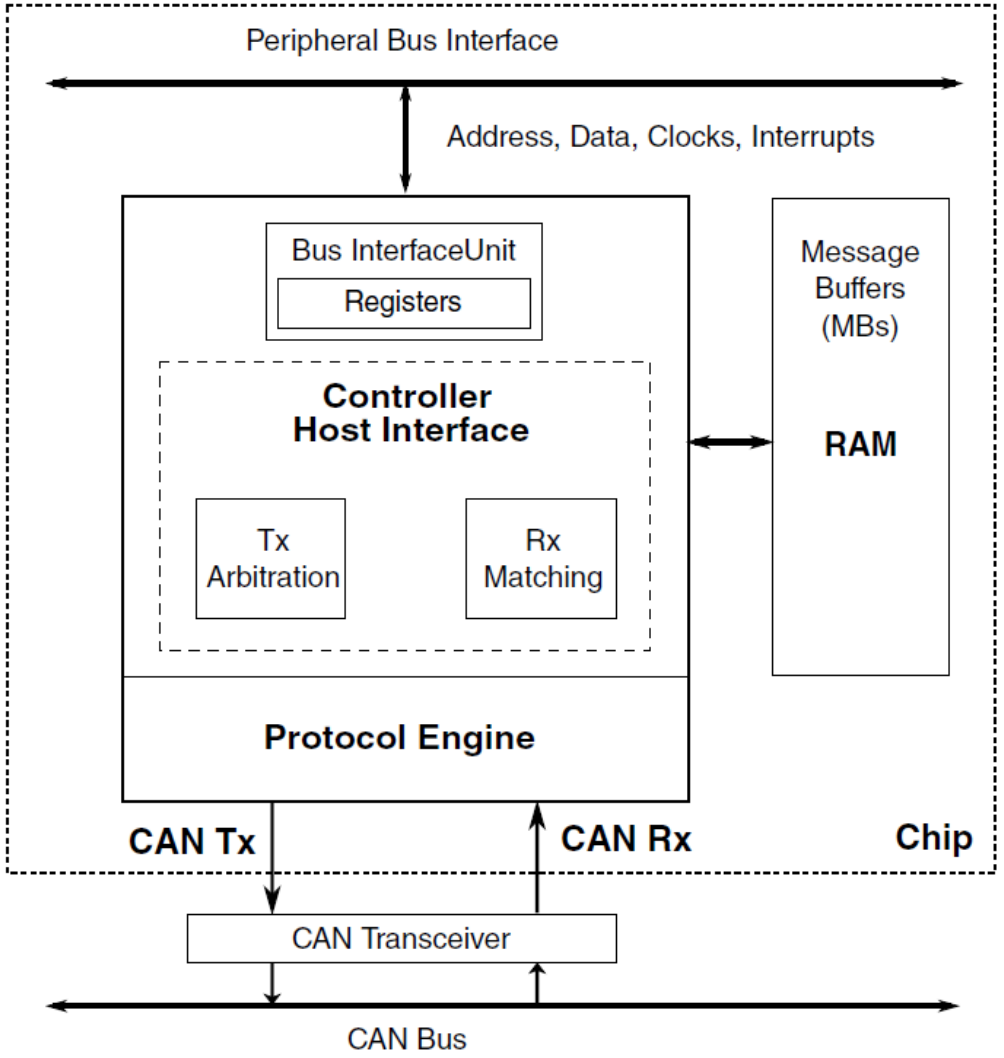
  符合ISO 11899-1标准
- Configurable mailboxes with 0-64 bytes of data

  Mailboxes可配置成0-64byte
- RX FIFO and Pretended Networking support

  支持RX FIFO模式和Pretended Networking模式
- 3 instances of FlexCAN available on S32K144

  S32K144中有3个FlexCAN

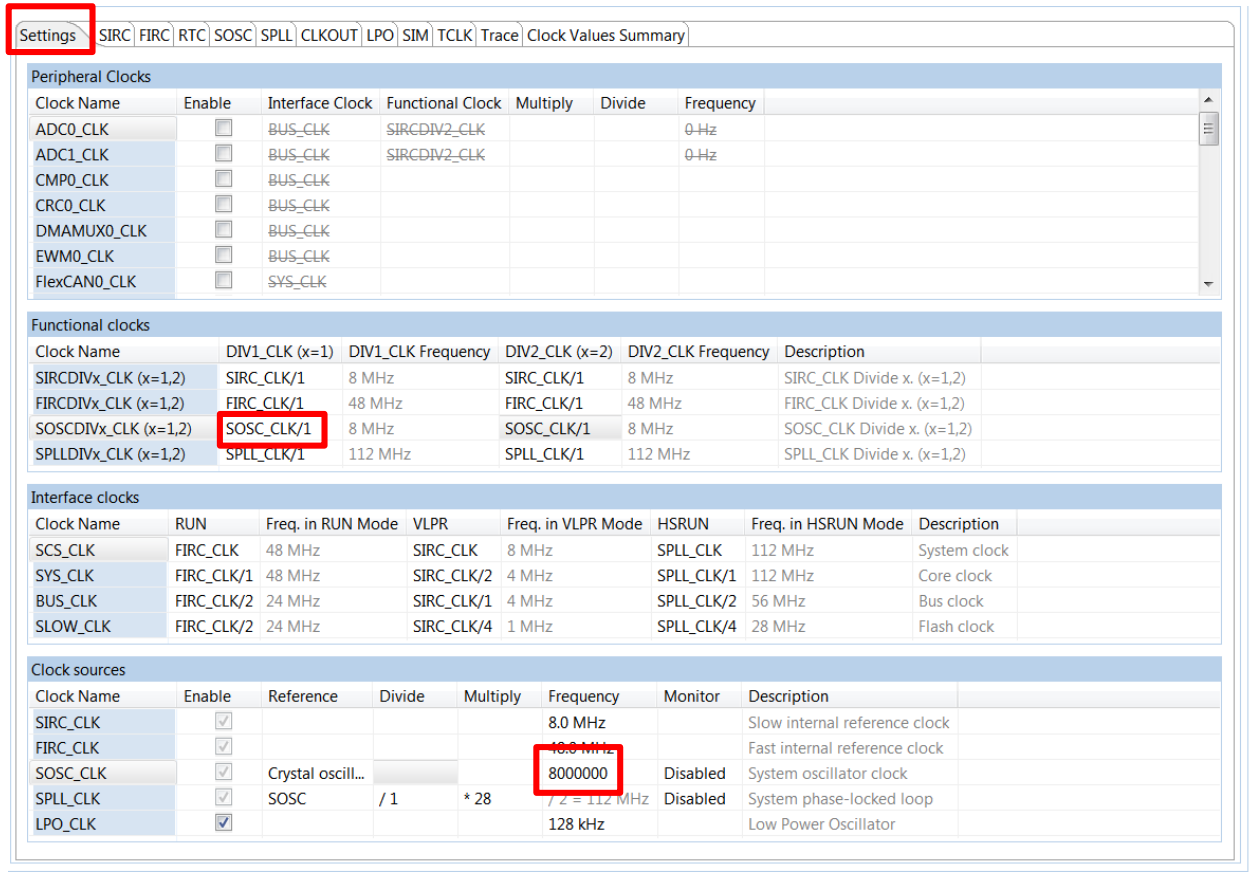
Module Names	Number of Message Buffers (MB)	ISO CAN FD feature	PNET feature
FlexCAN0	32 MBs	Yes	Yes
FlexCAN1	16 MBs	No	No
FlexCAN2	16 MBs	No	No



S32K144 FlexCAN Lab: Enable Clocks / 使能时钟

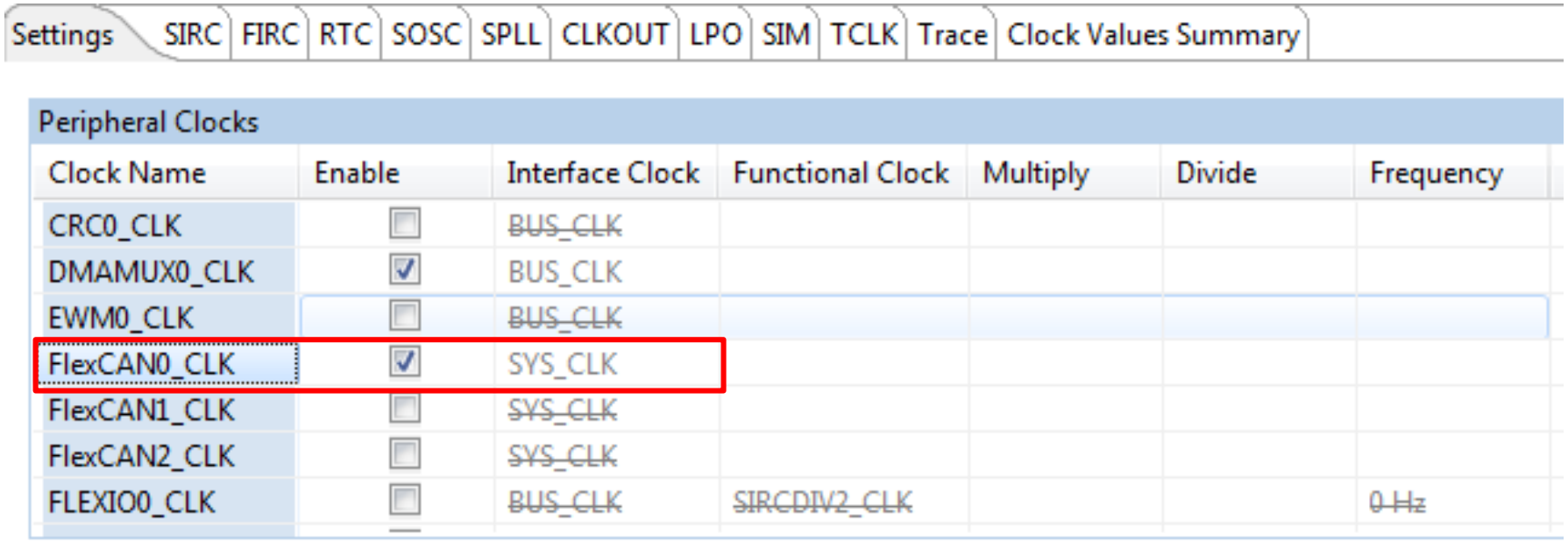
- Select the **clock_manager** component in the **Components** window
- In the **Clock sources -> SOSC_CLK** field write 8000000(corresponding to 8MHz)
- In the **Functional Clock-> SOSCDIVx_CLK->DIV1_CLK** select SOSC_CLK/1

  配置SOSC_CLK为8MHz，设置其分频时钟SOSCDIV1_CLK为SOSC_CLK/1



S32K144 FlexCAN Lab: Enable Clocks / 使能时钟

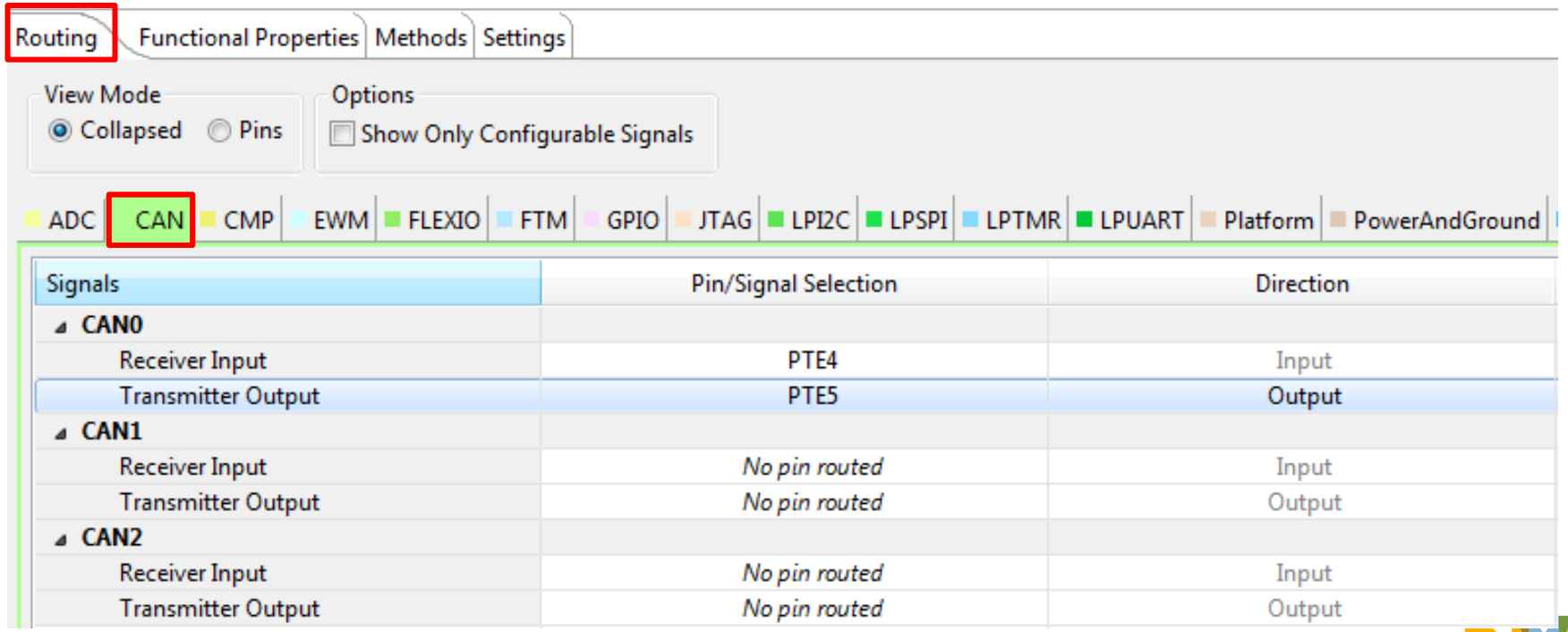
- In **Peripheral Clock section** -> **FlexCAN0_CLK** check: **enable**
  使能FLexCAN0模块的时钟

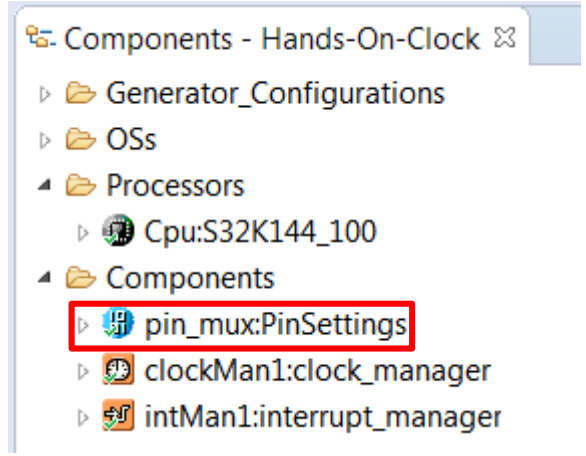


S32K144 FlexCAN Lab: Select I/O pins / 设置I/O管脚

- Select the **pin_mux** component in the **Components** window
- Select **CAN** tab inside the **Routing** tab
- In the **Pin/Signal Selection** Column, select **PTE4(Rx)** and **PTE5(Tx)**.

  配置CAN0模块的管脚，PTE4为Rx，PTE5为Tx

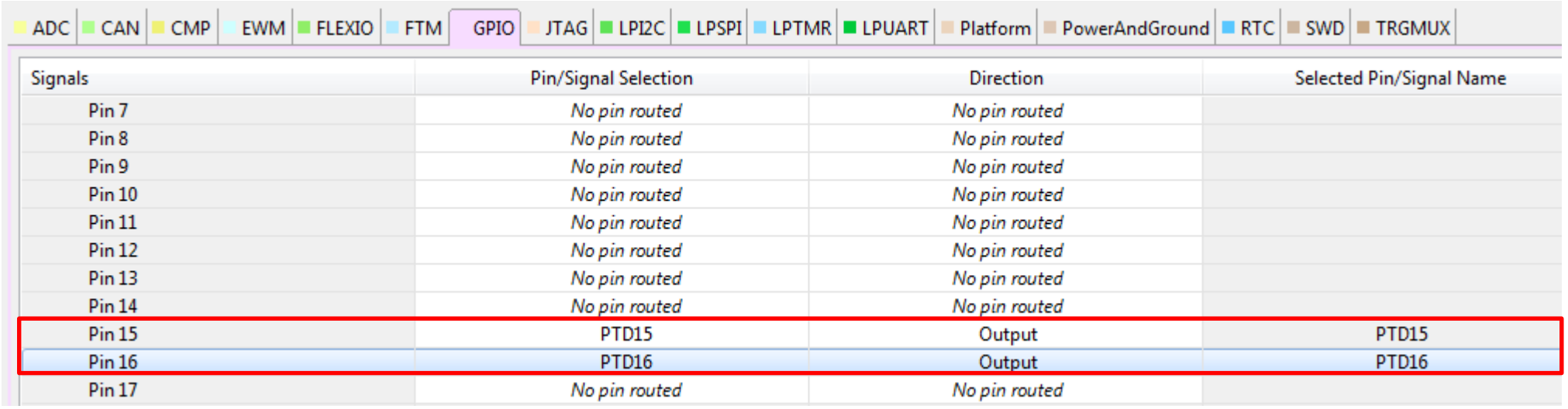




S32K144 FlexCAN Lab: Select I/O pins / 设置I/O管脚

- Select **PTD**, Pin 15 and 16 -> set **PTD15** and **PTD16**
- In the **direction** tab configure the pin as **Output**

  配置PTD15和PTD16为输出管脚

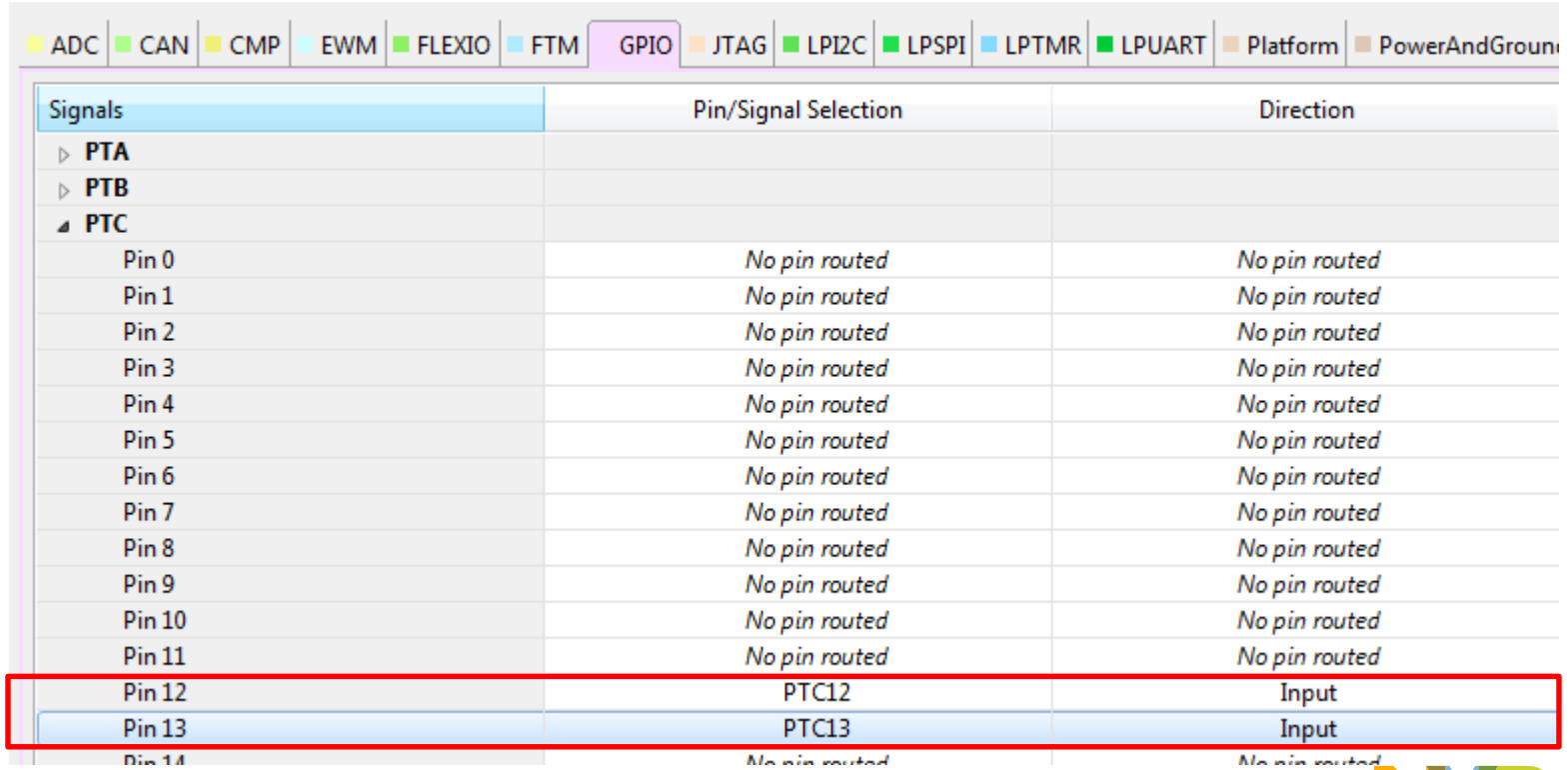


Signals	Pin/Signal Selection	Direction	Selected Pin/Signal Name
Pin 7	*No pin routed*	*No pin routed*	
Pin 8	*No pin routed*	*No pin routed*	
Pin 9	*No pin routed*	*No pin routed*	
Pin 10	*No pin routed*	*No pin routed*	
Pin 11	*No pin routed*	*No pin routed*	
Pin 12	*No pin routed*	*No pin routed*	
Pin 13	*No pin routed*	*No pin routed*	
Pin 14	*No pin routed*	*No pin routed*	
Pin 15	PTD15	Output	PTD15
Pin 16	PTD16	Output	PTD16
Pin 17	*No pin routed*	*No pin routed*	

S32K144 FlexCAN Lab: Select I/O pins / 设置I/O管脚

- Select **PTC**, Pin 12 and 13 -> set **PTC12** and **PTC13**
- In the **direction** tab configure the pin as **Input**

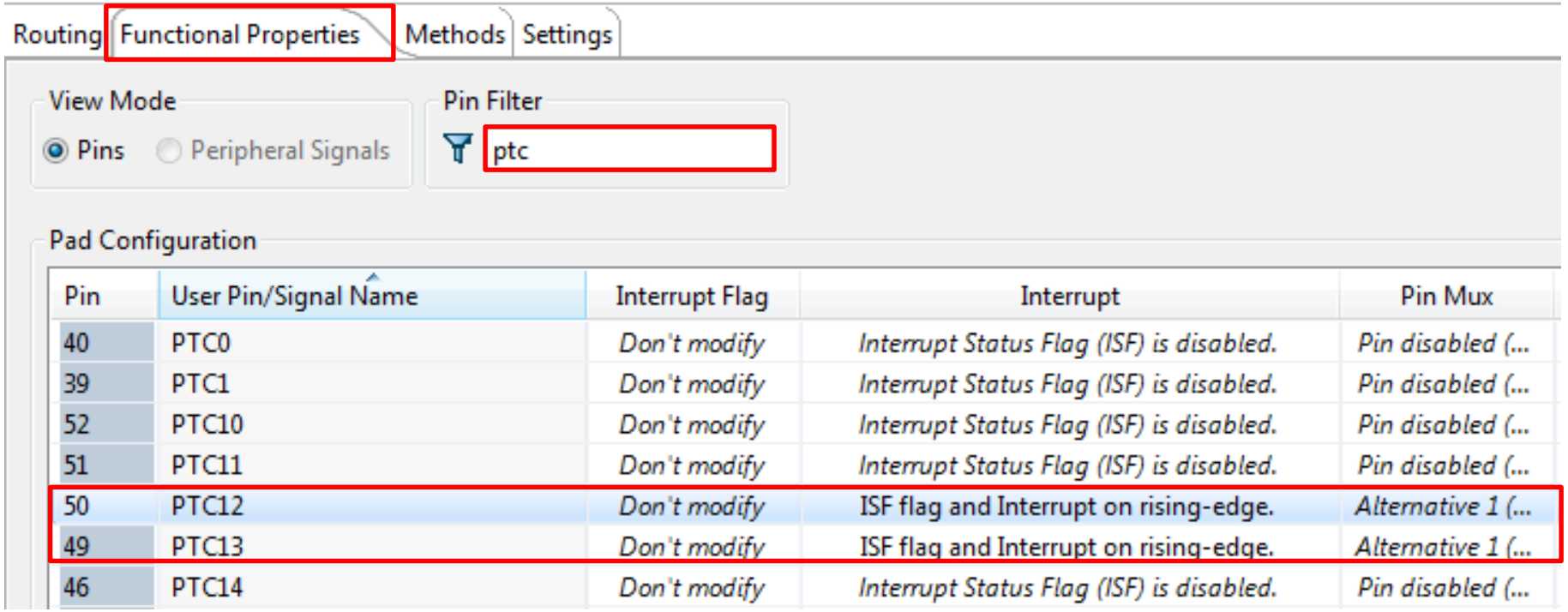
  配置PTC12和PTC13为输入管脚



Signals	Pin/Signal Selection	Direction
PTA		
PTB		
PTC		
Pin 0	*No pin routed*	*No pin routed*
Pin 1	*No pin routed*	*No pin routed*
Pin 2	*No pin routed*	*No pin routed*
Pin 3	*No pin routed*	*No pin routed*
Pin 4	*No pin routed*	*No pin routed*
Pin 5	*No pin routed*	*No pin routed*
Pin 6	*No pin routed*	*No pin routed*
Pin 7	*No pin routed*	*No pin routed*
Pin 8	*No pin routed*	*No pin routed*
Pin 9	*No pin routed*	*No pin routed*
Pin 10	*No pin routed*	*No pin routed*
Pin 11	*No pin routed*	*No pin routed*
Pin 12	PTC12	Input
Pin 13	PTC13	Input

S32K144 FlexCAN Lab: Configure Button Interrupt / 配置按键中断

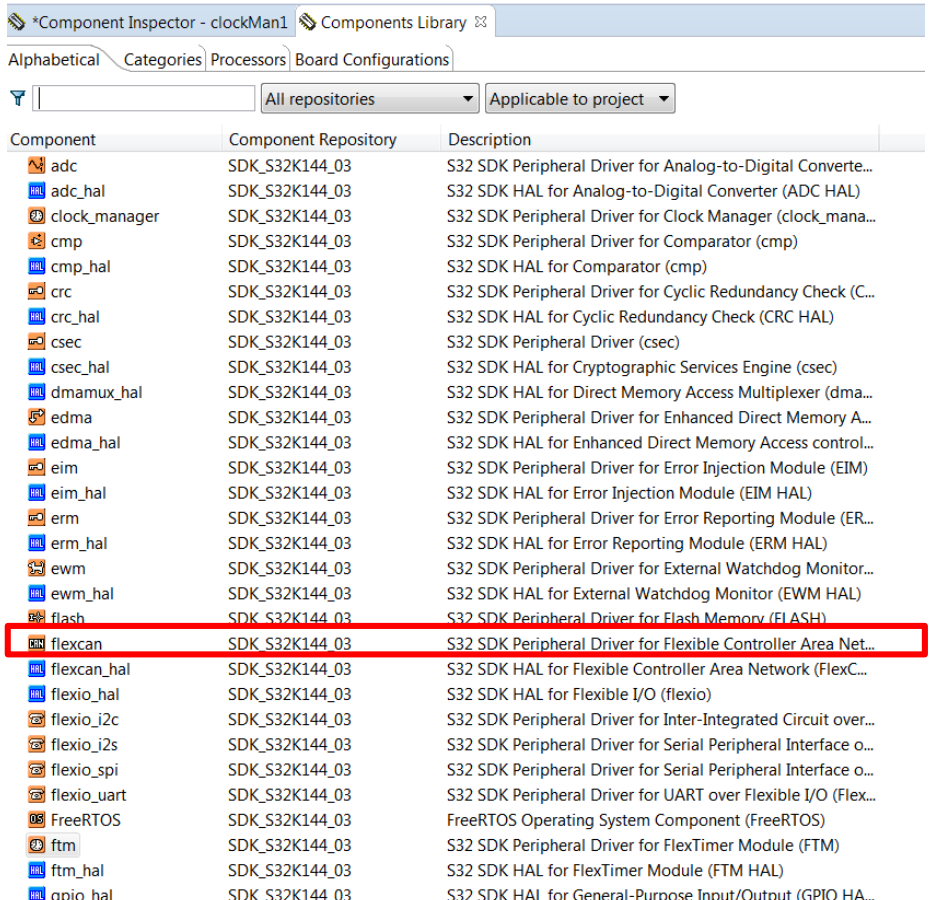
- Switch to **Functional Properties** tab and type **ptc** in **Pin Filter** input
- Select “ISF Flag and Interrupt on rising-edge” for PTC12 and PTC13
  在Functional Properties标签栏中设置PTC12和PTC13为上升沿中断

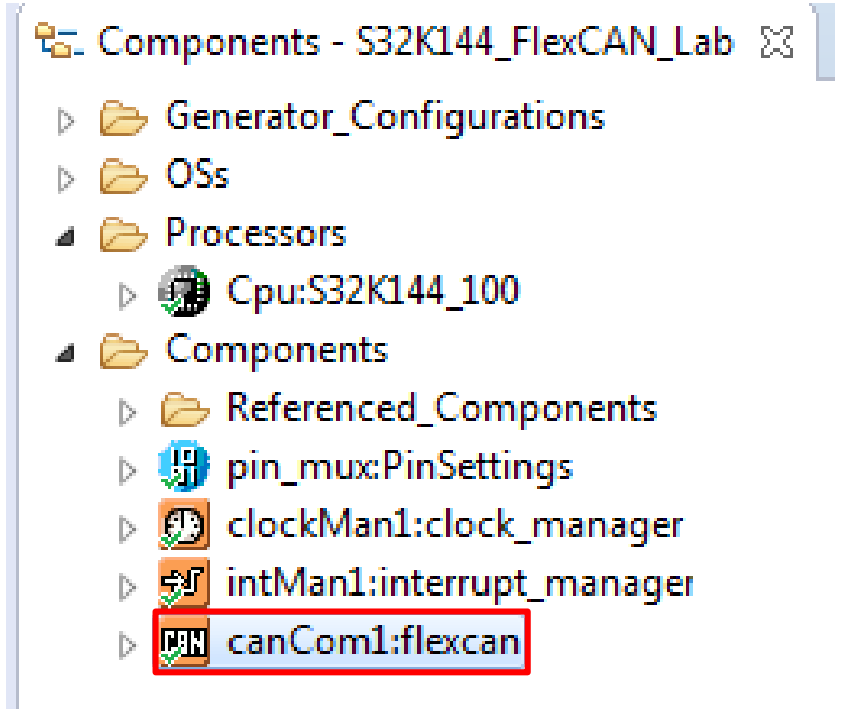


S32K144 FlexCAN Lab: Configure FlexCAN / 配置FlexCAN

- Go to **Component Library** window. Select the **flexcan** in the Alphabetical tab.
- Double click **flexcan** to add to your project. FlexCAN component should appear on the component window.

添加FlexCAN部件至工程中

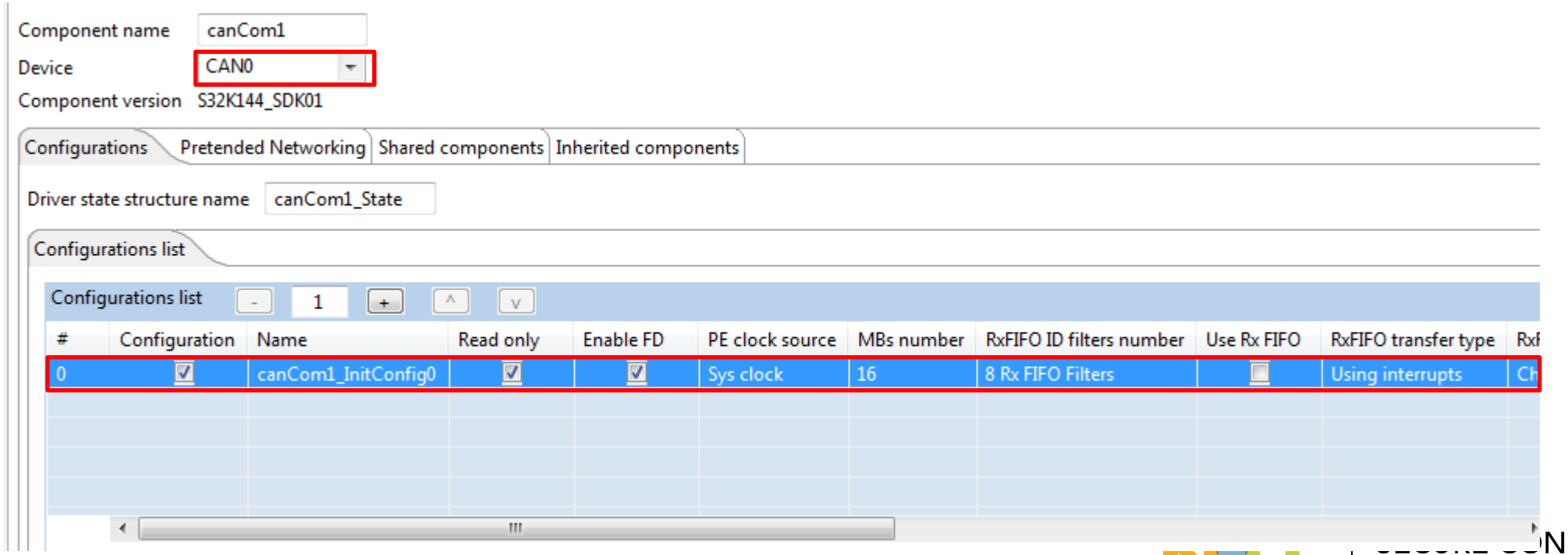




S32K144 FlexCAN Lab: Configure FlexCAN / 配置FlexCAN

- Select **CAN0** as Device.
- Select first item from the **Configurations list**.
- The configuration will appear under the table.

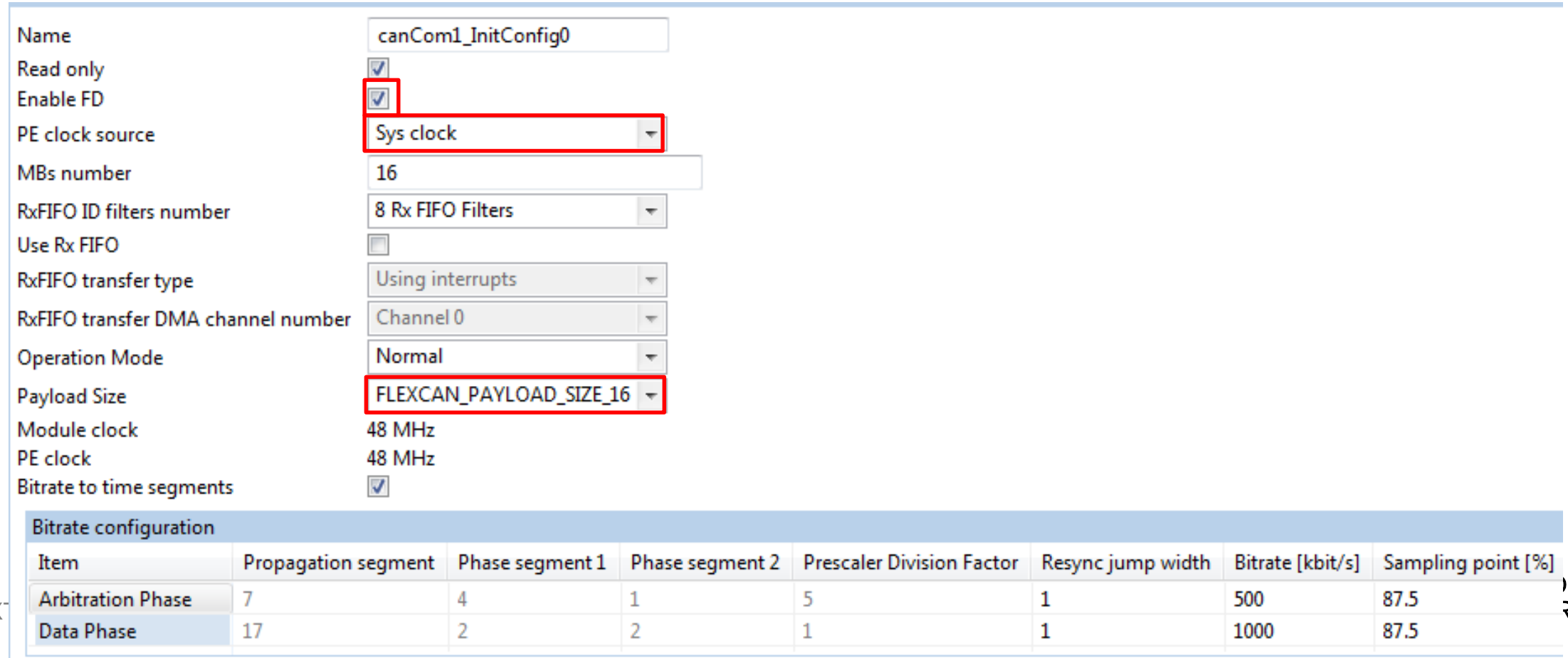
选择CAN0,，及其配置



S32K144 FlexCAN Lab: Configure FlexCAN / 配置FlexCAN

- Check the box linked to **Enable FD**
- Select **PE clock** source as **Sys clock** and **Payload size** to **16**

  勾选Enable FD，选择Sys Clock做为CAN模块的时钟源

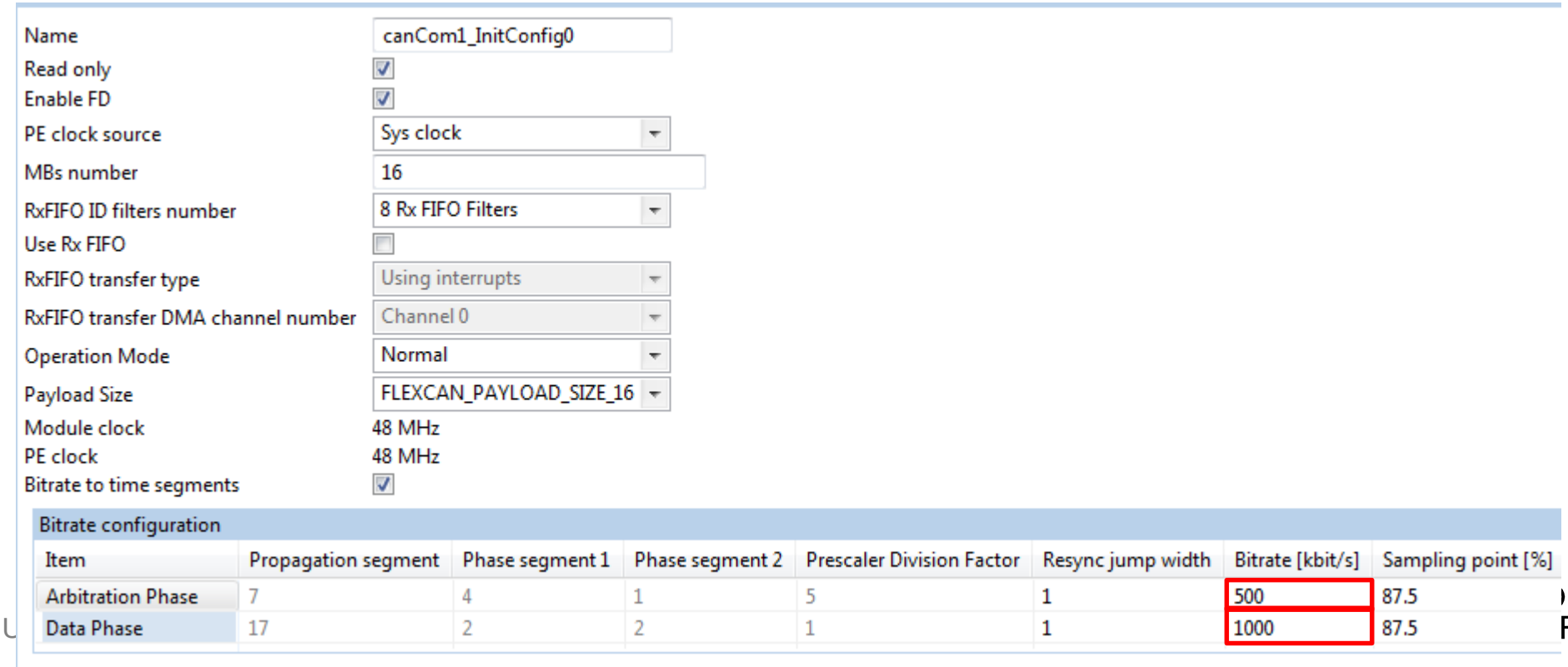


S32K144 FlexCAN Lab: Configure FlexCAN / 配置FlexCAN

- Set **Bitrate** for **Arbitration Phase** to 500kbit/s
- Set **Bitrate** for **Data Phase** to 1000kbit/s

  配置Arbitration Phase的波特率为500kbit/s

  配置Data Phase的波特率为1000kbit/s



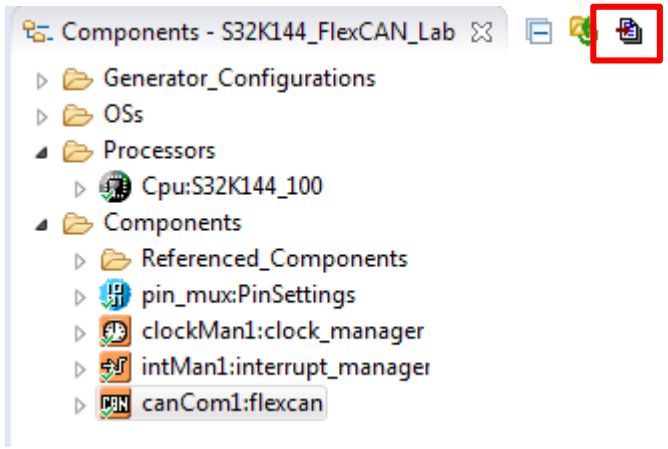
S32K144 FlexCAN Lab: Generate the code / 代码生成

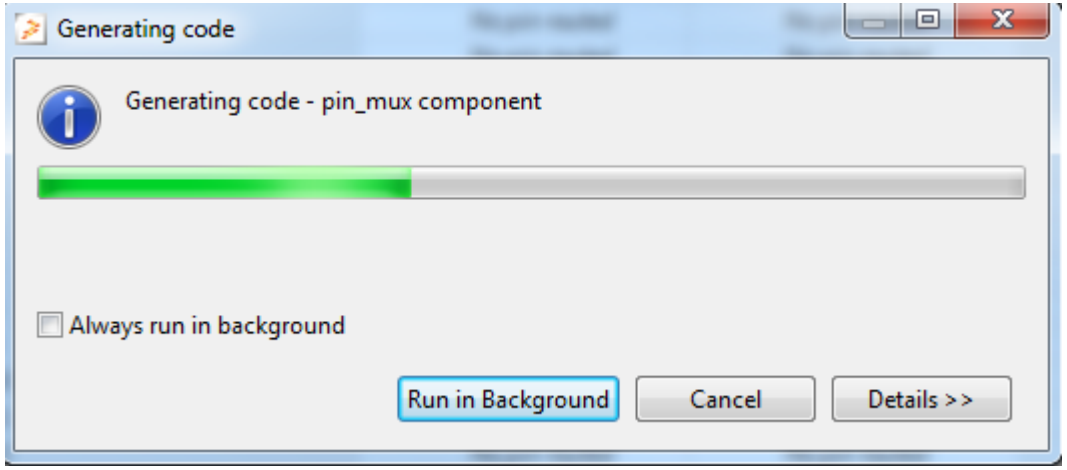
- To generate the code for the configuration select, click the **generate code** icon in the **Components** window.

  配置代码生成，点击Generate Code图标

- Wait for the code to be generated.

  等待进度条完成



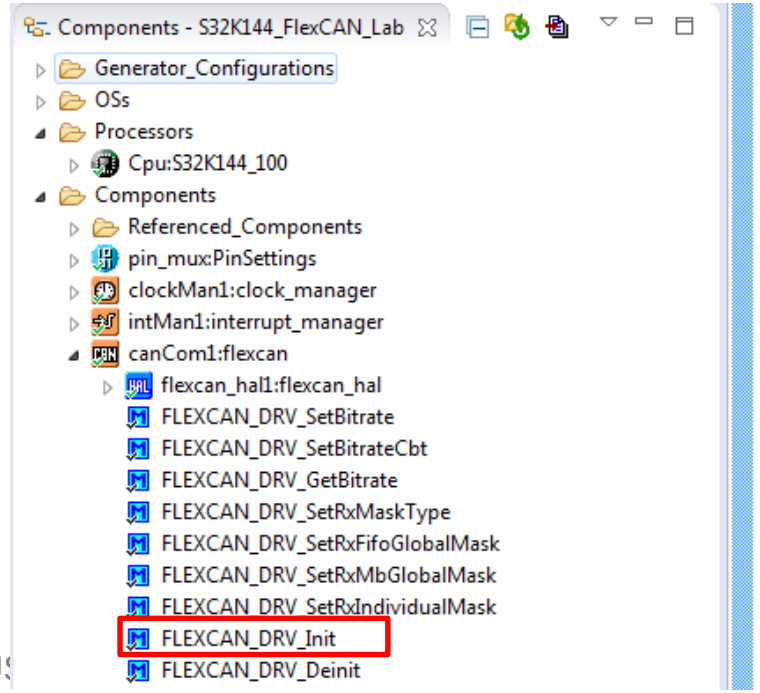


S32K144 FlexCAN Lab: FlexCAN Configuration / 配置FlexCAN

- Add Clock Init and Update Configuration functions.
- Initialize pins.

  初始化时钟和Pin配置
- Drag and Drop **FLEXCAN_DRV_Init** into main.c file, after **PINS_DRV_Init** call.

  拖拽FLEXCAN_DRV_Init函数至Main函数，放在PINS_DRV_Init函数之后

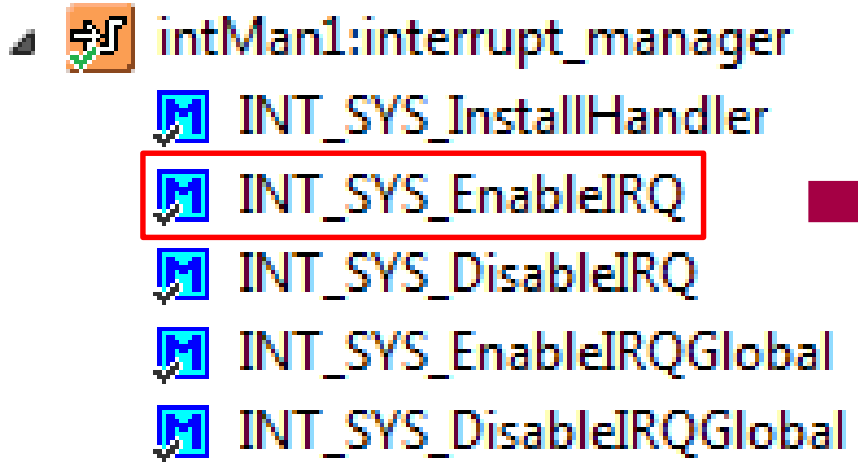



```
/* Write your local variable definition here */

/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! *
#ifdef PEX_RTOS_INIT
  PEX_RTOS_INIT();                 /* Initialization of the selected RTO
#endif
/*** End of Processor Expert internal initialization.                    *

  CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                 g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
  CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_AGREEMENT);

  PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

  FLEXCAN_DRV_Init(INST_CANCOM1, &canCom1_State, &canCom1_InitConfig0);

/*** Don't write any code pass this line, or it will be deleted during cod
/*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS compon
#ifdef PEX_RTOS_START
  PEX_RTOS_START();                /* Startup of the selected RTOS. Macr
#endif
/*** End of RTOS startup code.  ***/
/*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
for(;;) {
  if(exit_code != 0) {
    break;
```

S32K144 FlexCAN Lab: Buttons interrupt / 配置按键中断

- Enable IRQ for buttons PORTC

  使能PORTC的中断




```
CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
               g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_AGREEMENT);

PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

FLEXCAN_DRV_Init(INST_CANCOM1, &canCom1_State, &canCom1_InitConfig0);

INT_SYS_EnableIRQ(PORTC_IRQn);
```

Drag and drop **INT_SYS_EnableIRQ** from interrupt manager component.
Add ***PORTC_IRQn*** parameter to enable IRQ for **PORTC**
拖拽INT_SYS_EnableIRQ函数至Main函数，填入参数PORTC_IRQn

S32K144 FlexCAN Lab: Buttons interrupt / 配置按键中断

- Add **PORTC_IRQHandler** function in main.c file to get which button was pressed and to set the message that will be sent

新建PORTC_IRQHandler函数，当按键按下时，设置要发送的消息

```
void PORTC_IRQHandler(void)
{
    uint32_t buttonsPressed;
    buttonsPressed = PORT_HAL_GetPortIntFlag(PORTC);

    switch(buttonsPressed)
    {
        case (1 << 12):
                pMessage = message1;
                toSend = true;
                break;
        case (1 << 13):
                pMessage = message2;
                toSend = true;
                break;
        default:
            break;
    }

    PORT_HAL_ClearPortIntFlagCmd(PORTC);
}
```

S32K144 FlexCAN Lab: Adding global variables / 添加全局变量

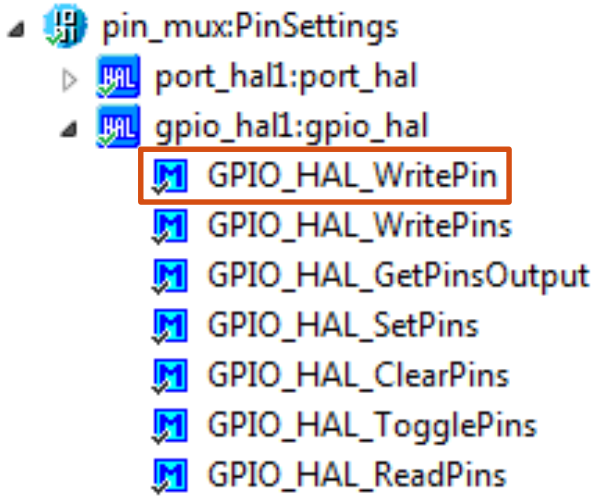
- Add global variables to **main.c** file, for storing messages and flags
  添加全局变量，存储要发送的消息，记录标志

```
const uint8_t message1[] = "Hello World!";
const uint8_t message2[] = "Hello Board!";

const uint8_t * pMessage;
volatile bool toSend;
volatile bool dataReceived;

uint8_t rxDataBuffer[16];
```

S32K144 FlexCAN Lab: Turning off LEDs / 关闭LED灯

- Add function calls to GPIO in **main** function, to turn off both LEDs

  在Main函数中先关闭LED



```
INT_SYS_EnableIRQ(PORTC_IRQn);

GPIO_HAL_WritePin(PTD, 15, 1);
GPIO_HAL_WritePin(PTD, 16, 1);

while(1)
{

}
```

S32K144 FlexCAN Lab: CAN Mailboxes / 配置CAN消息盒子

- Declare **FlexCAN** data descriptor in the entry of **main**
- Define frame information : ID size, FD support, data length

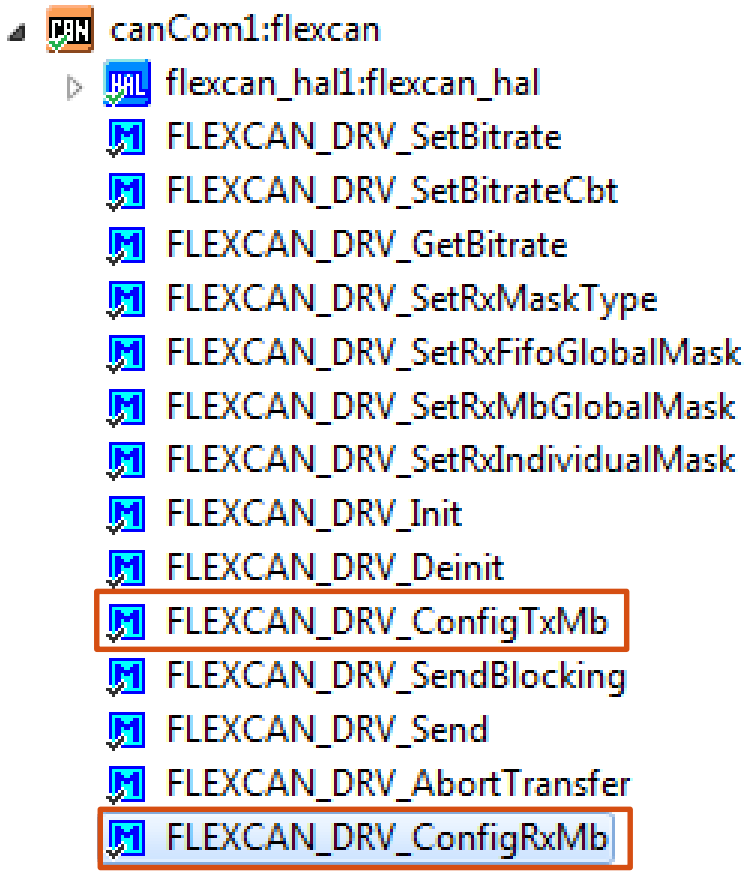
在Main函数里，声明FLexCAN数据描述符，并定义帧的信息

```
/* Write your local variable definition here */
  flexcan_data_info_t canDataInfo;
/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***
#ifdef PEX_RTOS_INIT
  PEX_RTOS_INIT();                 /* Initialization of the selected RTOS.
#endif
```

```
GPIO_HAL_WritePin(PTD, 15, 1);
GPIO_HAL_WritePin(PTD, 16, 1);
```

```
canDataInfo.msg_id_type = FLEXCAN_MSG_ID_STD;
canDataInfo.is_remote = false;
canDataInfo.enable_brs = true;
canDataInfo.fd_enable = true;
canDataInfo.fd_padding = 0U;
canDataInfo.data_length = sizeof(message1);
```

S32K144 FlexCAN Lab: CAN Mailboxes / CAN消息盒子




```
canDataInfo.msg_id_type = FLEXCAN_MSG_ID_STD;
canDataInfo.is_remote = false;
canDataInfo.enable_brs = true;
canDataInfo.fd_enable = true;
canDataInfo.fd_padding = 0U;
canDataInfo.data_length = sizeof(message1);

FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1,);
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1,);
```

- Drag and Drop **FLEXCAN_DRV_ConfigTxMb** and **FLEXCAN_DRV_ConfigRxMb** from the FlexCAN PEx component.

拖拽FLEXCAN_DRV_ConfigTxMb 、_ConfigRxMb至main函数

S32K144 FlexCAN Lab: CAN Mailboxes / CAN消息盒子

- Fill in the required parameters.
  填入所需参数
  - For Tx:
    - Mailbox number – **0U**
    - Message Descriptor - **&canDataInfo**
    - Message ID – **0U or IU**
  - For Rx:
    - Mailbox number – **1U**
    - Message Descriptor - **&canDataInfo**
    - Message ID – **1U or 0U**

```
FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, 0U, &canDataInfo, 0U);
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, 1U, &canDataInfo, 1U);
```

← Board 1

```
FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, 0U, &canDataInfo, 1U);
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, 1U, &canDataInfo, 0U);
```

← Board 2

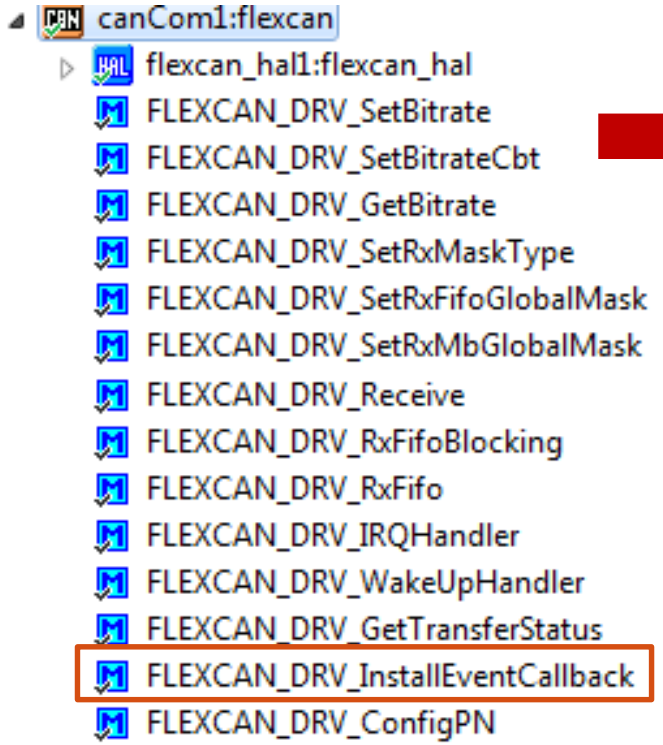
S32K144 FlexCAN Lab: Defining FlexCAN callback / 定义回调函数

- Define FlexCAN callback. In our case, the only event action needed is to set the ***dataReceived*** flag when RX is complete.

定义FLexCAN回调函数，本例子使用了表示接收完成的标识符dataReceived作为事件。

```
void flexcan0_Callback(uint8_t instance,
                       flexcan_event_type_t eventType,
                       flexcan_state_t *flexcanState)
{
    (void)flexcanState;
    (void)instance;

    switch(eventType)
    {
    case FLEXCAN_EVENT_RX_COMPLETE:
        dataReceived = true;
        break;
    case FLEXCAN_EVENT_TX_COMPLETE:
        break;
    default:
        break;
    }
}
```

S32K144 FlexCAN Lab: Installing FlexCAN callback / 装载回调函数




```
FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, 0U, &canDataInfo, 0U);
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, 1U, &canDataInfo, 1U);

FLEXCAN_DRV_InstallEventCallback(INST_CANCOM1,);
```

Drag and Drop
FLEXCAN_DRV_InstallEventCallback
from the FlexCAN PEx component.
拖拽FLEXCAN_DRV_InstallEventCallback至Main函数

- Add the required parameters to the method:
  - Callback – callback defined previous : **flexcan0_Callback**
  - User Paremeters – Not required in this lab : **NULL**

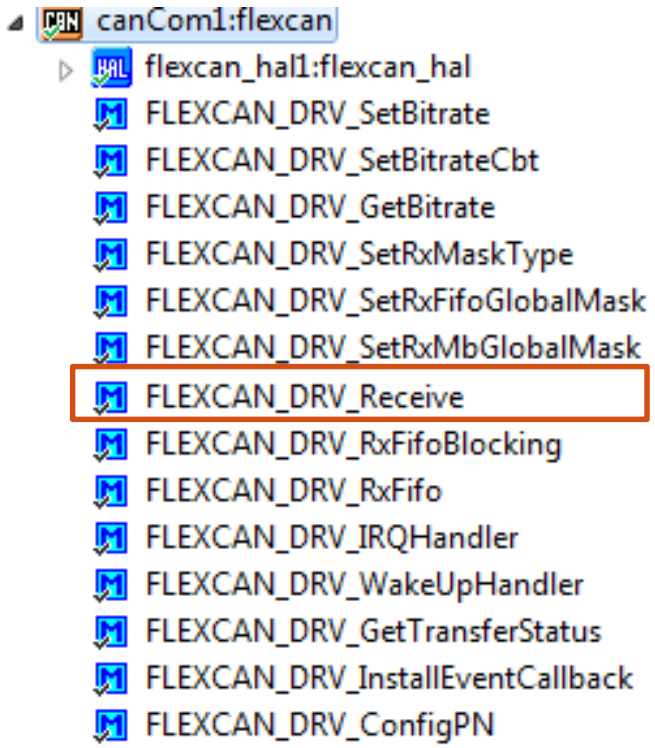
装载回调函数，本实验无需用户参数

```
FLEXCAN_DRV_InstallEventCallback(INST_CANCOM1, flexcan0_Callback, NULL);
```

S32K144 FlexCAN Lab: Starting initial Receive / 接收初始化

- Prior to the infinite loop, we need to call **FLEXCAN_DRV_Receive** to start the receive process.

  在循环之前，需先调用FlexCAN_DRV_Receive函数来开始CAN接收流程




```
FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, 0U, &canDataInfo, 0U);
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, 1U, &canDataInfo, 1U);

FLEXCAN_DRV_InstallEventCallback(INST_CANCOM1, flexcan0_Callback, NULL);

FLEXCAN_DRV_Receive(INST_CANCOM1,);
```

Drag and Drop
FLEXCAN_DRV_Receive
from the FlexCAN PEx component.
拖拽FLEXCAN_DRV_Receive函数

S32K144 FlexCAN Lab: Starting initial Receive / 接收初始化

- For the receive operation, we first need to allocate the required buffer.
  分配所需Buffer，来进行接收操作

```
/* Write your local variable definition here */
  flexcan_data_info_t canDataInfo;
  flexcan_msgbuff_t rxDataBuff;
/*** Processor Expert internal initialization. D
```

- Add parameters to **FLEXCAN_DRV_Receive** 添加参数
  - Mailbox number – **1U**
  - Data buffer – as defined before, **&rxDataBuffer**

Mailbox should be the same as previously configured

```
FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, 0U, &canDataInfo, 0U);
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, 1U, &canDataInfo, 1U);

FLEXCAN_DRV_InstallEventCallback(INST_CANCOM1, flexcan0_Callback, NULL);

FLEXCAN_DRV_Receive(INST_CANCOM1, 1U, &rxDataBuff);
```

S32K144 FlexCAN Lab: Infinite loop / 主循环

- If ***toSend*** flag is set, we must perform a send operation. This is done by calling **FLEXCAN_DRV_Send**.
- To avoid sending when it is not necessary, ***toSend*** flag must be set as **false**.

如tosend标识符为True，则开始发送操作，此时调用FLEXCAN_DRV_Send函数。之后再讲tosend置为False，避免不必要的发送。

```
while(1){
    if(toSend)
    {
        FLEXCAN_DRV_Send(INST_CANCOM1, 0U, &canDataInfo, 0U, pMessage);

        FLEXCAN_DRV_Send(INST_CANCOM1, 0U, &canDataInfo, 1U, pMessage);

        toSend = false;
    }
```

Board 1

Board 2

Mailbox number Descriptor Message ID Message

S32K144 FlexCAN Lab: Infinite loop / 主循环

- Composed from multiple blocks:
  - Copying Rx data to a local buffer
    拷贝Rx数据至本地Buffer
  - Comparing received data with expected strings
    将接收的数据和固定的字符串做比较
  - Restarting receive process
    重新开始接收流程

```
while(1)
  {
      if(toSend)
      {
          FLEXCAN_DRV_Send(INST_CANCOM1, 0U, &canDataInfo, 0U, pMessage);
          toSend = false;
      }
      if(dataReceived)
      {
          memcpy(rxDataBuffer, rxDataBuff.data, rxDataBuff.dataLen);
          if(strcmp((char *)rxDataBuffer, (char *)message1) == 0)
          {
              GPIO_HAL_TogglePins(PTD, (1 << 16));
          }
          else if(strcmp((char *)rxDataBuffer, (char *)message2) == 0)
          {
              GPIO_HAL_TogglePins(PTD, (1 << 15));
          }
          else
          {
              GPIO_HAL_WritePin(PTD, 15, 1);
              GPIO_HAL_WritePin(PTD, 16, 1);
          }
          FLEXCAN_DRV_Receive(INST_CANCOM1, 1U, &rxDataBuff);
          dataReceived = false;
      }
  }
```

S32K144 FlexCAN Lab: Comparing received data / 比较接收的数据

Add system includes to main
for **memcpy** and **strcmp**
Include系统头文件，
支持memcpy和strcmp语句

```
volatile int exit_code = 0;
/* User includes (#include be
#include <string.h>
#include <stdlib.h>
```

```
memcpy(rxDataBuffer, rxDataBuff.data, rxDataBuff.dataLen);
```

- Use **memcpy** function to copy data from FlexCAN buffer to local buffer
  使用memcpy函数将FlexCAN buffer中的数据拷贝至本地buffer中
- Fill the following parameters for **memcpy** call:
  填入以下参数
  - ***rxDataBuffer*** : Global buffer declared before as uint8_t
    rxDataBuffer: 已定义的uint8_t类型的全局buffer
  - ***rxDataBuff.data*** : Data retrieved by the CAN engine from the signals on CAN High and CAN Low
    rxDataBuff.data: 从CAN模块CAN高和CAN低的信号线中得到的数据
  - ***rxDataBuff.dataLen*** : Data length of the frame determined from the DLC
    rxDataBuff.dataLen : 通过DLC定义的帧长

S32K144 FlexCAN Lab: Comparing received data / 比较接收的数据

- Use **strcmp** function to compare received string with predefined messages.

  使用Strcmp函数比较两个字符串
- Three cases available:
  - Rx string is equal to message 1 – toggle green LED

    Rx数据和Message1相同-点亮绿色LED
  - Rx string is equal to message 2 – toggle RED LED

    Rx数据和Message2相同-点亮红色LED
  - Rx string is not equal to message 1 or message 2 – turn off both LEDs

    Rx数据和Message1/2都不相同-关闭两个LED

```
memcpy(rxDataBuffer, rxDataBuff.data, rxDataBuff.dataLen);
if(strcmp((char *)rxDataBuffer, (char *)message1) == 0)
{
    GPIO_HAL_TogglePins(PTD, (1 << 16));
}
else if(strcmp((char *)rxDataBuffer, (char *)message2) == 0)
{
    GPIO_HAL_TogglePins(PTD, (1 << 15));
}
else
{
    GPIO_HAL_WritePin(PTD, 15, 1);
    GPIO_HAL_WritePin(PTD, 16, 1);
}
```

S32K144 FlexCAN Lab: Restarting Receive / 重新接收

- Restart process needs to be started after the previous operation was finished.

  重新接收数据

- Use **FLEXCAN_DRV_Receive** to restart the process.

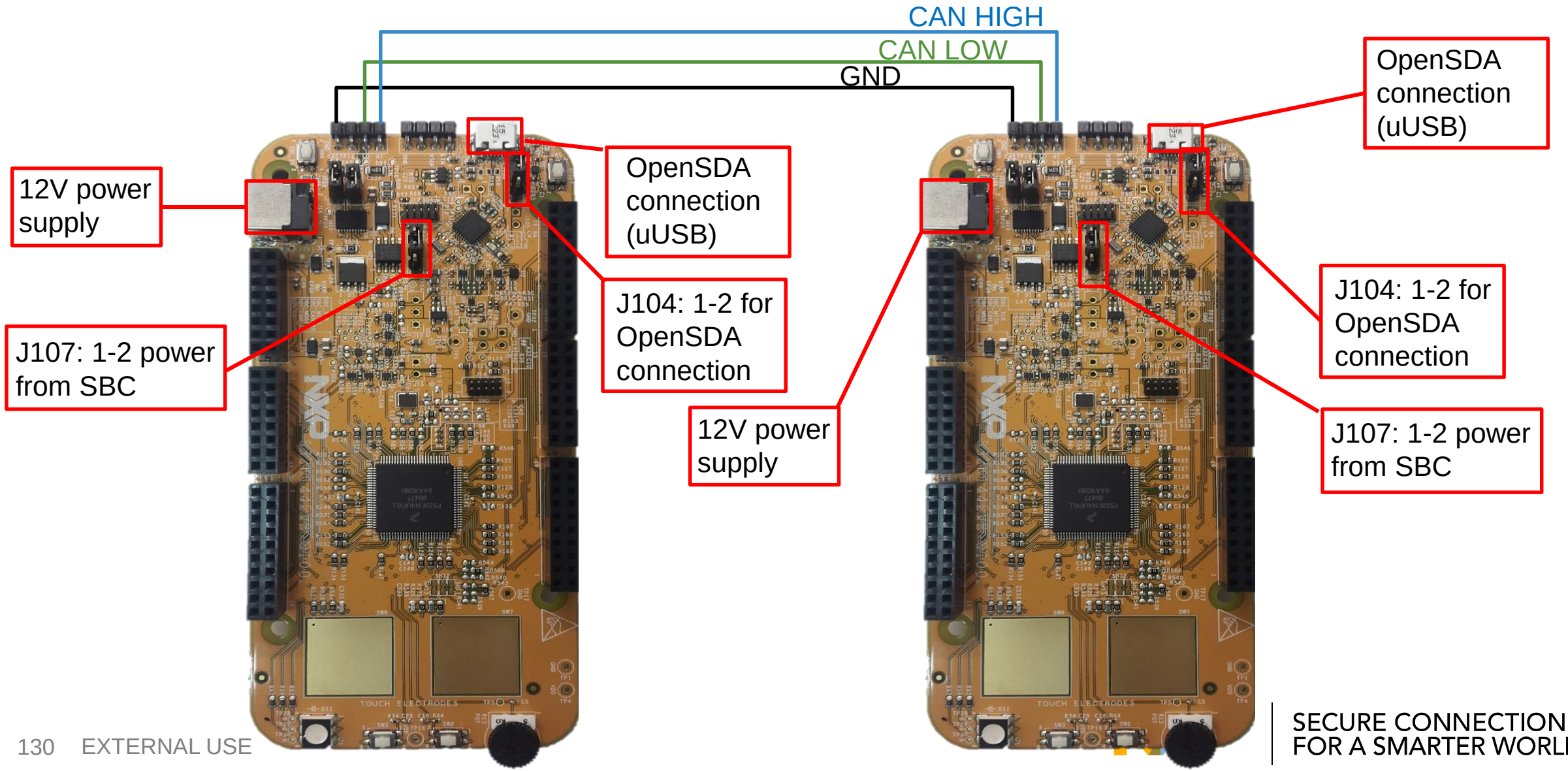
  使用FLEXCAN_DRV_Receive

- Also ***dataReceived*** flag needs to be cleared to avoid entering the branch when not required.

  清除*dataReceived*标志符，以免进入不必要的分支

```
while(1)
{
    if(toSend)
    {
        FLEXCAN_DRV_Send(INST_CANCOM1, 0U, &canDataInfo, 0U, pMessage);
        toSend = false;
    }
    if(dataReceived)
    {
        memcpy(rxDataBuffer, rxDataBuff.data, rxDataBuff.dataLen);
        if(strcmp((char *)rxDataBuffer, (char *)message1) == 0)
        {
            GPIO_HAL_TogglePins(PTD, (1 << 16));
        }
        else if(strcmp((char *)rxDataBuffer, (char *)message2) == 0)
        {
            GPIO_HAL_TogglePins(PTD, (1 << 15));
        }
        else
        {
            GPIO_HAL_WritePin(PTD, 15, 1);
            GPIO_HAL_WritePin(PTD, 16, 1);
        }
        FLEXCAN_DRV_Receive(INST_CANCOM1, 1U, &rxDataBuff);
        dataReceived = false;
    }
}
```

S32K144 FlexCAN Lab: Boards setup / 板子连接



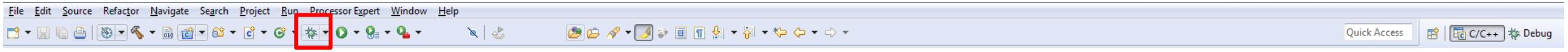
S32K144 FlexCAN Lab: Build and debug the lab / 编译调试工程

- Click on the build icon to make sure that there a no compiler errors.

  编译工程，确认没有报错

- Configure the debug configuration start a new debug session

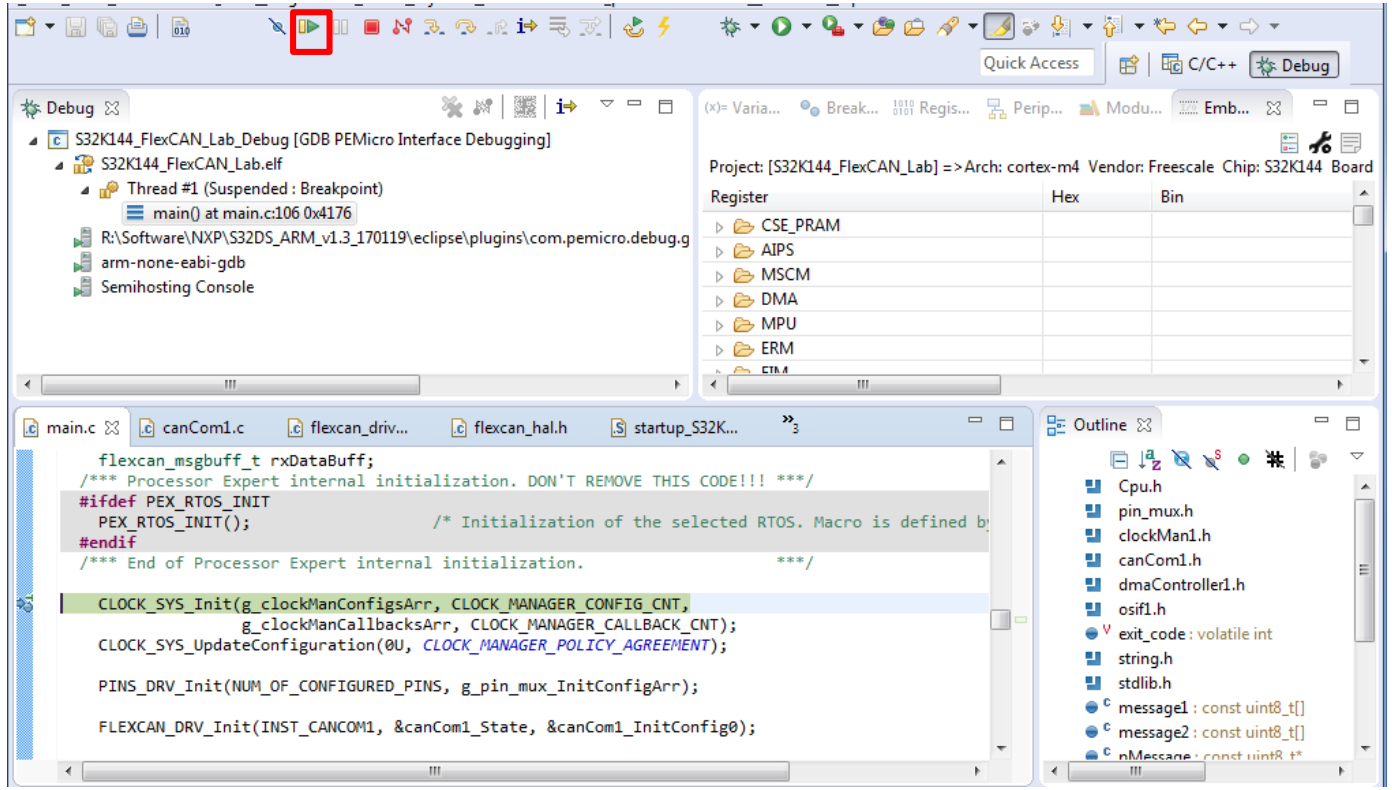
  配置调试口，开始调试



S32K144 FlexCAN Lab: Build and debug the lab / 编译调试工程

- In the debug perspective click the run icon to start the project.
- On buttons press, LEDs on the other boards will toggle.

  全速运行，按下按键，另一块板子上的相应的LED灯会亮起



Lab 05.

S32K144 ADC-FTM

- In this lab you will learn:

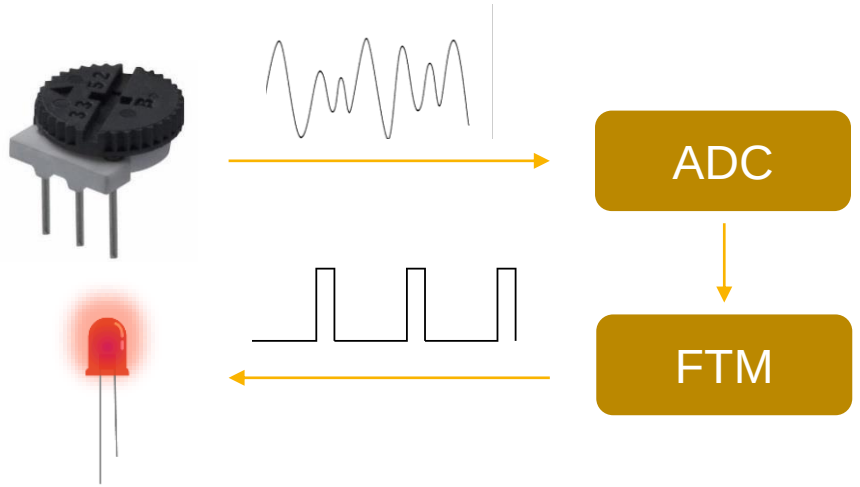
  本实验中，你将学习一下内容

  - How to use FTM peripheral in PWM edge aligned mode

    如何使用FTM模块产生边界对齐的PWM

  - How to use ADC peripheral with software trigger

    如何使用软件触发的ADC

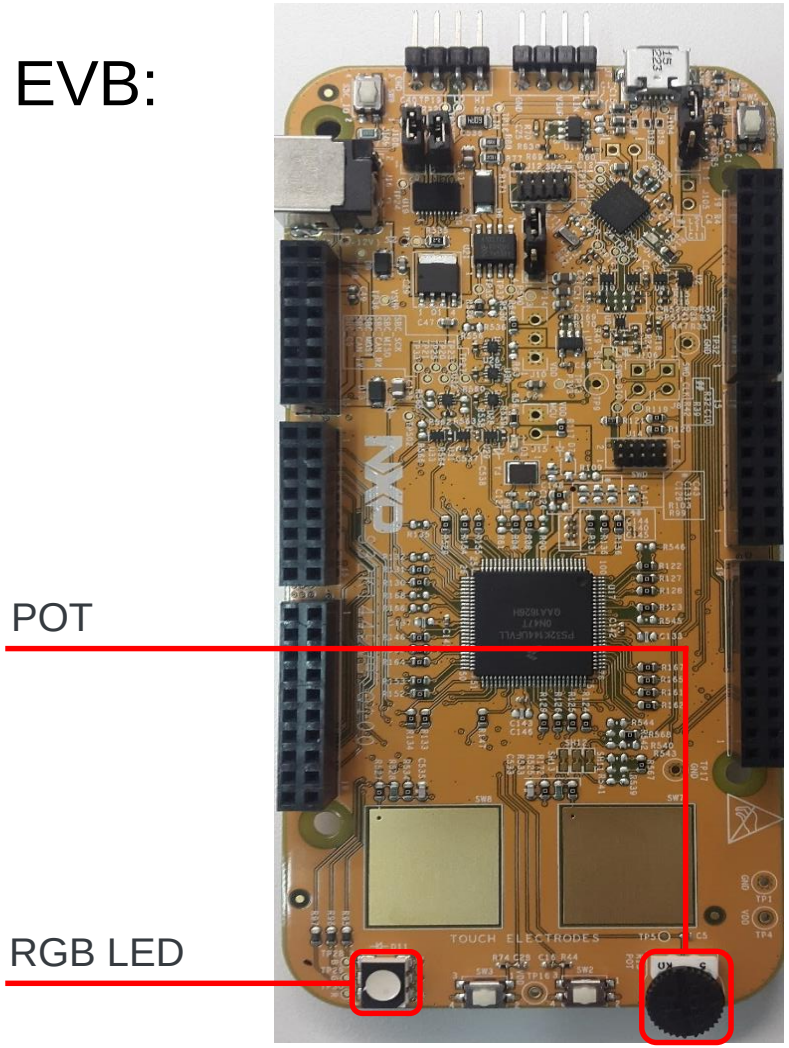


NXP SECURE CONNECTIONS FOR A SMARTER WORLD

S32K144 FTM-ADC Lab: Theory

- In this lab will be used the following components of the EVB:
  本实验中，你将使用以下EVB资源
  - RED LED – connected to FTM0 Channel 0
    红色LED，对应FTM0 Channel0
  - POTENTIOMETER – connected to ADC0 SE12
    电位器，对应ADC0 SE12

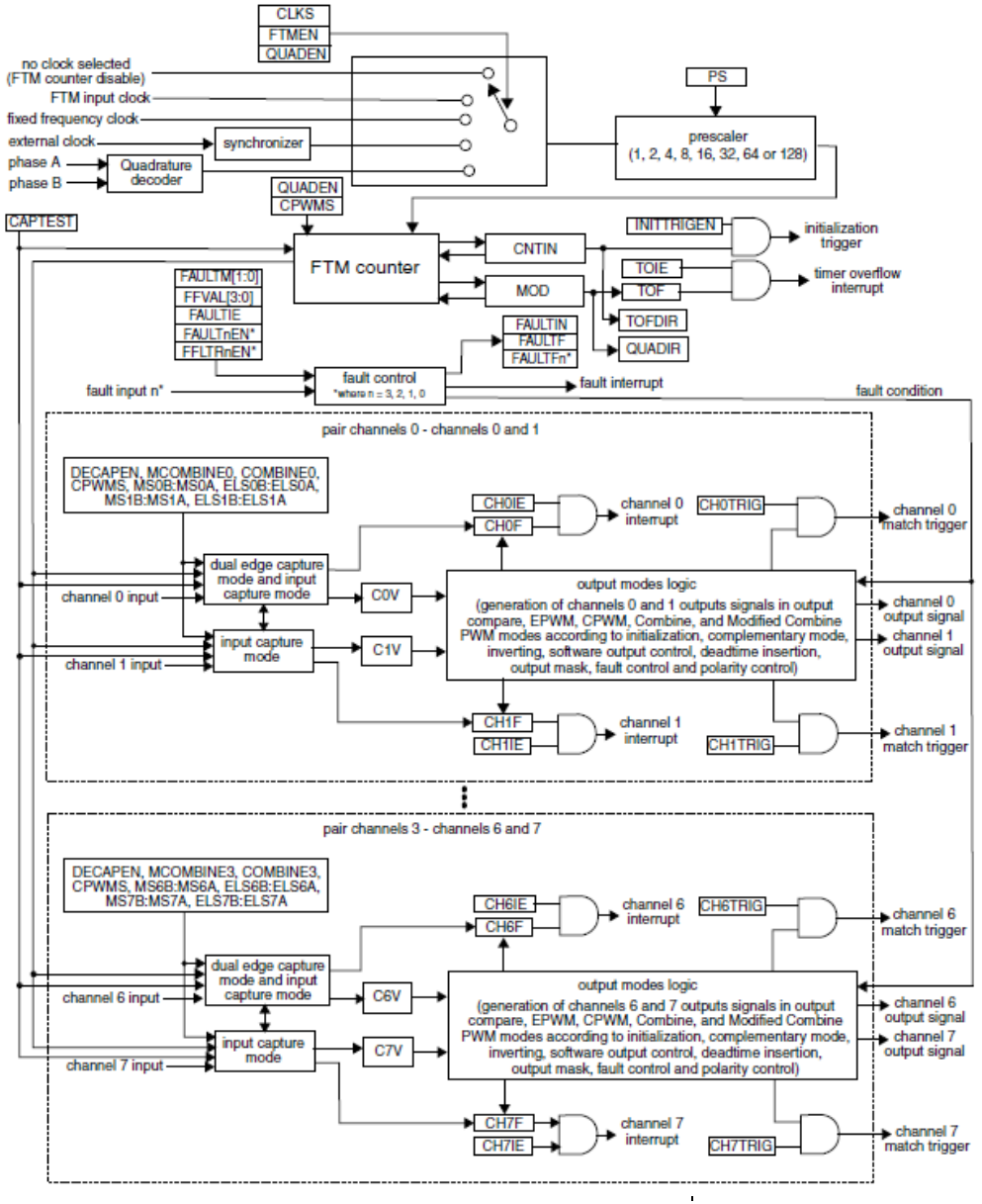
Component	S32K144 PIN
BLUE LED	PTD0
RED LED	PTD15
GREEN LED	PTD16
Potentiometer	PTC14 (ADC0_SE12)



S32K144 FTM-ADC Lab: FlexTimer

FTM(FlexTimer)

- One counter (16 bits) 16位计数器
- 8 channels 8个通道
- Configurable clock source 可配置的时钟源
- Combine channels mode 支持组合通道模式
- Fault control 错误控制
- Supported modes: 支持的模式
  - PWM (edge aligned, center aligned, combined modes)
    PWM (边界对齐，中心对齐，组合模式）
  - Input capture (single and dual edge input capture)
    输入捕获（单个沿和双个沿）
  - Output compare
    输出比较
  - Quadrature decoder
    正交解码
- Register synchronization to avoid glitch
  寄存器同步，避免毛刺



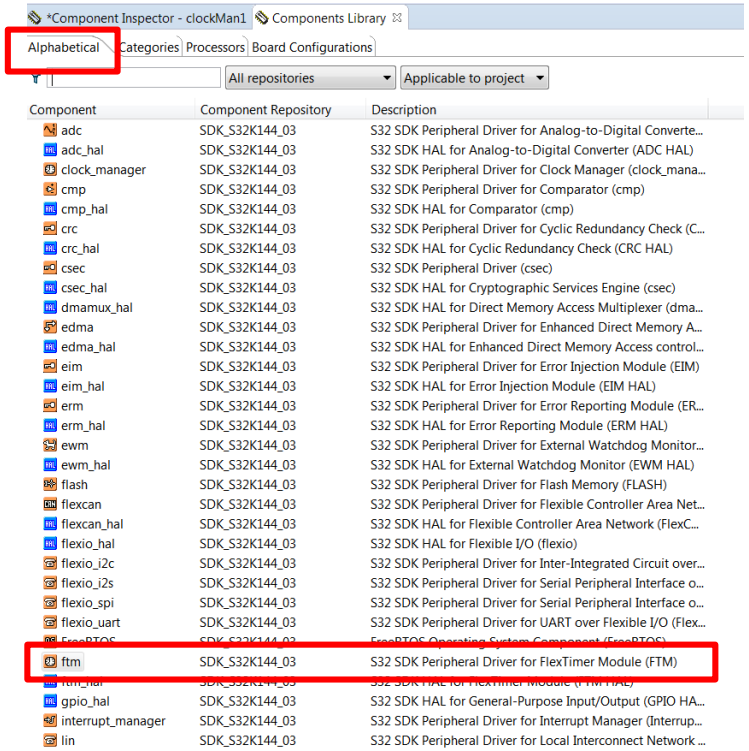
S32K144 FTM-ADC Lab: Clock and Pins / 配置时钟和管脚

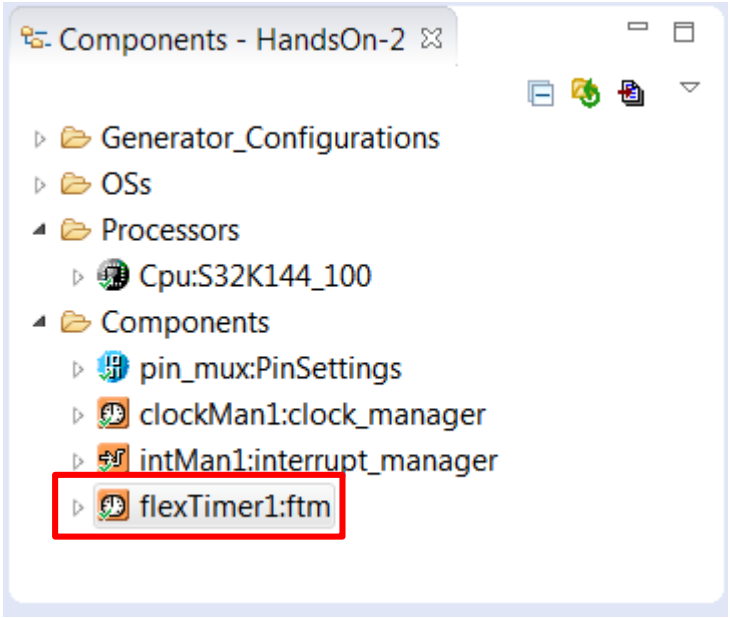
- Select **ClockMan** component in the Components Window
  选择ClockMan部件
- In the **Clock sources -> SOSC_CLK** field write 8000000(corresponding to 8MHz)
  SOSC_CLK设置为8MHz
- In the **Functional Clock-> SOSCDIVx_CLK->DIV1_CLK** select SOSC_CLK/1
  设置SOSCDIV1_CLK为SOSC_CLK/1
- In **Peripheral Clock** section **-> FTM0_CLK** check: enable
  使能FTM0的时钟
- In **Peripheral Clock** section **-> FTM0_CLK -> Functional Clock**: SOSCDIV1_CLK
  选择FTM0的功能时钟为SOSCFDIV1_CLK
- Select the pin_mux component in the Components window, select **FTM** tab inside the **Routing** tab
  选择pin_mux部件，选择FTM选项卡
- Go to **FTM0** and select **Channel 0**
  选择FTM0的Channel 0
- In the **Pin/Signal Selection** Colum, select **PTD15**.
  在Pin/Signal Selection列，选择PTD15
- In the **Direction** Colum, select **Output.**
  设置为Output

S32K144 FTM-ADC Lab: Configure FTM / 配置FTM

- Go to **Component Library** window. Select the **ftm** in the Alphabetical tab. Double click **ftm** to add to your project. FTM component should appear on the component window.

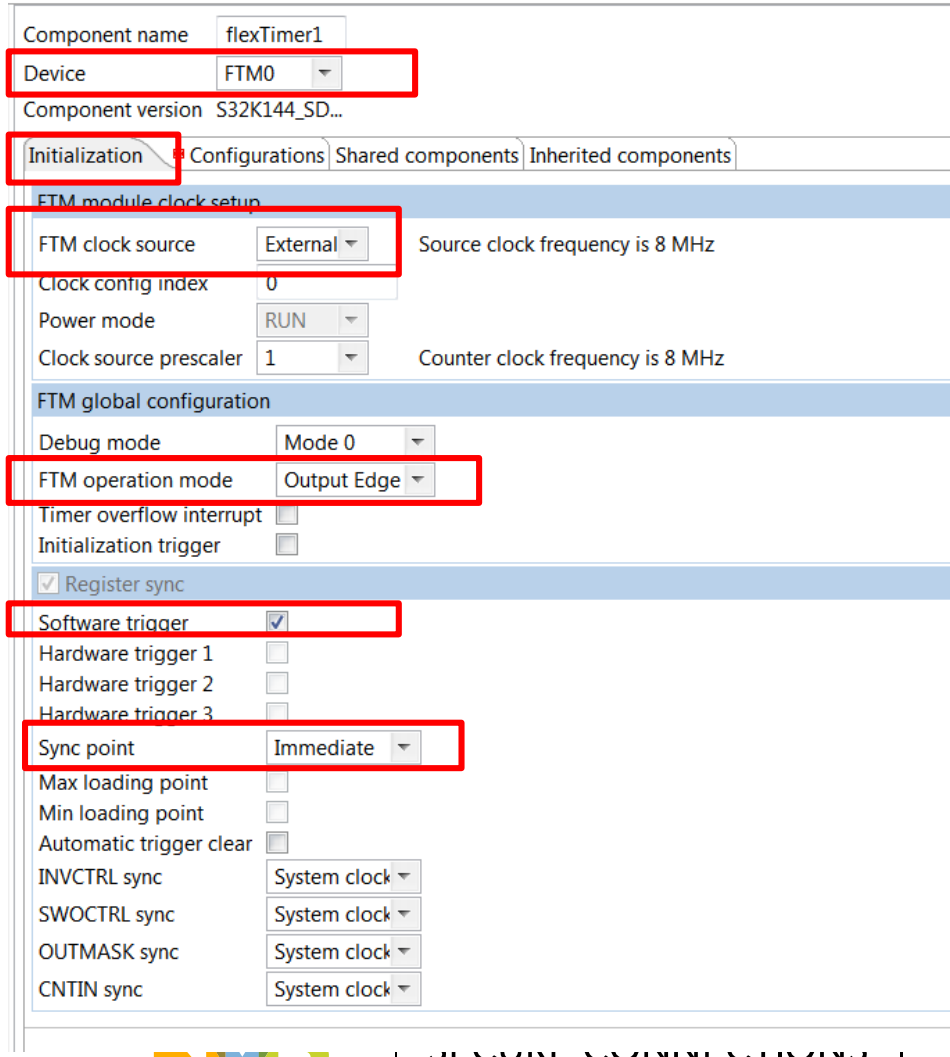
添加Ftm部件至当前工程





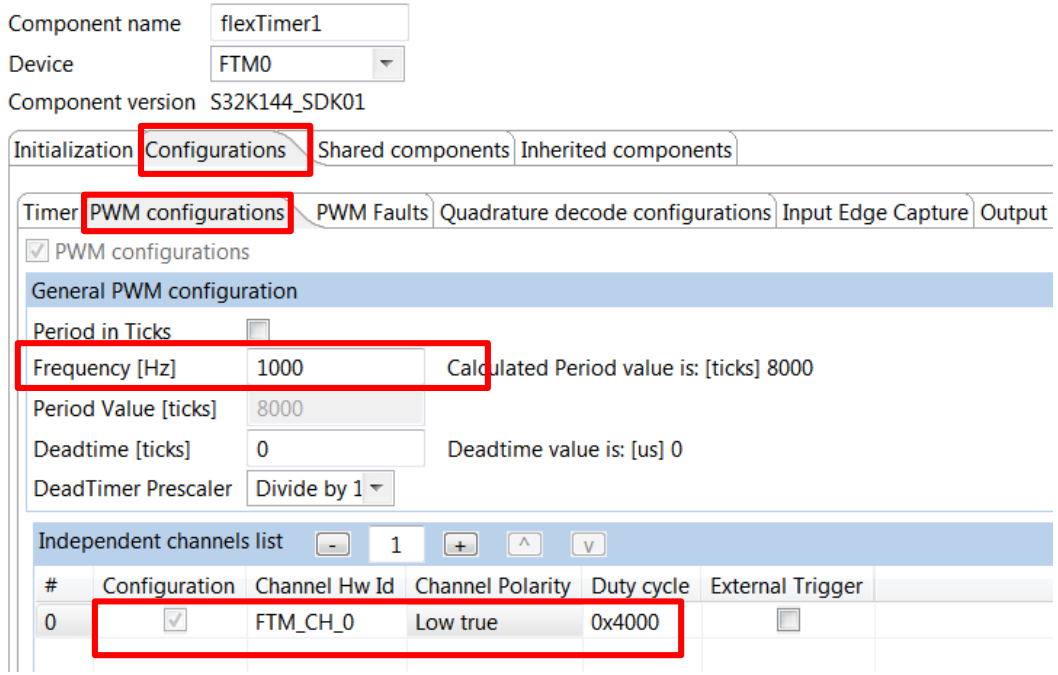
S32K144 FTM-ADC Lab: Configure FTM / 配置FTM

- Select the **ftm** component in the **Components** window
  选择FTM部件
- Select **Device**: **FTM0**
  选择FTM0模块
- Select **Initialization** tab
  选择Initialization标签栏
  - Select **FTM clock source**: **External**
  - 设置FTM时钟源为“External”
  - Select **FTM operation mode**: **Output Edge Aligned PWM**
  - 选择FTM模式为“边界对齐PWM输出”
  - Check **Software trigger**
  - 勾选Software Trigger触发模式
  - Select **Sync point**: **Immediate**
  - 选择Sync Point为“立即同步”



S32K144 FTM-ADC Lab: Configure FTM / 配置FTM

- Select **Configurations** tab
  选择Configuration标签栏
- Select **PWM configurations** tab
  选择PWM Configuration标签栏
- In **Frequency [Hz]** type: **1000**
  设置频率为1000
- In **Independent channels list** configure **FTM_CH_0** channel with **Duty cycle** 50% (**0x4000**).
  在Independent channels list设置FTM_CH_0通道的占空比为50%
- Set **Channel Polarity**: **Low True** (LED logic is inverted)
  设置通道的极性



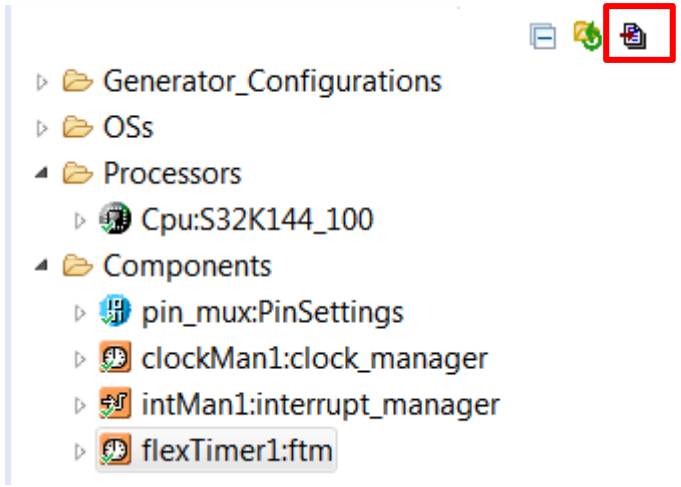
S32K144 FTM-ADC Lab: Generate the code / 代码生成

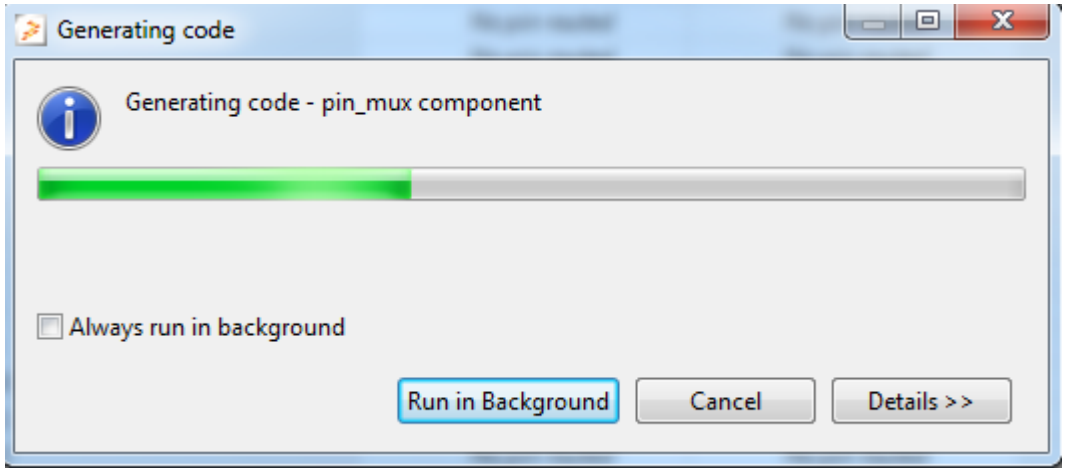
- To generate the code for the configuration select, click the **generate code** icon in the **Components** window.

  配置代码生成，点击Generate Code图标

- Wait for the code to be generated.

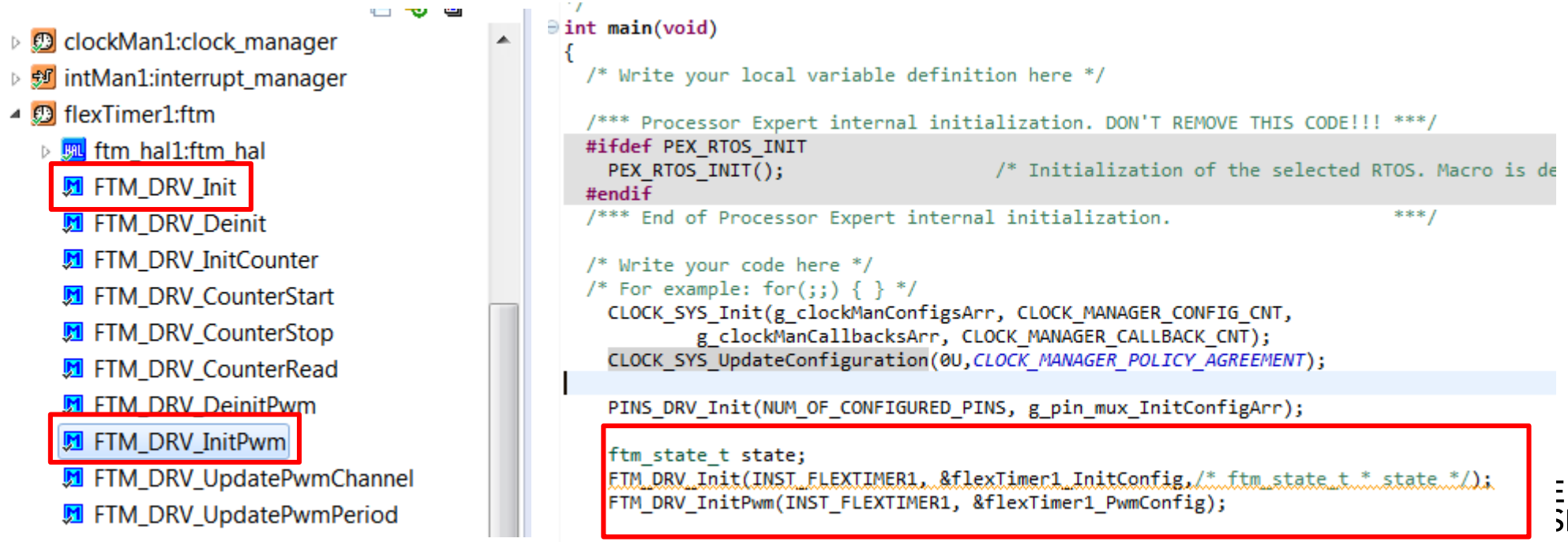
  等待进度条完成





S32K144 FTM-ADC Lab: Initialize FTM / 初始化FTM

- Declare state structure: **ftm_state_t state;**
  声明状态结构体：ftm_state_t state
- Expand the **ftm** component in the **Components** Window.
  展开ftm部件
  - Drag and drop the **FTM_DRV_Init** function inside the, into main, below the pins configuration
    拖拽FTM_DRV_Init函数至主函数，放在管脚配置函数之后
  - Drag and drop the **FTM_DRV_InitPwm** function inside the, into main, below **FTM_DRV_Init**
    拖拽FTM_DRV_InitPWM函数至主函数，放在FTM_DRV_Init函数之后



S32K144 FTM-ADC Lab: Initialize FTM / 初始化FTM

填入以下参数

- Add the configuration structure into the **FTM_DRV_Init**

```
FTM_DRV_Init(INST_FLEXTIMER1, &flexTimer1_InitConfig, &state);
```

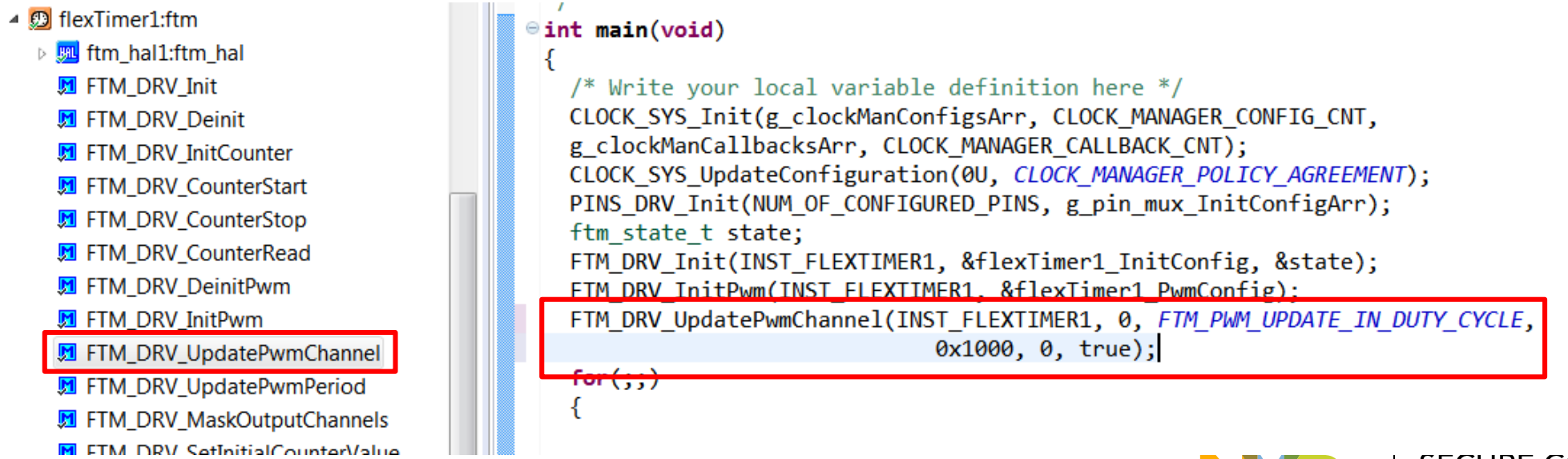
- Add the configuration structure into the **FTM_DRV_InitPwm**

```
FTM_DRV_InitPwm(INST_FLEXTIMER1, &flexTimer1_PwmConfig);
```

S32K144 FTM-ADC Lab: Change Duty Cycle / 修改占空比

- Expand the **ftm** component in the **Components** Window.
  展开Ftm部件
- Drag and drop the **FTM_DRV_UpdatePwmChannel** function inside the, into main, below **FTM_DRV_InitPwm**
  拖拽FTM_DRV_UpdatePwmChannel 至主函数，放在FTM_DRV_InitPwm之后
- Add the new Duty cycle and generate trigger into FTM_DRV_UpdatePwmChannel:
  添加新的占空比触发FTM_DRV_UpdatePwmChannel

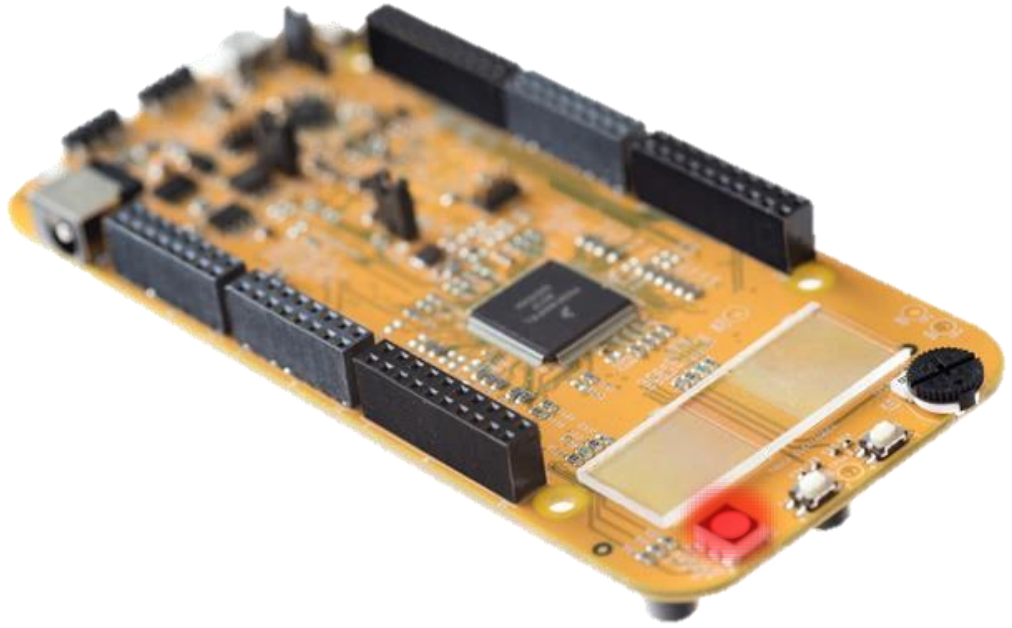
```
FTM_DRV_UpdatePwmChannel(INST_FLEXTIMER1, 0, FTM_PWM_UPDATE_IN_DUTY_CYCLE,0x1000, 0, true);
```



S32K144 FTM-ADC Lab: Test FTM code / 测试FTM代码是否正确

- Try to change the duty cycle value and check if the intensity of light is changing.

编译下载代码至S32K144，改变占空比，观察LED等亮度是否变化



S32K144 FTM-ADC Lab: ADC

- Linear successive approximation algorithm with up to 12-bit resolution

  线性逐次逼近算法，12位精度
- Up to 16 single-ended external analog inputs

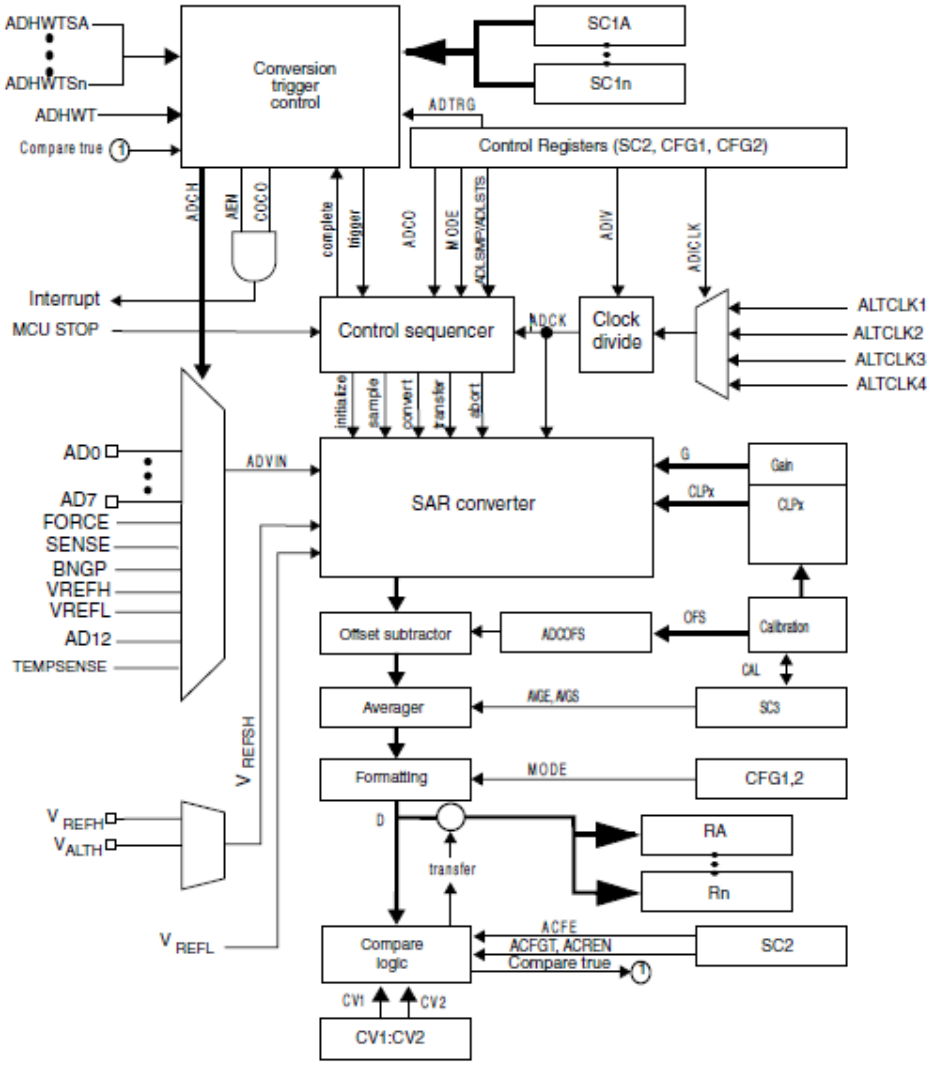
  最多支持16个外部单端模拟信号输入
- Hardware average function

  支持硬件均值
- Automatic compare

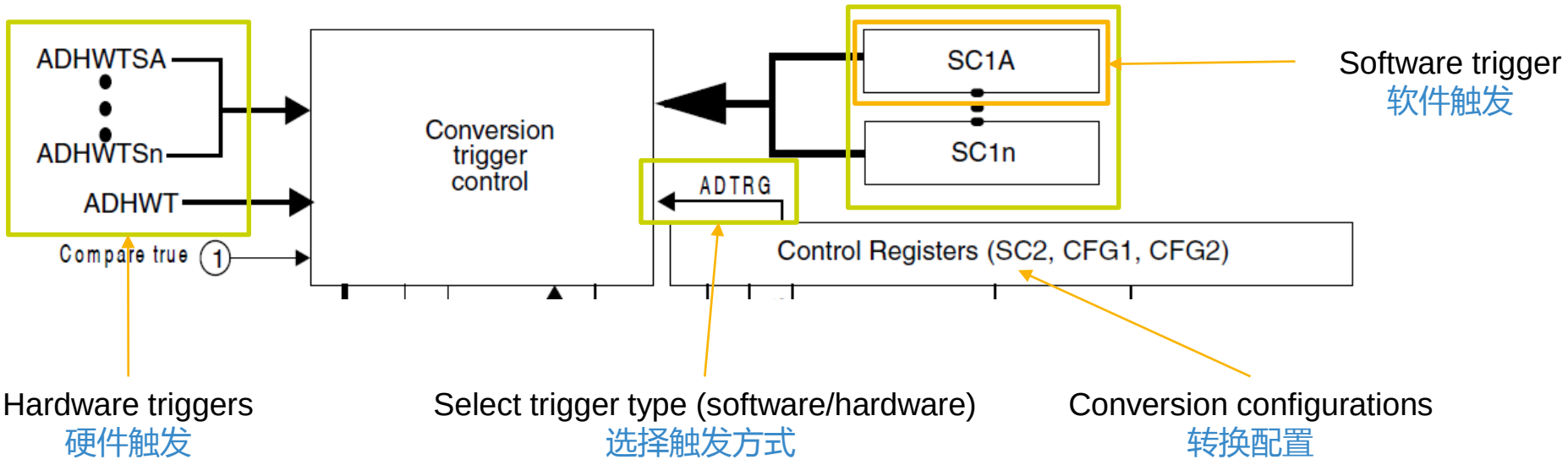
  自动比较
- Configurable sample time and conversion speed/power

  可配置的采样时长和转换速度/功耗
- Multiple trigger sources (hardware and software)

  多种触发源（硬件/软件触发）



S32K144 FTM-ADC Lab: ADC – Software trigger / 软件触发ADC



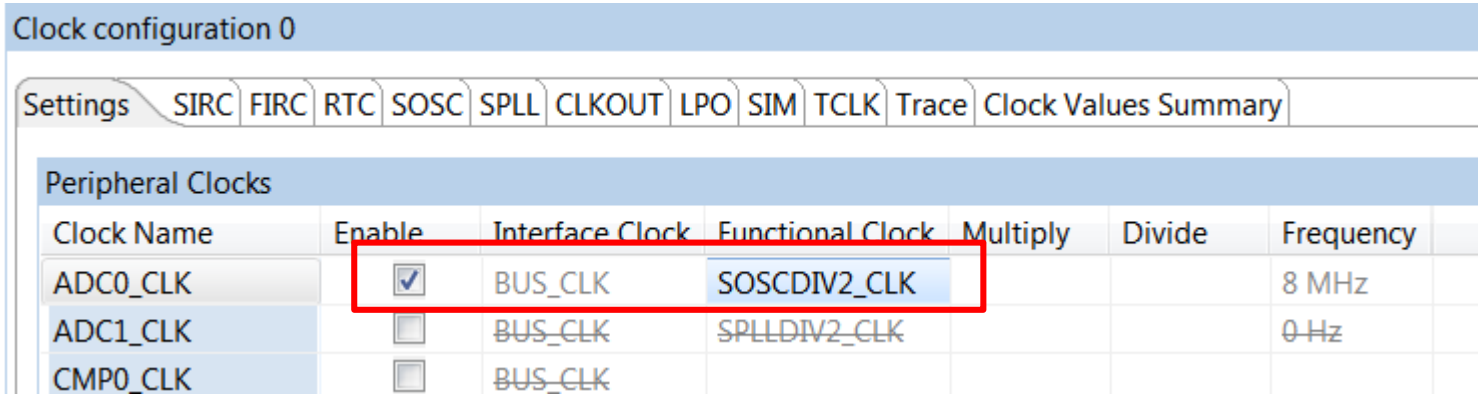
- Writing of SC1A will trigger the conversion using SC1A configuration if software trigger type is used. This is the only one method to generate a software trigger.

当使用软件触发时，写入SC1A寄存器，将会触发基于使用基于SC1A的配置的ADC转换。这是产生软件触发的唯一方式。

S32K144 FTM-ADC Lab: Enable ADC Clocks / 使能ADC时钟

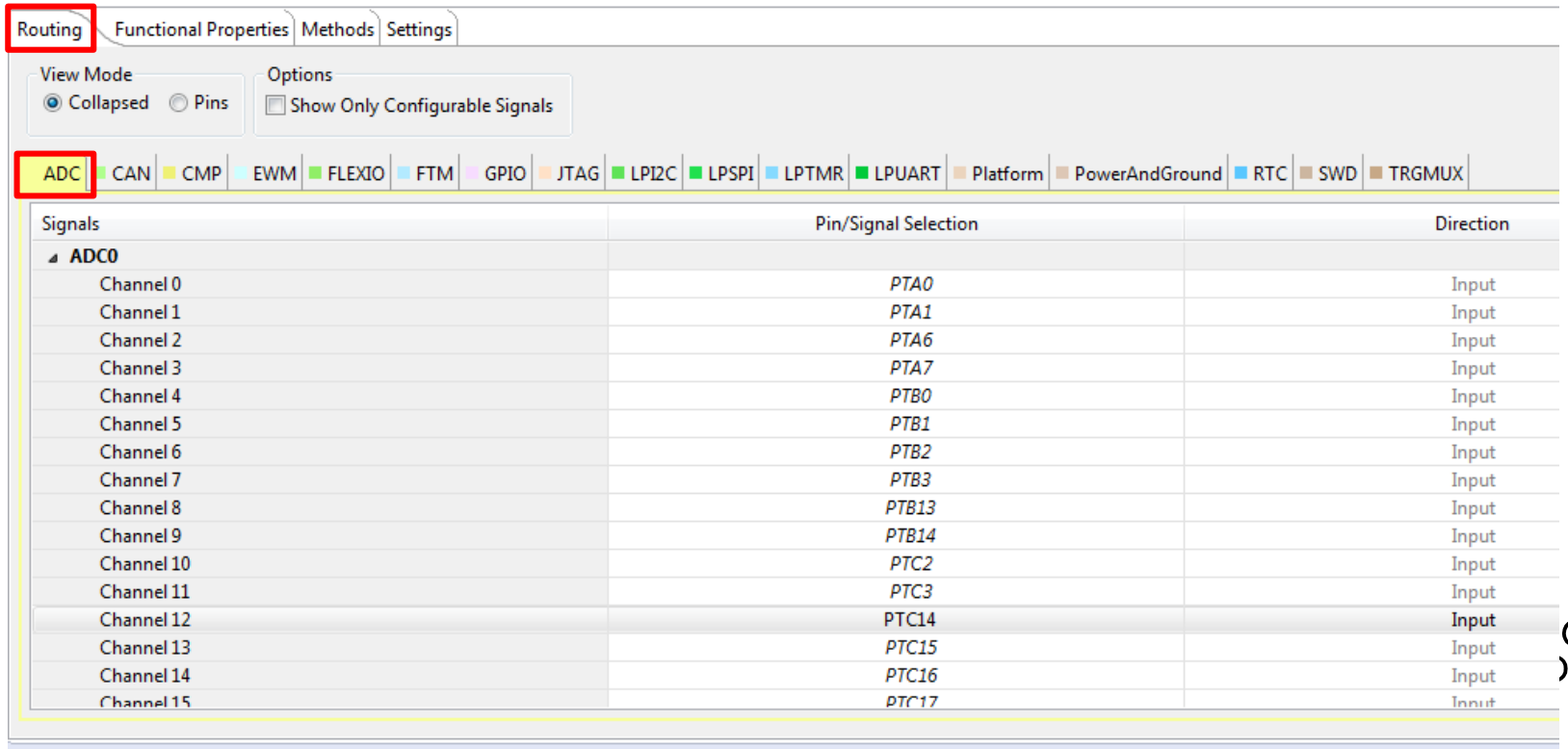
- Select the **clock_manager** component in the **Components** window.
- Go to **Settings** tab
- In **Peripheral Clock section** -> **ADC0_CLK** check: **enable**
- In **Peripheral Clock section** -> **ADC0_CLK -> Functional Clock: SOSCDIV1_CLK**

  使能ADC0的时钟，并将功能时钟设置为SOSCDIV1_CLK



S32K144 FTM-ADC Lab: Select I/O pins / 设置管脚

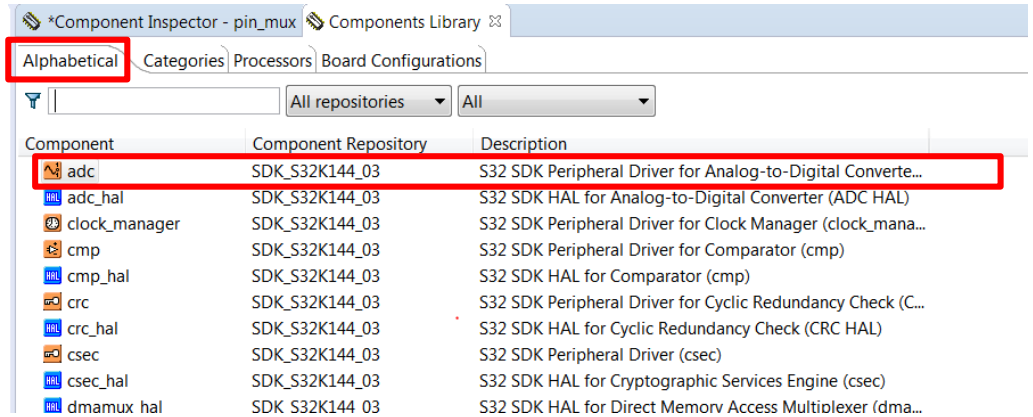
- In the **Component Inspector** window
- Select **ADC** tab inside the **Routing** tab
- Go to **ADC0** and select **Channel 12**
- In the **Pin/Signal Selection** Colum, select **PTC14**.
  将PTC14管脚配置成ADC0 Channel12

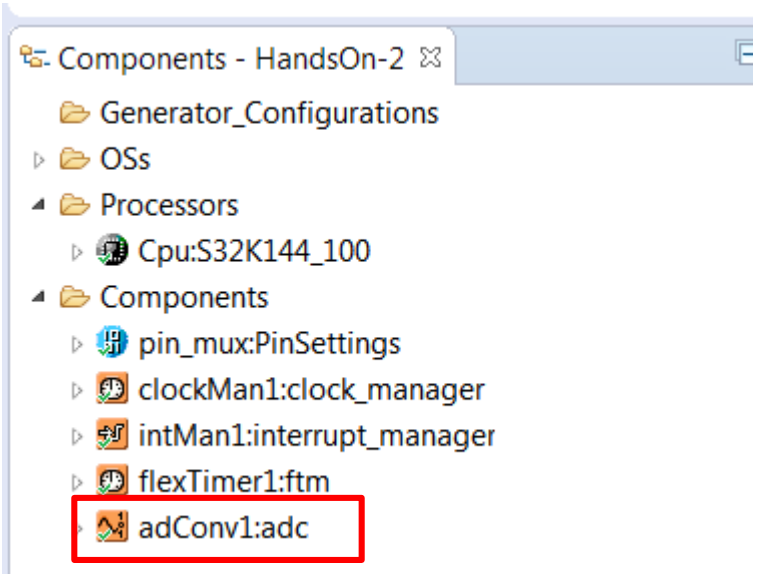


S32K144 FTM-ADC Lab: Configure ADC / 配置ADC

- Go to **Component Library** window.
- Select the **adc** in the Alphabetical tab.
- Double click **adc** to add to your project.
- ADC component should appear on the component window.

  添加ADC部件至当前工程





S32K144 FTM-ADC Lab: Configure ADC / 配置ADC

- Select **Device**: **ADC0**

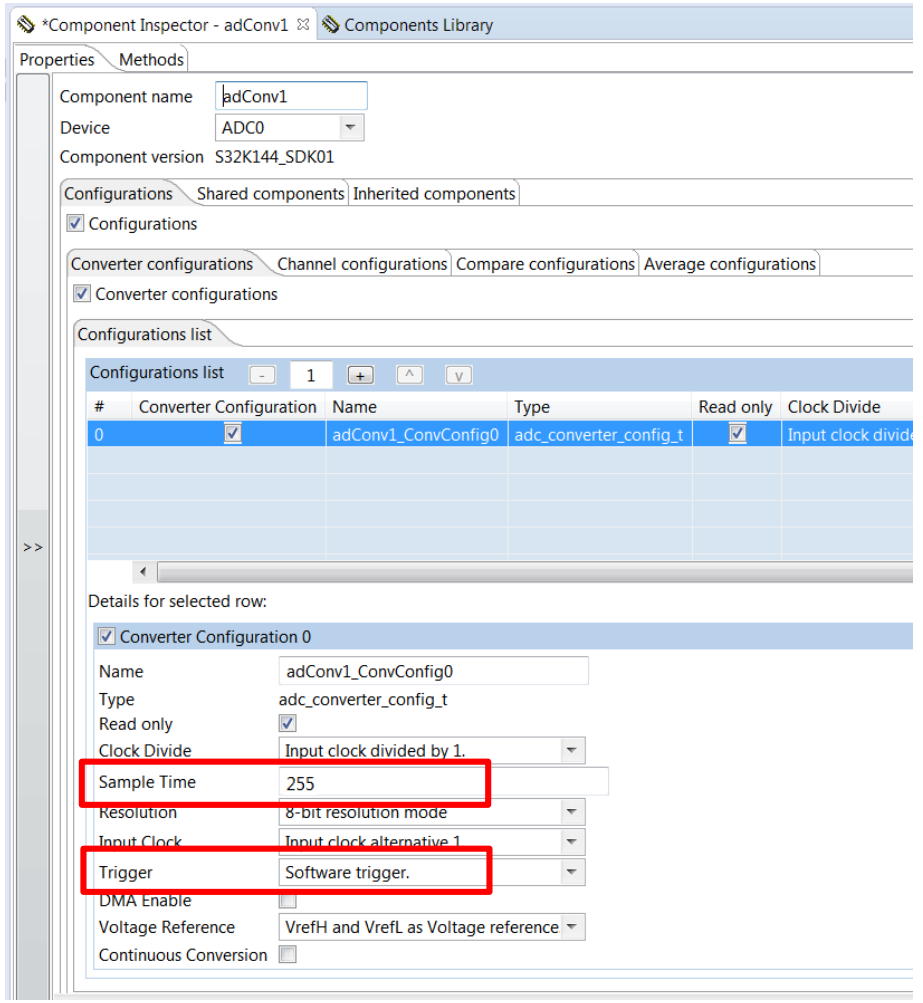
  选择ADC0

- Select **Configurations** tab and after that select **Converter Configurations** tab

  在Converter Configuration标签栏下选择Configuration子标签栏

  - Select **Trigger**: **Software trigger**
  - Select **Sample Time**: **255**

    设置触发类型、采样时长



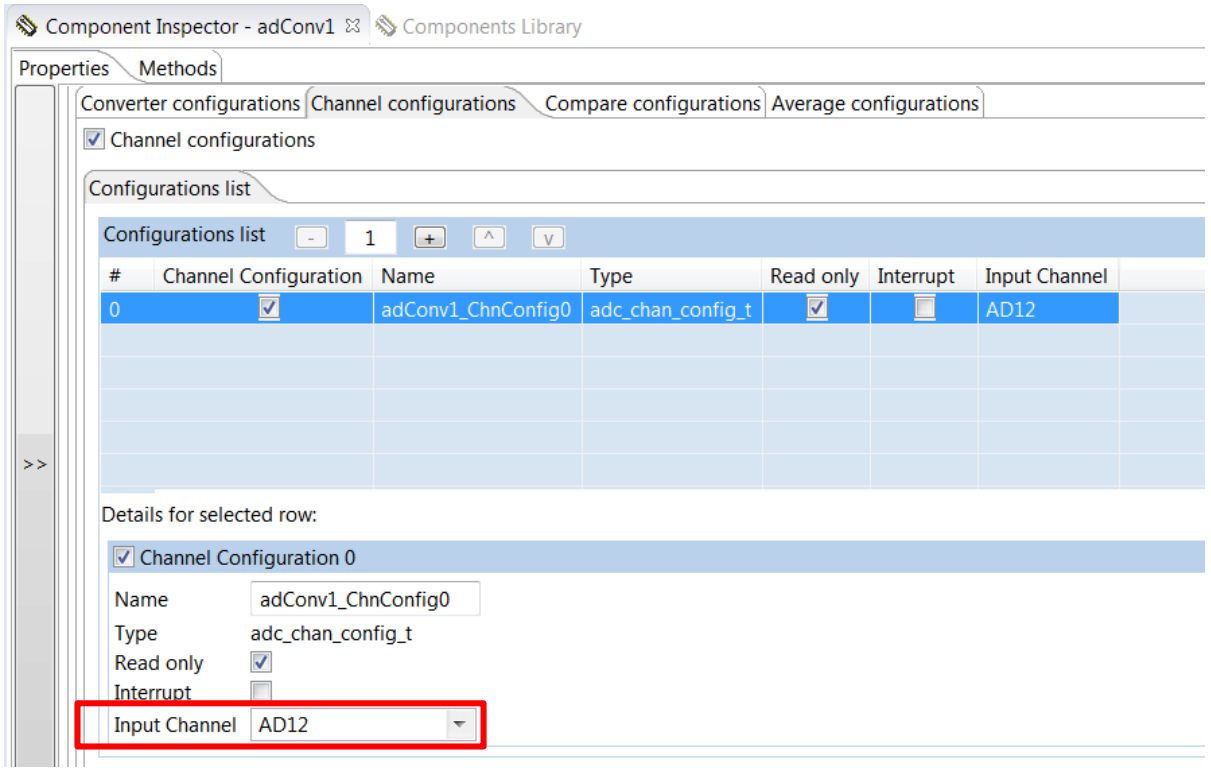
S32K144 FTM-ADC Lab: Configure ADC / 配置ADC

- Select **Configurations** tab and after that select **Channel configurations** tab

  在Converter Configuration标签栏下选择Channel Configuration子标签栏

- Select **Input Channel**: **AD12**

  设置输入通道：AD12



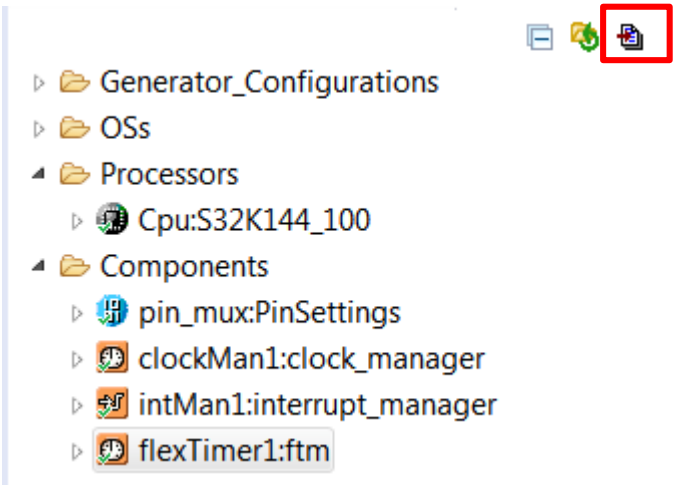
S32K144 FTM-ADC Lab: Generate the code / 代码生成

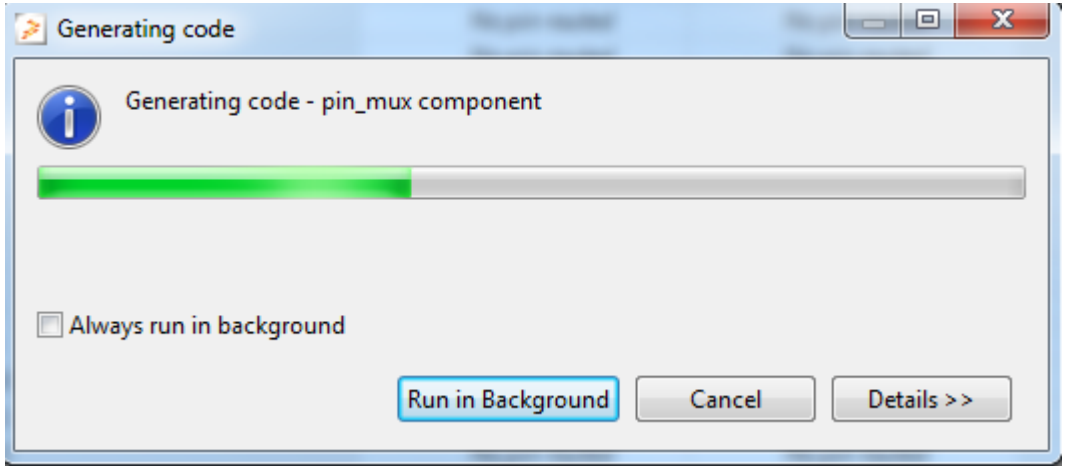
- To generate the code for the configuration select, click the **generate code** icon in the **Components** window.

  配置代码生成，点击Generate Code图标

- Wait for the code to be generated.

  等待进度条完成

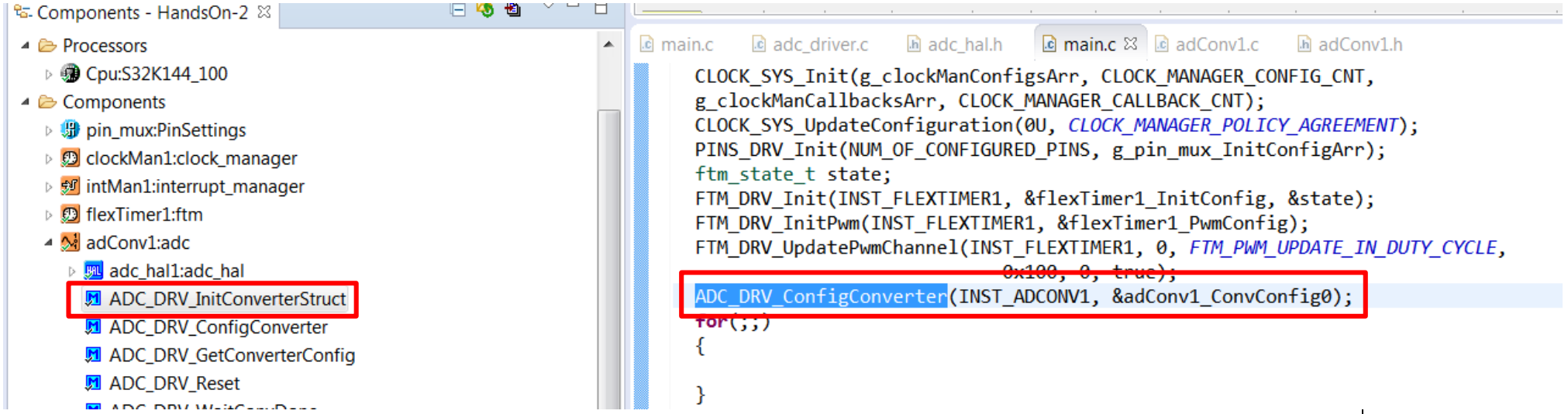




S32K144 FTM-ADC Lab: Configure ADC / 配置ADC

- Expand the **adc** component in the **Components** Window.
  展开adConv部件
- Drag and drop **ADC_DRV_ConfigConverter** function into main after PWM initialization.
  拖拽ADC_DRV_ConfigConverter函数至主函数，放在PWM初始化之后

```
ADC_DRV_ConfigConverter(INST_ADCONV1, &adConv1_ConvConfig0);
```



S32K144 FTM-ADC Lab: Configure ADC / 配置ADC

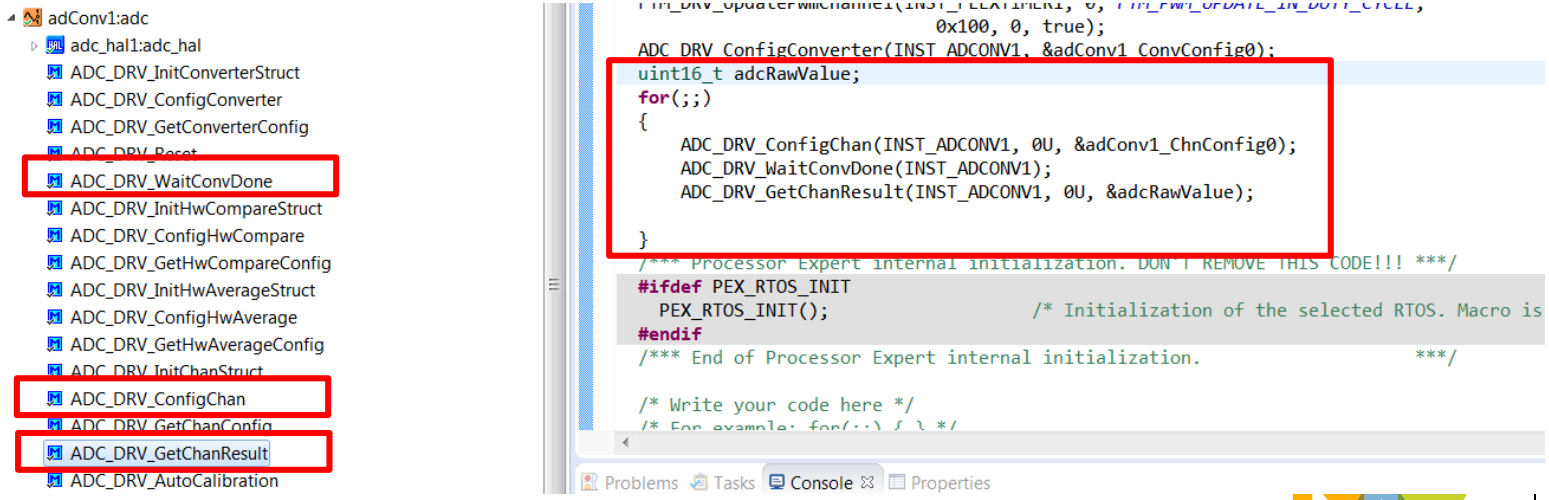
- Declare a uint16_t variable named **adcRawValue**

  声明一个类型为uint16_t的变量，命名为adcRawValue

- Drag and drop **ADC_DRV_ConfigChan**, **ADC_DRV_WaitConvDone** and **ADC_DRV_GetChanResult** function into an infinite loop.

  拖拽ADC_DRV_ConfigChan, ADC_DRV_WaitConvDone and ADC_DRV_GetChanResult至主循环

```
ADC_DRV_ConfigChan(INST_ADCONV1, 0U, &adConv1_ChnConfig0);
ADC_DRV_WaitConvDone(INST_ADCONV1);
ADC_DRV_GetChanResult(INST_ADCONV1, 0U, &adcRawValue);
```



S32K144 FTM-ADC Lab: Final step

- Update the duty cycle using **adcRawValue** and **FTM_DRV_UpdatePwmChannel** function.

  使用adcRawValue更新PWM的占空比

- Add the following function after you get the conversion result.

  在得到ADC转换值函数之后，添加如下函数

```
FTM_DRV_UpdatePwmChannel(INST_FLEXTIMER1, 0, FTM_PWM_UPDATE_IN_DUTY_CYCLE,
                                             adcRawValue, 0, true);
```

```
for(;;)
{
    ADC_DRV_ConfigChan(INST_ADCONV1, 0U, &adConv1_ChnConfig0);
    ADC_DRV_WaitConvDone(INST_ADCONV1);
    ADC_DRV_GetChanResult(INST_ADCONV1, 0U, &adcRawValue);
    FTM_DRV_UpdatePwmChannel(INST_FLEXTIMER1, 0, FTM_PWM_UPDATE_IN_DUTY_CYCLE,
            adcRawValue, 0, true);

}
```

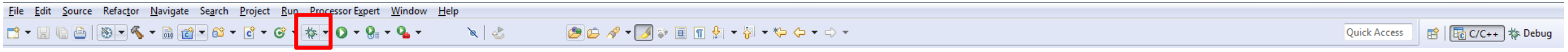
S32K144 FTM-ADC Lab: Build and debug the lab / 编译调试工程

- Click on the build icon to make sure that there a no compiler errors.

  编译工程，确认没有报错

- Configure the debug configuration start a new debug session

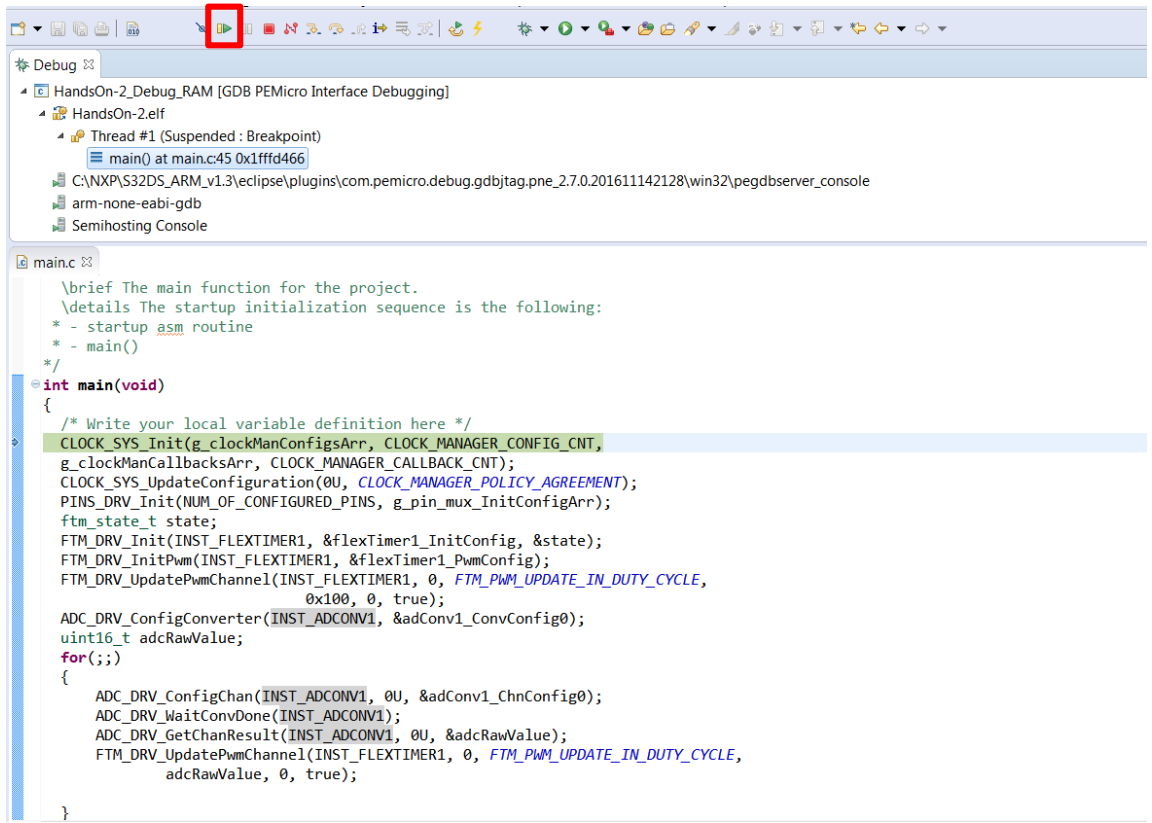
  配置调试口，开始调试

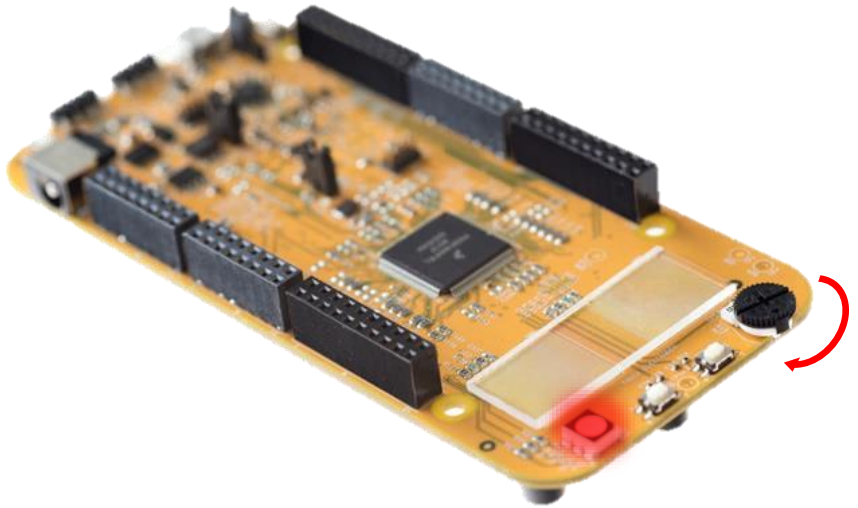


S32K144 FTM-ADC Lab: Build and debug the lab / 编译调试工程

- In the debug perspective click the run icon to start the project.
- Red LED intensity is changed by potentiometer.

全速运行，观察LED灯的亮度是否随旋转电位器而改变





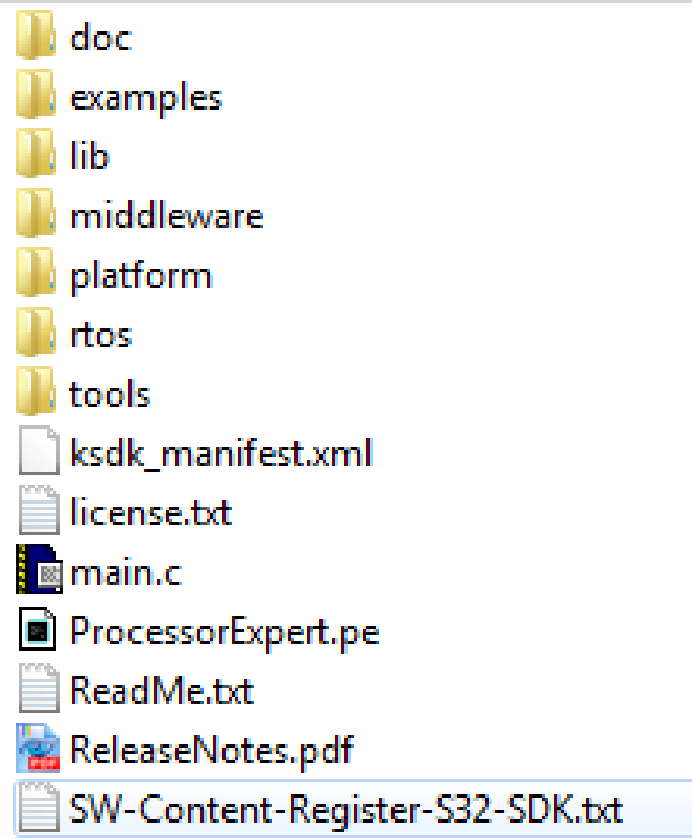
BACKUP

Documents fasten your design / SDK相关帮助文档

- Below materials may help you,

  以下资源能够帮助你学习S32K SDK

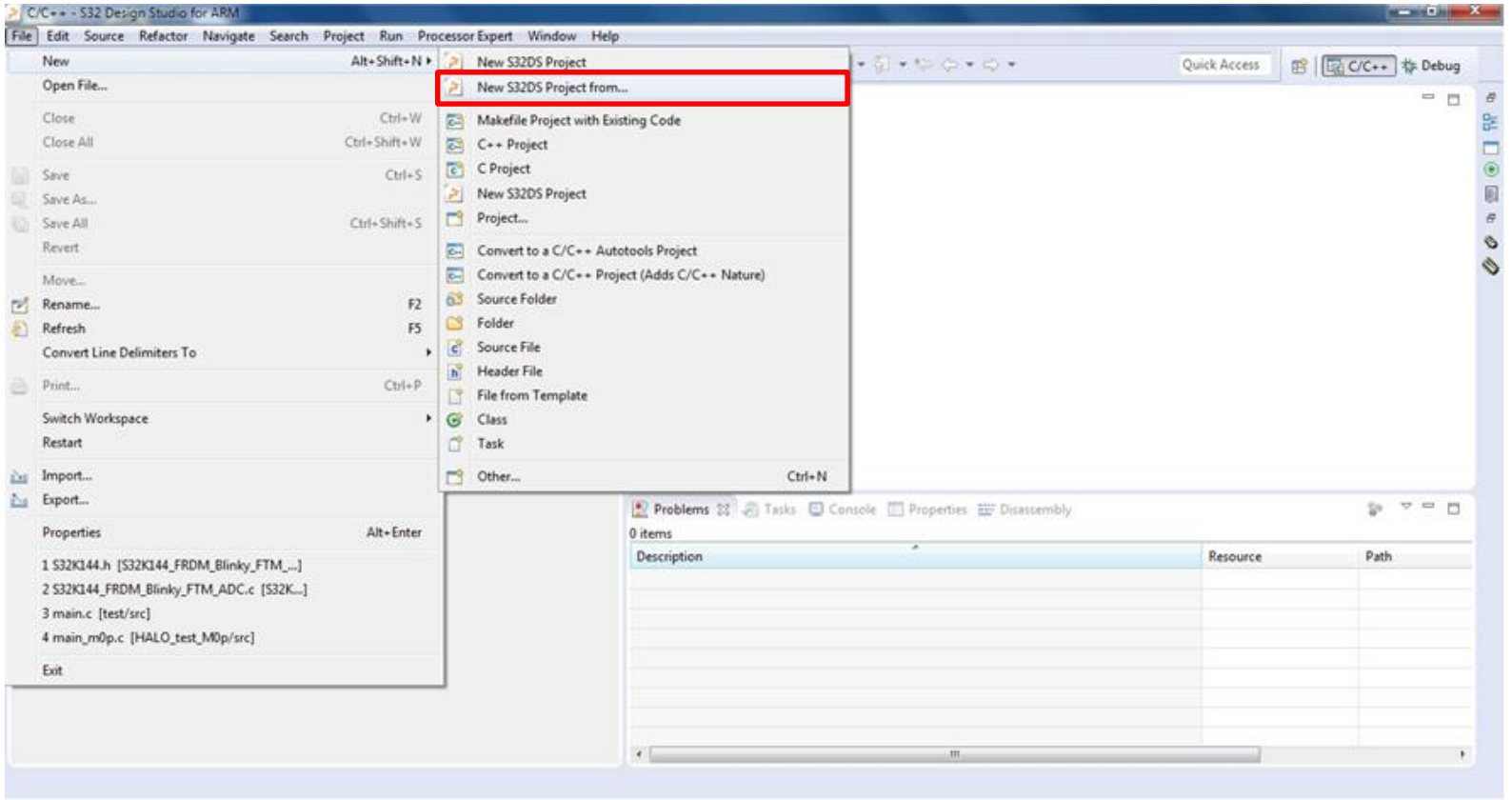
  - ..\S32_ARM_v1.3\S32DS\S32SDK_S32K144_RTM_1.0.0\
    - For driver manuals, see “doc”. 驱动文档
      - For quick start guide, see “**doc\S32SDKQSG.pdf**” 快速上手指南
      - For user manual and integration guidelines with S32DS 1.2, see "**doc\Start_here.html**". 使用手册和集成指南
      - For API details, see “**doc\S32_SDK_API_Documentation.pdf**”. API详细说明
    - For demos, see “**examples\demo_apps**”. Demo工程
    - For driver examples, see “**examples\driver_examples**”. 驱动样例工程

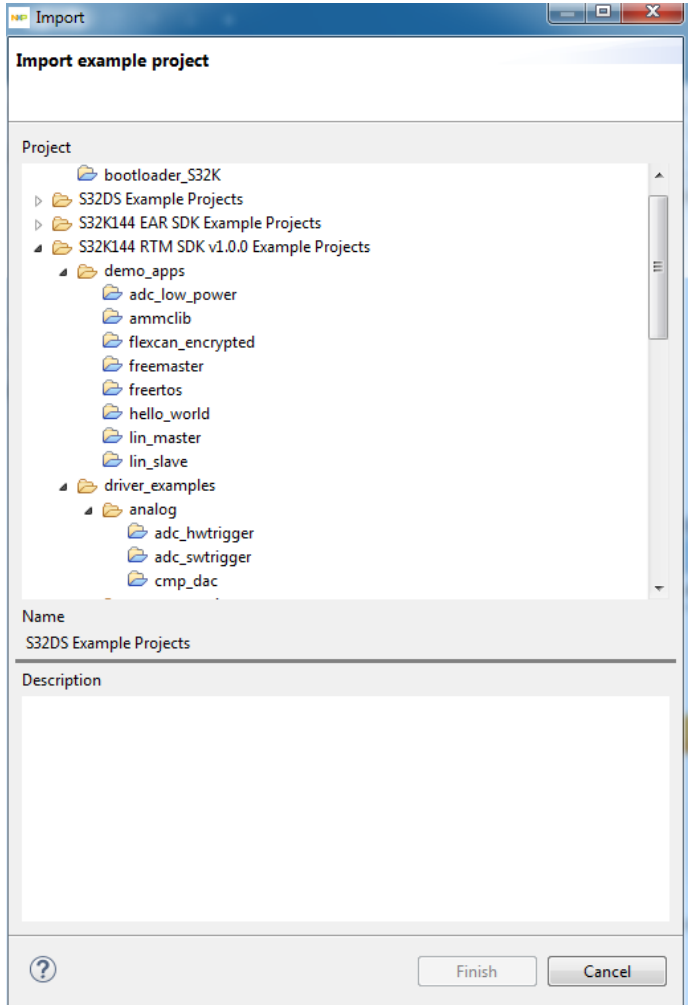


S32 SDK Examples and Drivers / 样例工程

- Import SDK examples to IDE

  导入样例工程





/**
 COPYRIGHT:

NXP Semiconductor, INC. All Rights Reserved. You are hereby granted a copyright license to use, modify, and distribute the SOFTWARE so long as this entire notice is retained without alteration in any modified and/or redistributed versions, and that such modified versions are clearly identified as such. No licenses are granted by implication, estoppel or otherwise under any patents or trademarks of NXP Semiconductor, Inc. This software is provided on an "AS IS" basis and without warranty.

To the maximum extent permitted by applicable law, NXP Semiconductor DISCLAIMS ALL WARRANTIES WHETHER EXPRESS OR IMPLIED, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY AGAINST INFRINGEMENT WITH REGARD TO THE SOFTWARE (INCLUDING ANY MODIFIED VERSIONS THEREOF) AND ANY ACCOMPANYING WRITTEN MATERIALS.

To the maximum extent permitted by applicable law, IN NO EVENT SHALL NXP Semiconductor BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OF THE USE OR INABILITY TO USE THE SOFTWARE.

NXP Semiconductor assumes no responsibility for the maintenance and support of this software

**/

SECURE CONNECTIONS
FOR A SMARTER WORLD