STC8A8K64D4 系列单片机
技术参考手册

技术支持网站：www.STCAI.com
资料更新日期：**2022/11/14**

目录

1 概述

STC8A8K64D4 系列单片机是不需要外部晶振和外部复位的单片机，是以超强抗干扰/超低价/高速/低功耗为目标的 8051 单片机，在相同的工作频率下，STC8A8K64D4 系列单片机比传统的 8051 约快 12 倍（速度快 11.2~13.2 倍），依次按顺序执行完全部的 111 条指令，STC8A8K64D4 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8A8K64D4 系列单片机是 STC 生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/强抗静电/较强抗干扰的新一代 8051 单片机，超级加密。指令代码完全兼容传统 8051。

MCU 内部集成高精度 R/C 时钟(±0.3%，常温下+25℃)，-1.38%～+1.42%温飘(-40℃~+85℃)，-0.88%～+1.05%温飘(-20℃~+65℃)。ISP 编程时 4MHz~45MHz 宽范围可设置（注意：温度范围为-40℃~+85℃时，最高频率须控制在 45MHz 以下），可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP 编程时 4 级复位门槛电压可选)。

MCU 内部有 3 个可选时钟源：内部高精度 IRC 时钟（ISP 下载时可进行调节）、内部 32KHz 的低速 IRC、外部 4M~33M 晶振或外部时钟信号。用户代码中可自由选择时钟源，时钟源选定后可再经过 8-bit 的分频器分频后再将时钟信号提供给 CPU 和各个外设（如定时器、串口、SPI 等）。

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.0mA（6MHz 工作频率）。STOP 模式即为主时钟停振模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外设都停止工作，功耗可降低到 0.6uA@Vcc=5.0V，0.4uA@Vcc=3.3V。

掉电模式可以使用 INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2) 、 T3(P0.4) 、 T4(P0.6) 、 RXD(P3.0/P3.6/P1.6/P4.3) 、 RXD2(P1.0/P4.0) 、 RXD3(P0.0/P5.0) 、RXD4(P0.2/P5.0)、CCP0(P1.7/P2.3/P7.0/P3.3)、CCP1(P1.6/P2.4/P7.1/P3.2)、CCP2(P1.5/P2.5/P7.2/P3.1)、CCP3(P1.4/P2.6/P7.3/P3.0)、I2C_SDA(P1.4/P2.4/P3.3)、SPI_SS(P1.2/P2.2/P3.5)以及所有端口的 I/O 中断、比较器中断、低压检测中断、掉电唤醒定时器唤醒。

MCU 提供了丰富的数字外设（串口、定时器、PCA、增强型 PWM 以及 I^2C、SPI）接口与模拟外设（速度高达 800K 即每秒 80 万次采样的 12 位*15 路超高速 ADC、比较器），可满足广大用户的设计需求。

STC8A8K64D4 系列单片机内部集成了增强型的双数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能。

产品线	I/O	UART	定时器	ADC	增强型 PWM	PCA	CMP	SPI	I2C	MDU16	I/O 中断	LCM	DMA
STC8A8K64D4 系列-64Pin/48Pin	59	4	5	15CH*12B	●	●	●	●	●	●	●	●	●

2 特性、价格及管脚

2.1 STC8A8K64D4-LQFP64/48/44，PDIP40 系列

2.1.1 特性及价格（有 16 位硬件乘除法器 MDU16，准 16 位单片机）

- 选型价格（不需要外部晶振、不需要外部复位）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	xdata，大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	MDU16 硬件 16 位乘除法器	8080/6800 LCM 彩屏接口驱动(8 位和 16 位)	DMA(支持 ADC，串口，SPI，LCM 等)	SPI	I²C	所有的 I/O 口均支持中断并可掉电唤醒	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	15 位增强型 PWM 满足无台灯光要求	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（24MHz 可调）	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	本身就可在线仿真	部分封装及含税价格						2021 年新品供货信息
																														LQFP64	QFN64 <8x8mm>	LQFP48	QFN48 <6x6mm>	LQFP44	PDIP40	
STC8A8K16D4	1.9-5.5	16K	8K	2	48K	59	4	有	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	√	√	√	√	√		现货
STC8A8K32D4	1.9-5.5	32K	8K	2	32K	59	4	有	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥4.5	√	¥3.9	√	¥3.9		
STC8A8K48D4	1.9-5.5	48K	8K	2	16K	59	4	有	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥4.5	√	¥3.9	√	¥3.9		
STC8A8K60D4	1.9-5.5	60K	8K	2	4K	59	4	有	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	√	√	√	√	√		
STC8A8K64D4	1.9-5.5	64K	8K	2	IAP	59	4	有	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	¥4.5	√	¥3.9	√	¥3.9		

- **内核**
  - ✓ 超高速 8051 内核（1T），比传统 8051 约快 12 倍以上
  - ✓ 指令代码完全兼容传统 8051
  - ✓ 43 个中断源，4 级中断优先级
  - ✓ 支持在线仿真
- **工作电压**
  - ✓ 1.9V～5.5V
  - ✓ 内建 LDO
- **工作温度**
  - ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）
- **Flash 存储器**
  - ✓ 最大 64K 字节 FLASH 程序存储器（ROM），用于存储用户代码
  - ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
  - ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
  - ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制
- **SRAM**
  - ✓ 128 字节内部直接访问 RAM（DATA）
  - ✓ 128 字节内部间接访问 RAM（IDATA）

- ✓ 8192 字节内部扩展 RAM（内部 XDATA）

➢ **时钟控制**

- ✓ 内部高精度 IRC（4MHz～45MHz，ISP 编程时可进行上下调整，还可以用户软件分频到较低的频率工作，如 100KHz）
  - ⌖ 误差±0.3%（常温下 25℃）
  - ⌖ -1.38%～+1.42%温漂（全温度范围，-40℃～85℃）
  - ⌖ -0.88%～+1.05%温漂（温度范围，-20℃～65℃）
- ✓ 内部 32KHz 低速 IRC（误差较大）
- ✓ 外部晶振（4MHz～45MHz）和外部时钟

➢ **复位**

- ✓ 硬件复位
  - ⌖ 上电复位，实测电压值为 1.69V～1.82V。（**在芯片未使能低压复位功能时有效**）
    上电复位电压由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到上电复位的下限门槛电压时，芯片处于复位状态；当电压从 0V 上升到上电复位的上限门槛电压时，芯片解除复位状态。
  - ⌖ 复位脚复位，出厂时 P5.4 默认为 I/O 口，ISP 下载时可将 P5.4 管脚设置为复位脚（**注意：当设置 P5.4 管脚为复位脚时，复位电平为低电平**）
  - ⌖ 看门狗溢出复位
  - ⌖ 低压检测复位，提供 4 级低压检测电压：2.0V（实测为 1.90V～2.04V）、2.4V（实测为 2.30V～2.50V）、2.7V（实测为 2.61V～2.82V）、3.0V（实测为 2.90V～3.13V）。
    每级低压检测电压都是由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到低压检测的下限门槛电压时，低压检测生效；当电压从 0V 上升到低压检测的上限门槛电压时，低压检测生效。
- ✓ 软件复位
  - ⌖ 软件方式写复位触发寄存器

➢ **中断**

- ✓ 提供 43 个中断源：INT0（支持上升沿和下降沿中断）、INT1（支持上升沿和下降沿中断）、INT2（只支持下降沿中断）、INT3（只支持下降沿中断）、INT4（只支持下降沿中断）、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、串口 3、串口 4、ADC 模数转换、LVD 低压检测、SPI、I2C、比较器、PCA/CCP/PWM、增强型 PWM、增强型 PWM 异常检测、所有的 I/O 中断（8 组）、LCD 驱动中断、串口 1 的 DMA 接收和发送中断、串口 2 的 DMA 接收和发送中断、串口 3 的 DMA 接收和发送中断、串口 4 的 DMA 接收和发送中断、SPI 的 DMA 中断、ADC 的 DMA 中断、LCD 驱动的 DMA 中断以及存储器到存储器的 DMA 中断。
- ✓ 提供 4 级中断优先级
- ✓ 时钟停振模式下可以唤醒的中断：INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、T3(P0.4)、T4(P0.6)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.0)、RXD3(P0.0/P5.0)、RXD4(P0.2/P5.0)、CCP0(P1.7/P2.3/P7.0/P3.3)、CCP1(P1.6/P2.4/P7.1/P3.2)、CCP2(P1.5/P2.5/P7.2/P3.1)、CCP3(P1.4/P2.6/P7.3/P3.0)、I2C_SDA(P1.4/P2.4/P3.3)、SPI_SS(P1.2/P2.2/P3.5)以及所有端口的 I/O 中断、比较器中断、低压检测中断、掉电唤醒定时器唤醒

➢ **数字外设**

- ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 4 个高速串口：串口 1、串口 2、串口 3、串口 4，波特率时钟源最快可为 FOSC/4
- ✓ 4 组 16 位 PCA 模块：CCP0、CCP1、CCP2、CCP3，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM

输出

- ✓ 8 组 15 位增强型 PWM，可实现带死区的控制信号，并支持外部异常检测功能，另外还有 4 组传统的 PCA/CCP/PWM 可作 PWM
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ I^2C：支持主机模式和从机模式
- ✓ **MDU16：**硬件 16 位乘除法器（支持 32 位除以 16 位、16 位除以 16 位、16 位乘 16 位、数据移位以及数据规格化等运算）
- ✓ **I/O 口中断：**所有的 I/O 均支持中断，每组 I/O 中断有独立的中断入口地址，所有的 I/O 中断可支持 4 种中断模式：高电平中断、低电平中断、上升沿中断、下降沿中断
  （本系列的 I/O 口中断可以进行掉电唤醒，且有 4 级中断优先级）
- ✓ **LCD 驱动模块：**支持 8080 和 6800 两种接口以及 8 位和 16 位数据宽度
- ✓ **DMA：**支持 SPI 移位接收数据到存储器、SPI 移位发送存储器的数据、串口 1/2/3/4 接收数据到的存储器、串口 1/2/3/4 发送存储器的数据、ADC 自动采样数据到存储器（同时计算平均值）、LCD 驱动发送存储器的数据、以及存储器到存储器的数据复制
- ✓ **硬件数字 ID：**支持 32 字节

➢ **模拟外设**

- ✓ 超高速 ADC，支持 **12 位高精度** 15 通道（通道 0～通道 14）的模数转换，**速度最快能达到 800K（每秒进行 80 万次 ADC 转换）**
- ✓ ADC 的通道 15 用于测试内部 1.19V 参考信号源（芯片在出厂时，内部参考信号源已调整为 1.19V）
- ✓ 比较器，一组比较器（比较器的正端可选择 CMP+端口、CMP+_2、CMP+_3 和所有的 ADC 输入端口，比较器的负端可选择 CMP-端口和内部 1.19V 的参考源，所以比较器可当作多路比较器进行分时复用）
- ✓ DAC：8 组增强型 PWM 定时器可当 8 路 DAC 使用、4 路 PCA 可当 4 路 DAC 使用

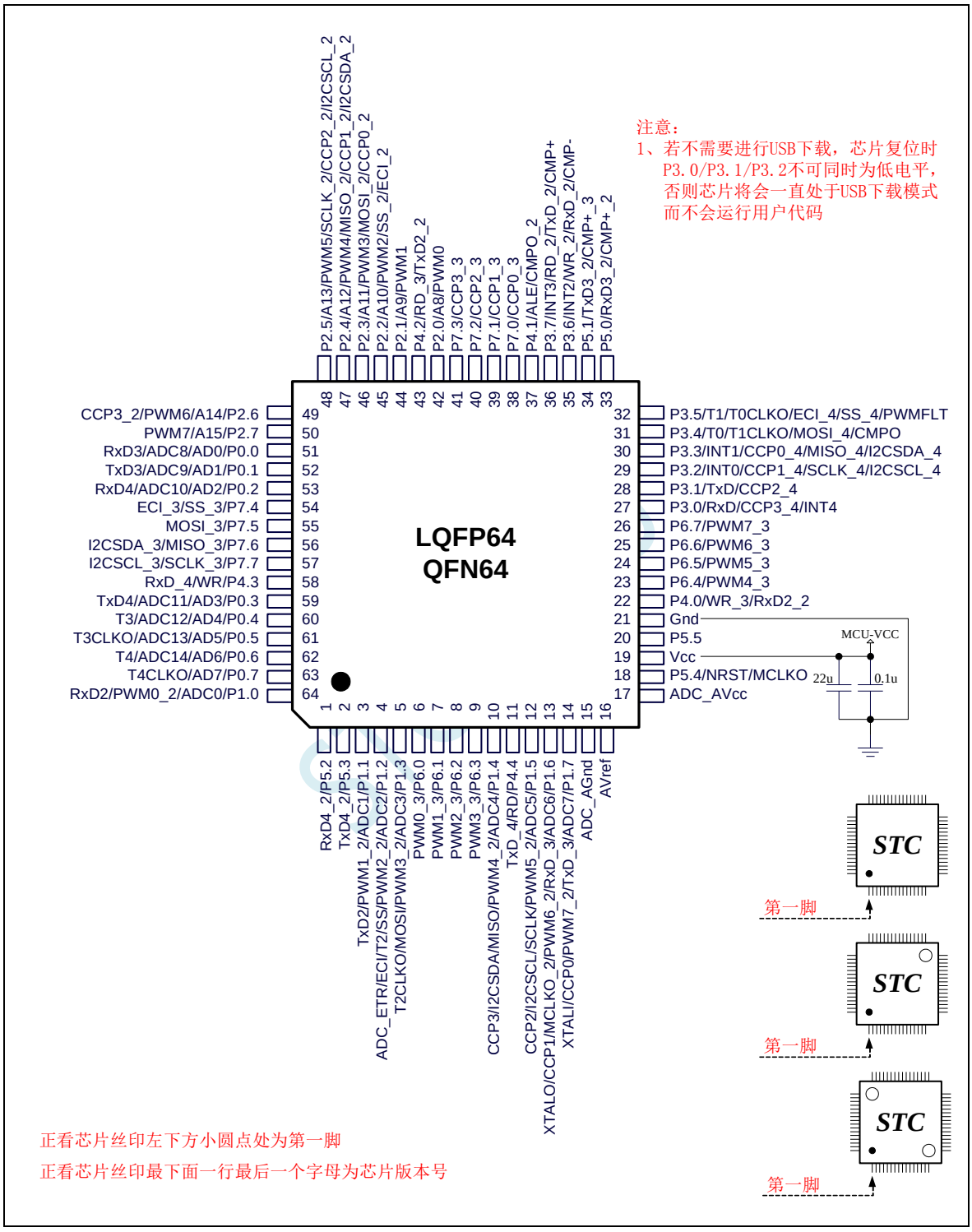
➢ **GPIO**

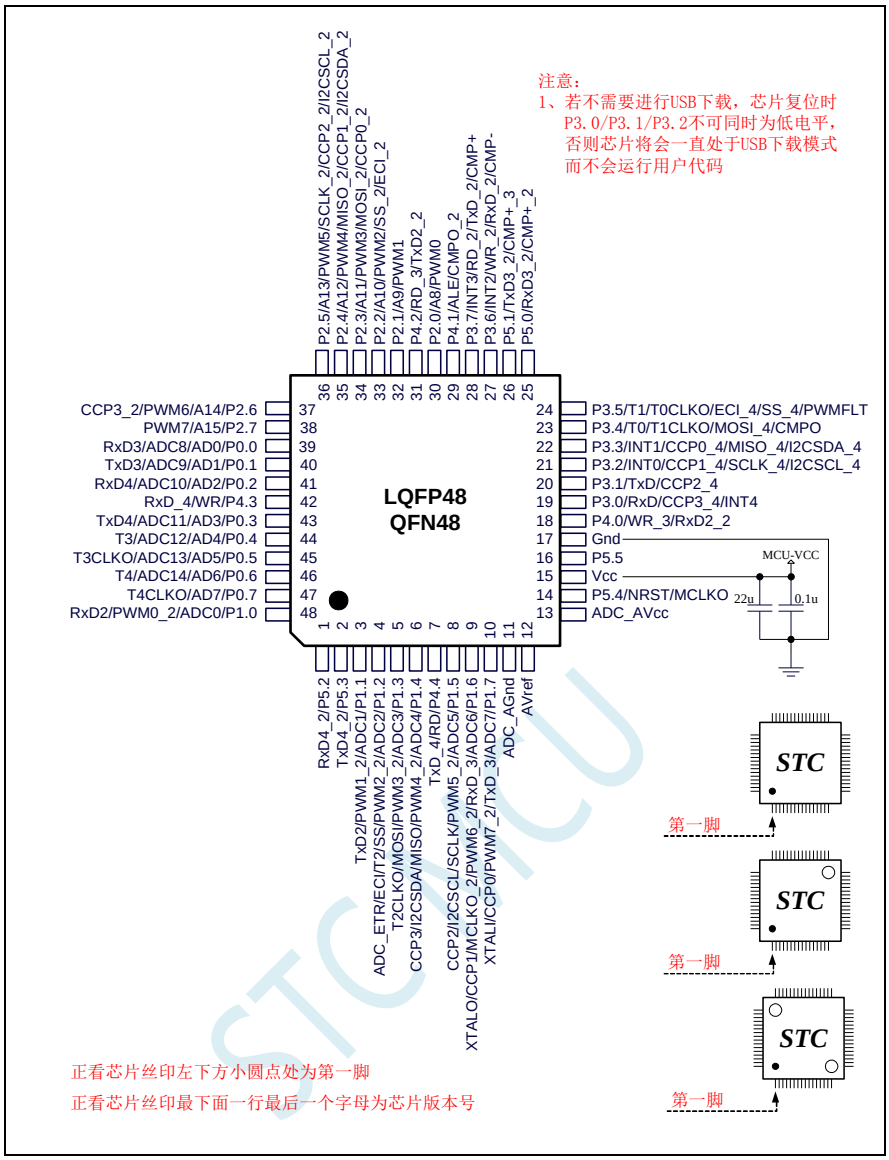
- ✓ 最多可达 59 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.4、P5.0~P5.5、P6.0~P6.7、P7.0~P7.7
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式
- ✓ **除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻**

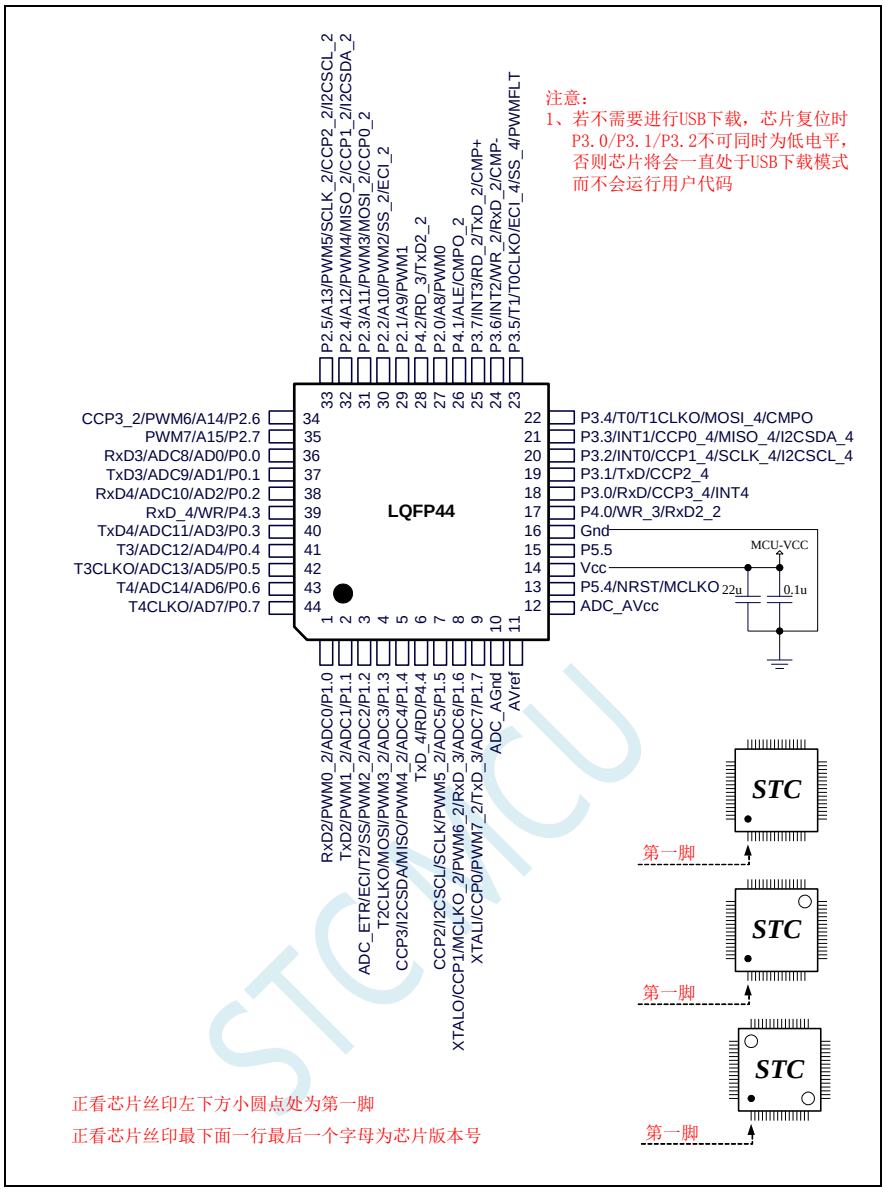
➢ **封装**

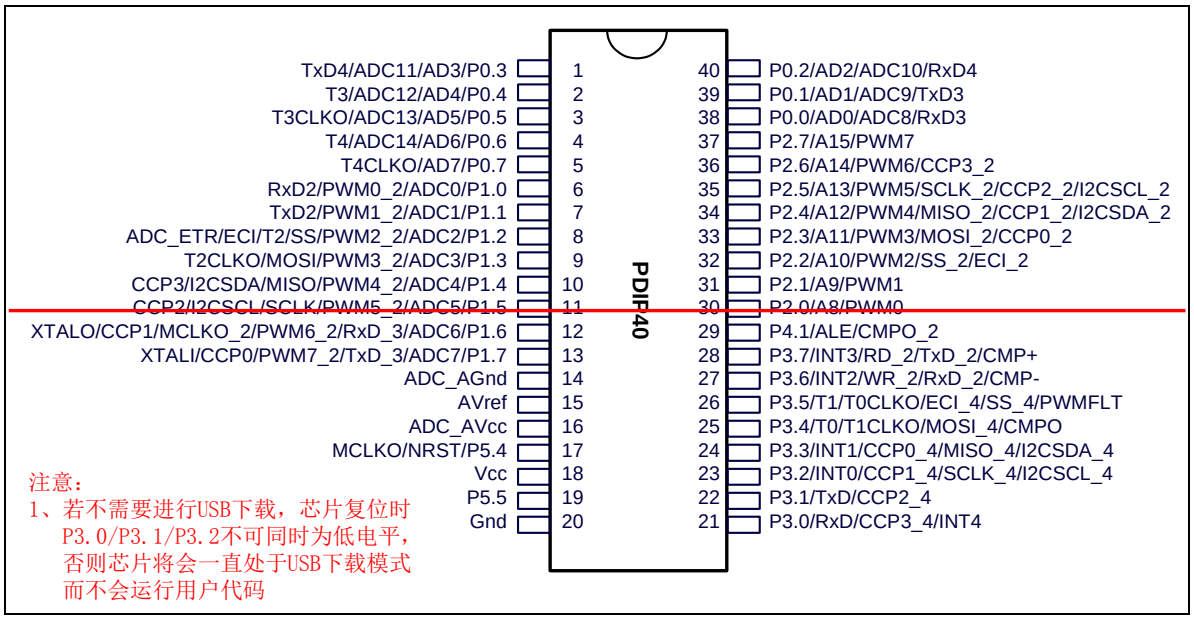
- ✓ LQFP64、LQFP48、LQFP44

2.1.2 管脚图，最小系统


注意：
1、若不需要进行USB下载，芯片复位时
P3.0/P3.1/P3.2不可同时为低电平，
否则芯片将会一直处于USB下载模式
而不会运行用户代码
LQFP64
QFN64
P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
P2.3/A11/PWM3/MOSI_2/CCP0_2
P2.2/A10/PWM2/SS_2/ECI_2
P2.1/A9/PWM1
P4.2/RD_3/TxD2_2
P2.0/A8/PWM0
P7.3/CCP3_3
P7.2/CCP2_3
P7.1/CCP1_3
P7.0/CCP0_3
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P5.1/TxD3_2/CMP+_3
P5.0/RxD3_2/CMP+_2
CCP3_2/PWM6/A14/P2.6
PWM7/A15/P2.7
RxD3/ADC8/AD0/P0.0
TxD3/ADC9/AD1/P0.1
RxD4/ADC10/AD2/P0.2
ECI_3/SS_3/P7.4
MOSI_3/P7.5
I2CSDA_3/MISO_3/P7.6
I2CSCL_3/SCLK_3/P7.7
RxD_4/WR/P4.3
TxD4/ADC11/AD3/P0.3
T3/ADC12/AD4/P0.4
T3CLKO/ADC13/AD5/P0.5
T4/ADC14/AD6/P0.6
T4CLKO/AD7/P0.7
RxD2/PWM0_2/ADC0/P1.0
P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
P3.4/T0/T1CLKO/MOSI_4/CMPO
P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
P3.1/TxD/CCP2_4
P3.0/RxD/CCP3_4/INT4
P6.7/PWM7_3
P6.6/PWM6_3
P6.5/PWM5_3
P6.4/PWM4_3
P4.0/WR_3/RxD2_2
Gnd
P5.5
Vcc
P5.4/NRST/MCLKO
ADC_AVcc
MCU-VCC
22u
0.1u
RxD4_2/P5.2
TxD4_2/P5.3
TxD2/PWM1_2/ADC1/P1.1
ADC_ETR/ECI/T2/SS/PWM2_2/ADC2/P1.2
T2CLKO/MOSI/PWM3_2/ADC3/P1.3
PWM0_3/P6.0
PWM1_3/P6.1
PWM2_3/P6.2
PWM3_3/P6.3
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
TxD_4/RD/P4.4
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
ADC_AGnd
AVref
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


注意：
1、若不需要进行USB下载，芯片复位时
P3.0/P3.1/P3.2不可同时为低电平，
否则芯片将会一直处于USB下载模式
而不会运行用户代码
LQFP48
QFN48
P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
P2.3/A11/PWM3/MOSI_2/CCP0_2
P2.2/A10/PWM2/SS_2/ECI_2
P2.1/A9/PWM1
P4.2/RD_3/TxD2_2
P2.0/A8/PWM0
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P5.1/TxD3_2/CMP+_3
P5.0/RxD3_2/CMP+_2
CCP3_2/PWM6/A14/P2.6
PWM7/A15/P2.7
RxD3/ADC8/AD0/P0.0
TxD3/ADC9/AD1/P0.1
RxD4/ADC10/AD2/P0.2
RxD_4/WR/P4.3
TxD4/ADC11/AD3/P0.3
T3/ADC12/AD4/P0.4
T3CLKO/ADC13/AD5/P0.5
T4/ADC14/AD6/P0.6
T4CLKO/AD7/P0.7
RxD2/PWM0_2/ADC0/P1.0
P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
P3.4/T0/T1CLKO/MOSI_4/CMPO
P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
P3.1/TxD/CCP2_4
P3.0/RxD/CCP3_4/INT4
P4.0/WR_3/RxD2_2
Gnd
P5.5
Vcc
P5.4/NRST/MCLKO
ADC_AVcc
MCU_VCC
22u
0.1u
RxD4_2/P5.2
TxD4_2/P5.3
TxD2/PWM1_2/ADC1/P1.1
ADC_ETR/ECI/T2/SS/PWM2_2/ADC2/P1.2
T2CLKO/MOSI/PWM3_2/ADC3/P1.3
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
TxD_4/RD/P4.4
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
XTALO/CCP0/PWM7_2/TxD_3/ADC7/P1.7
XTALI
ADC_AGnd
AVref
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


注意：
1、若不需要进行USB下载，芯片复位时
P3.0/P3.1/P3.2不可同时为低电平，
否则芯片将会一直处于USB下载模式
而不会运行用户代码
LQFP44
P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
P2.3/A11/PWM3/MOSI_2/CCP0_2
P2.2/A10/PWM2/SS_2/ECI_2
P2.1/A9/PWM1
P4.2/RD_3/TxD2_2
P2.0/A8/PWM0
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
CCP3_2/PWM6/A14/P2.6
PWM7/A15/P2.7
RxD3/ADC8/AD0/P0.0
TxD3/ADC9/AD1/P0.1
RxD4/ADC10/AD2/P0.2
RxD_4/WR/P4.3
TxD4/ADC11/AD3/P0.3
T3/ADC12/AD4/P0.4
T3CLKO/ADC13/AD5/P0.5
T4/ADC14/AD6/P0.6
T4CLKO/AD7/P0.7
P3.4/T0/T1CLKO/MOSI_4/CMPO
P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
P3.1/TxD/CCP2_4
P3.0/RxD/CCP3_4/INT4
P4.0/WR_3/RxD2_2
Gnd
P5.5
Vcc
P5.4/NRST/MCLKO
ADC_AVcc
MCU-VCC
22u
0.1u
RxD2/PWM0_2/ADC0/P1.0
TxD2/PWM1_2/ADC1/P1.1
ADC_ETR/ECI/T2/SS/PWM2_2/ADC2/P1.2
T2CLKO/MOSI/PWM3_2/ADC3/P1.3
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
TxD_4/RD/P4.4
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
ADC_AGnd
AVref
STC
第一脚
STC
第一脚
STC
第一脚
正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号


TxD4/ADC11/AD3/P0.3 1
T3/ADC12/AD4/P0.4 2
T3CLKO/ADC13/AD5/P0.5 3
T4/ADC14/AD6/P0.6 4
T4CLKO/AD7/P0.7 5
RxD2/PWM0_2/ADC0/P1.0 6
TxD2/PWM1_2/ADC1/P1.1 7
ADC_ETR/ECI/T2/SS/PWM2_2/ADC2/P1.2 8
T2CLKO/MOSI/PWM3_2/ADC3/P1.3 9
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4 10
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5 11
XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6 12
XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7 13
ADC_AGnd 14
AVref 15
ADC_AVcc 16
MCLKO/NRST/P5.4 17
Vcc 18
P5.5 19
Gnd 20
PDIP40
40 P0.2/AD2/ADC10/RxD4
39 P0.1/AD1/ADC9/TxD3
38 P0.0/AD0/ADC8/RxD3
37 P2.7/A15/PWM7
36 P2.6/A14/PWM6/CCP3_2
35 P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
34 P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
33 P2.3/A11/PWM3/MOSI_2/CCP0_2
32 P2.2/A10/PWM2/SS_2/ECI_2
31 P2.1/A9/PWM1
30 P2.0/A8/PWM0
29 P4.1/ALE/CMPO_2
28 P3.7/INT3/RD_2/TxD_2/CMP+
27 P3.6/INT2/WR_2/RxD_2/CMP-
26 P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
25 P3.4/T0/T1CLKO/MOSI_4/CMPO
24 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
23 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
22 P3.1/TxD/CCP2_4
21 P3.0/RxD/CCP3_4/INT4
注意：
1、若不需要进行USB下载，芯片复位时
P3.0/P3.1/P3.2不可同时为低电平，
否则芯片将会一直处于USB下载模式
而不会运行用户代码

2.1.3 管脚说明

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
1	1			P5.2	I/O	标准 IO 口
				RxD4_2	I	串口 4 的接收脚
2	2			P5.3	I/O	标准 IO 口
				TxD4_2	O	串口 4 的发送脚
3	3	2	7	P1.1	I/O	标准 IO 口
				ADC1	I	ADC 模拟输入通道 1
				PWM1_2	O	增强 PWM 通道 1 输出脚
				TxD2	O	串口 2 的发送脚
4	4	3	8	P1.2	I/O	标准 IO 口
				ADC2	I	ADC 模拟输入通道 2
				PWM2_2	O	增强 PWM 通道 2 输出脚
				SS	I/O	SPI 从机选择
				T2	I	定时器 2 外部时钟输入
				ECI	I	PCA 的外部脉冲输入
				ADC_ETR	I	ADC 外部触发输入
5	5	4	9	P1.3	I/O	标准 IO 口
				ADC3	I	ADC 模拟输入通道 3
				PWM3_2	O	增强 PWM 通道 3 输出脚
				MOSI	I/O	SPI 主机输出从机输入
				T2CLKO	O	定时器 2 时钟分频输出
6				P6.0	I/O	标准 IO 口
				PWM0_3	O	增强 PWM 通道 0 输出脚
7				P6.1	I/O	标准 IO 口
				PWM1_3	O	增强 PWM 通道 1 输出脚
8				P6.2	I/O	标准 IO 口
				PWM2_3	O	增强 PWM 通道 2 输出脚
9				P6.3	I/O	标准 IO 口
				PWM3_3	O	增强 PWM 通道 3 输出脚

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
10	6	5	10	P1.4	I/O	标准 IO 口
				ADC4	I	ADC 模拟输入通道 4
				PWM4_2	O	增强 PWM 通道 4 输出脚
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
				CCP3	I/O	PCA 的捕获输入和脉冲输出
11	7	6		P4.4	I/O	标准 IO 口
				RD	O	外部总线的读信号线
				TxD_4	O	串口 1 的发送脚
12	8	7	11	P1.5	I/O	标准 IO 口
				ADC5	I	ADC 模拟输入通道 5
				PWM5_2	O	增强 PWM 通道 5 输出脚
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
				CCP2	I/O	PCA 的捕获输入和脉冲输出
13	9	8	12	P1.6	I/O	标准 IO 口
				ADC6	I	ADC 模拟输入通道 6
				RxD_3	I	串口 1 的接收脚
				PWM6_2	O	增强 PWM 通道 6 输出脚
				MCLKO_2	O	主时钟分频输出
				CCP1	I/O	PCA 的捕获输入和脉冲输出
				XTALO	O	外部晶振的输出脚
14	10	9	13	P1.7	I/O	标准 IO 口
				ADC7	I	ADC 模拟输入通道 7
				TxD_3	O	串口 1 的发送脚
				PWM7_2	O	增强 PWM 通道 7 输出脚
				CCP0	I/O	PCA 的捕获输入和脉冲输出
				XTALI	I	外部晶振/外部时钟的输入脚
15	11	10	14	ADC_AGnd	GND	ADC 地线
16	12	11	15	AVref	I	ADC 的参考电压脚
17	13	12	16	ADC_AVcc	VCC	ADC 电源脚
18	14	13	17	P5.4	I/O	标准 IO 口
				NRST	I	复位引脚（低电平复位）
				MCLKO	O	主时钟分频输出
19	15	14	18	Vcc	VCC	电源脚
20	16	15	19	P5.5	I/O	标准 IO 口
21	17	16	20	Gnd	GND	地线

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
22	18	17		P4.0	I/O	标准 IO 口
				WR_3	O	外部总线的写信号线
				RxD2_2	I	串口 2 的接收脚
23				P6.4	I/O	标准 IO 口
				PWM4_3	O	增强 PWM 通道 4 输出脚
24				P6.5	I/O	标准 IO 口
				PWM5_3	O	增强 PWM 通道 5 输出脚
25				P6.6	I/O	标准 IO 口
				PWM6_3	O	增强 PWM 通道 6 输出脚
26				P6.7	I/O	标准 IO 口
				PWM7_3	O	增强 PWM 通道 7 输出脚
27	19	18	21	P3.0	I/O	标准 IO 口
				RxD	I	串口 1 的接收脚
				CCP3_4	I/O	PCA 的捕获输入和脉冲输出
				INT4	I	外部中断 4
28	20	19	22	P3.1	I/O	标准 IO 口
				TxD	O	串口 1 的发送脚
				CCP2_4	I/O	PCA 的捕获输入和脉冲输出
29	21	20	23	P3.2	I/O	标准 IO 口
				INT0	I	外部中断 0
				CCP1_4	I/O	PCA 的捕获输入和脉冲输出
				SCLK_4	I/O	SPI 的时钟脚
				SCL_4	I/O	I2C 的时钟线
30	22	21	24	P3.3	I/O	标准 IO 口
				INT1	I	外部中断 1
				CCP0_4	I/O	PCA 的捕获输入和脉冲输出
				MISO_4	I/O	SPI 主机输入从机输出
				SDA_4	I/O	I2C 接口的数据线
31	23	22	25	P3.4	I/O	标准 IO 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				MOSI_4	I/O	SPI 主机输出从机输入
				CMPO	O	比较器输出

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
32	24	23	26	P3.5	I/O	标准 IO 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				ECI_4	I	PCA 的外部脉冲输入
				SS_4	I	SPI 的从机选择脚（主机为输出）
				PWMFLT	I	增强 PWM 的外部异常检测脚
33	25			P5.0	I/O	标准 IO 口
				RxD3_2	I	串口 3 的接收脚
				CMP+_2	I	比较器正极输入
34	26			P5.1	I/O	标准 IO 口
				TxD3_2	O	串口 3 的发送脚
				CMP+_3	I	比较器正极输入
35	27	24	27	P3.6	I/O	标准 IO 口
				INT2	I	外部中断 2
				WR_2	O	外部总线的写信号线
				RxD_2	I	串口 1 的接收脚
				CMP-	I	比较器负极输入
36	28	25	28	P3.7	I/O	标准 IO 口
				INT3	I	外部中断 3
				RD_2	O	外部总线的读信号线
				TxD_2	O	串口 1 的发送脚
				CMP+	I	比较器正极输入
37	29	26	29	P4.1	I/O	标准 IO 口
				ALE	O	地址锁存信号
				CMPO_2	O	比较器输出
38				P7.0	I/O	标准 IO 口
				CCP0_3	I/O	PCA 的捕获输入和脉冲输出
39				P7.1	I/O	标准 IO 口
				CCP1_3	I/O	PCA 的捕获输入和脉冲输出
40				P7.2	I/O	标准 IO 口
				CCP2_3	I/O	PCA 的捕获输入和脉冲输出
41				P7.3	I/O	标准 IO 口
				CCP3_3	I/O	PCA 的捕获输入和脉冲输出
42	30	27	30	P2.0	I/O	标准 IO 口
				A8	I	地址总线
				PWM0	O	增强 PWM 通道 0 输出脚
43	31	28		P4.2	I/O	标准 IO 口
				RD_3	O	外部总线的读信号线
				TxD2_2	O	串口 2 的发送脚

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
44	32	29	31	P2.1	I/O	标准 IO 口
				A9	I	地址总线
				PWM1	O	增强 PWM 通道 1 输出脚
45	33	30	32	P2.2	I/O	标准 IO 口
				A10	I	地址总线
				PWM2	O	增强 PWM 通道 2 输出脚
				SS_2	I	SPI 的从机选择脚（主机为输出）
				ECI_2	I	PCA 的外部脉冲输入
46	34	31	33	P2.3	I/O	标准 IO 口
				A11	I	地址总线
				PWM3	O	增强 PWM 通道 3 输出脚
				MOSI_2	I/O	SPI 主机输出从机输入
				CCP0_2	I/O	PCA 的捕获输入和脉冲输出
47	35	32	34	P2.4	I/O	标准 IO 口
				A12	I	地址总线
				PWM4	O	增强 PWM 通道 4 输出脚
				MISO_2	I/O	SPI 主机输入从机输出
				SDA_2	I/O	I2C 接口的数据线
				CCP1_2	I/O	PCA 的捕获输入和脉冲输出
48	36	33	35	P2.5	I/O	标准 IO 口
				A13	I	地址总线
				PWM5	O	增强 PWM 通道 5 输出脚
				SCLK_2	I/O	SPI 的时钟脚
				SCL_2	I/O	I2C 的时钟线
				CCP2_2	I/O	PCA 的捕获输入和脉冲输出
49	37	34	36	P2.6	I/O	标准 IO 口
				A14	I	地址总线
				PWM6	O	增强 PWM 通道 6 输出脚
				CCP3_2	I/O	PCA 的捕获输入和脉冲输出
50	38	35	37	P2.7	I/O	标准 IO 口
				A15	I	地址总线
				PWM7	O	增强 PWM 通道 7 输出脚
51	39	36	38	P0.0	I/O	标准 IO 口
				AD0	I	地址总线
				ADC8	I	ADC 模拟输入通道 8
				RxD3	I	串口 3 的接收脚

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
52	40	37	39	P0.1	I/O	标准 IO 口
				AD1	I	地址总线
				ADC9	I	ADC 模拟输入通道 9
				TxD3	O	串口 3 的发送脚
53	41	38	40	P0.2	I/O	标准 IO 口
				AD2	I	地址总线
				ADC10	I	ADC 模拟输入通道 10
				RxD4	I	串口 4 的接收脚
54				P7.4	I/O	标准 IO 口
				SS_3	I	SPI 的从机选择脚（主机为输出）
				ECI_3	I	PCA 的外部脉冲输入
55				P7.5	I/O	标准 IO 口
				MOSI_3	I/O	SPI 主机输出从机输入
56				P7.6	I/O	标准 IO 口
				MISO_3	I/O	SPI 主机输入从机输出
				SDA_3	I/O	I2C 接口的数据线
57				P7.7	I/O	标准 IO 口
				SCLK_3	I/O	SPI 的时钟脚
				SCL_3	I/O	I2C 的时钟线
58	42	39		P4.3	I/O	标准 IO 口
				WR	O	外部总线的写信号线
				RxD_4	I	串口 1 的接收脚
59	43	40	1	P0.3	I/O	标准 IO 口
				AD3	I	地址总线
				ADC11	I	ADC 模拟输入通道 11
				TxD4	O	串口 4 的发送脚
60	44	41	2	P0.4	I/O	标准 IO 口
				AD4	I	地址总线
				ADC12	I	ADC 模拟输入通道 12
				T3	I	定时器 3 外部时钟输入

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
61	45	42	3	P0.5	I/O	标准 IO 口
				AD5	I	地址总线
				ADC13	I	ADC 模拟输入通道 13
				T3CLKO	O	定时器 3 时钟分频输出
62	46	43	4	P0.6	I/O	标准 IO 口
				AD6	I	地址总线
				ADC14	I	ADC 模拟输入通道 14
				T4	I	定时器 4 外部时钟输入
63	47	44	5	P0.7	I/O	标准 IO 口
				AD7	I	地址总线
				T4CLKO	O	定时器 4 时钟分频输出
64	48	1	6	P1.0	I/O	标准 IO 口
				ADC0	I	ADC 模拟输入通道 0
				PWM0_2	O	增强 PWM 通道 0 输出脚
				RxD2	I	串口 2 的接收脚

3 功能脚切换

STC8A8K64D4 系列单片机的特殊外设串口、SPI、PCA、I²C 以及总线控制脚可以在多个 I/O 直接进行切换，以实现一个外设当作多个设备进行分时复用。

3.1 功能脚切换相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]						SPEED[1:0]		00xx,xx00
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,000x
P_SW2	外设端口切换寄存器 2	BAH	EAXFR		I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM0CR	PWM0 控制寄存器	FF14H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000
PWM1CR	PWM1 控制寄存器	FF1CH	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000
PWM3CR	PWM3 控制寄存器	FF2CH	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000
PWM4CR	PWM4 控制寄存器	FF34H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000
PWM5CR	PWM5 控制寄存器	FF3CH	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000
PWM6CR	PWM6 控制寄存器	FF44H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000
PWM7CR	PWM7 控制寄存器	FF4CH	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
LCMIFCFG	LCM 接口配置寄存器	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFPS[1:0]		D16_D8	M68_80	0x00,0000
LCMIFCFG2	LCM 接口配置寄存器 2	FE51H		LCMIFEXPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000

3.1.1 总线速度控制寄存器（BUS_SPEED）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]						SPEED[1:0]	

RW_S[1:0]：外部总线 RD/WR 控制线选择位

RW_S[1:0]	RD	WR
00	P4.4	P4.3
01	P3.7	P3.6
10	P4.2	P4.0
11	保留	

3.1.2 外设端口切换控制寄存器 1（P_SW1），串口 1、CCP、SPI 切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-

S1_S[1:0]：串口 1 功能脚选择位

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.6	P3.7
10	P1.6	P1.7
11	P4.3	P4.4

CCP_S[1:0]：PCA 功能脚选择位

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2	CCP3
00	P1.2	P1.7	P1.6	P1.5	P1.4
01	P2.2	P2.3	P2.4	P2.5	P2.6
10	P7.4	P7.0	P7.1	P7.2	P7.3
11	P3.5	P3.3	P3.2	P3.1	P3.0

SPI_S[1:0]：SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P7.4	P7.5	P7.6	P7.7
11	P3.5	P3.4	P3.3	P3.2

3.1.3 外设端口切换控制寄存器 2（P_SW2），串口 2/3/4、I2C、比较器输出切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

EAXFR：扩展 RAM 区特殊功能寄存器（XFR）访问控制寄存器

0：禁止访问 XFR

1：使能访问 XFR。

当需要访问 XFR 时，必须先将 EAXFR 置 1，才能对 XFR 进行正常的读写

I2C_S[1:0]：I^2C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P2.5	P2.4
10	P7.7	P7.6
11	P3.2	P3.3

CMPO_S：比较器输出脚选择位

CMPO_S	CMPO
0	P3.4
1	P4.1

S4_S：串口 4 功能脚选择位

S4_S	RxD4	TxD4
0	P0.2	P0.3
1	P5.2	P5.3

S3_S：串口 3 功能脚选择位

S3_S	RxD3	TxD3
0	P0.0	P0.1
1	P5.0	P5.1

S2_S：串口 2 功能脚选择位

S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P4.0	P4.2

3.1.4 时钟选择寄存器（MCLKOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	FE05H	MCLKO_S	MCLKODIV[6:0]						

MCLKO_S：主时钟输出脚选择位

MCLKO_S	MCLKO
0	P5.4
1	P1.6

3.1.5 增强型 PWM 控制寄存器（PWMnCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF14H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF1CH	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF2CH	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF34H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI

PWM5CR	FF3CH	ENC5O	C5INI	-	C5_S[1:0]	EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF44H	ENC6O	C6INI	-	C6_S[1:0]	EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF4CH	ENC7O	C7INI	-	C7_S[1:0]	EC7I	EC7T2SI	EC7T1SI

C0_S[1:0]：增强型 PWM 通道 0 输出脚选择位

C0_S[1:0]	PWM0
00	P2.0
01	P1.0
10	P6.0
11	保留

C1_S[1:0]：增强型 PWM 通道 1 输出脚选择位

C1_S[1:0]	PWM1
00	P2.1
01	P1.1
10	P6.1
11	保留

C2_S[1:0]：增强型 PWM 通道 2 输出脚选择位

C2_S[1:0]	PWM2
00	P2.2
01	P1.2
10	P6.2
11	保留

C3_S[1:0]：增强型 PWM 通道 3 输出脚选择位

C3_S[1:0]	PWM3
00	P2.3
01	P1.3
10	P6.3
11	保留

C4_S[1:0]：增强型 PWM 通道 4 输出脚选择位

C4_S[1:0]	PWM4
00	P2.4
01	P1.4
10	P6.4
11	保留

C5_S[1:0]：增强型 PWM 通道 5 输出脚选择位

C5_S[1:0]	PWM5
00	P2.5
01	P1.5
10	P6.5
11	保留

C6_S[1:0]：增强型 PWM 通道 6 输出脚选择位

C6_S[1:0]	PWM6
00	P2.6
01	P1.6
10	P6.6

11	保留

C7_S[1:0]：增强型 PWM 通道 7 输出脚选择位

C7_S[1:0]	PWM7
00	P2.7
01	P1.7
10	P6.7
11	保留

3.1.6 LCM 接口配置寄存器（LCMIFCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80
LCMIFCFG2	FE51H		LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

LCMIFCPS[1:0]：LCM 接口控制脚选择位

LCMIFCPS [1:0]	RS	I8080 的读信号 RD M6800 的使能信号 E	I8080 的写信号 WR M6800 的读写信号 RW
00	P4.1	P4.4	P4.3
01	P4.1	P3.7	P3.6
10	P4.1	P4.2	P4.0
11	P4.0	P3.7	P3.6

LCMIFDPS[1:0]：LCM 接口数据脚选择位

LCMIFDPS [1:0]	D16_D8	数据高字节 DB[15:8]	数据低字节 DB[7:0]
00	0	N/A	P2[7:0]
01	0	N/A	P6[7:0]
10	0	N/A	P2[7:0]
11	0	N/A	P6[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	P7[7:0]
11	1	P6[7:0]	P7[7:0]

3.2 范例程序

3.2.1 串口 1 切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr         P_SW1           =   0xa2;

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW1 = 0x00;                   //RXD/P3.0, TXD/P3.1
//  P_SW1 = 0x40;                   //RXD_2/P3.6, TXD_2/P3.7
//  P_SW1 = 0x80;                   //RXD_3/P1.6, TXD_3/P1.7
//  P_SW1 = 0xc0;                   //RXD_4/P4.3, TXD_4/P4.4

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW1       DATA    0A2H

P0M1        DATA    093H
P0M0        DATA    094H
```

```
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW1,#00H      ;RXD/P3.0, TXD/P3.1
;           MOV     P_SW1,#40H      ;RXD_2/P3.6, TXD_2/P3.7
;           MOV     P_SW1,#80H      ;RXD_3/P1.6, TXD_3/P1.7
;           MOV     P_SW1,#0C0H     ;RXD_4/P4.3, TXD_4/P4.4

            SJMP    $

            END
```

3.2.2 串口 2 切换

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW2   =   0xba;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
```

```
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;                   //RXD2/P1.0, TXD2/P1.1
//  P_SW2 = 0x01;                   //RXD2_2/P4.0, TXD2_2/P4.2

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
```

```
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#00H              ;RXD2/P1.0, TXD2/P1.1
;       MOV     P_SW2,#01H              ;RXD2_2/P4.0, TXD2_2/P4.2

        SJMP    $

        END
```

3.2.3 串口 3 切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr     P_SW2       =   0xba;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;                       //RXD3/P0.0, TXD3/P0.1
//  P_SW2 = 0x02;                       //RXD3_2/P5.0, TXD3_2/P5.1

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#00H          ;RXD3/P0.0, TXD3/P0.1
;           MOV     P_SW2,#02H          ;RXD3_2/P5.0, TXD3_2/P5.1

            SJMP    $

            END
```

3.2.4 串口 4 切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr     P_SW2   =   0xba;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
```

```
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;                   //RXD4/P0.2, TXD4/P0.3
//  P_SW2 = 0x04;                   //RXD4_2/P5.2, TXD4_2/P5.3

    while (1);
}
```

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
```

```
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#00H          ;RXD4/P0.2, TXD4/P0.3
;       MOV     P_SW2,#04H          ;RXD4_2/P5.2, TXD4_2/P5.3

        SJMP    $

        END
```

3.2.5 SPI 切换

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW1   =   0xa2;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW1 = 0x00;                   //SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
//  P_SW1 = 0x04;                   //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
//  P_SW1 = 0x08;                   //SS_3/P7.4 MOSI_3/P7.5, MISO_3/P7.6, SCLK_3/P7.7
```

```
//      P_SW1 = 0x0c;                                   //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

        while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW1       DATA    0A2H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW1,#00H      ;SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
;           MOV     P_SW1,#04H      ;SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
;           MOV     P_SW1,#08H      ;SS_3/P7.4 MOSI_3/P7.5, MISO_3/P7.6, SCLK_3/P7.7
;           MOV     P_SW1,#0CH      ;SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

            SJMP    $

            END
```

3.2.6 PWM 切换

汇编代码

```
P_SW2       DATA    0BAH
PWM0CR      EQU     0FF14H
```

```
PWM1CR      EQU     0FF1CH
PWM2CR      EQU     0FF24H
PWM3CR      EQU     0FF2CH
PWM4CR      EQU     0FF34H
PWM5CR      EQU     0FF3CH
PWM6CR      EQU     0FF44H
PWM7CR      EQU     0FF4CH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #3FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H
            MOV     A,#00H          ;PWM0/P2.0
;           MOV     A,#08H          ;PWM0_2/P1.0
;           MOV     A,#10H          ;PWM0_3/P6.0
            MOV     DPTR,#PWM0CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM1/P2.1
;           MOV     A,#08H          ;PWM1_2/P1.1
;           MOV     A,#10H          ;PWM1_3/P6.1
            MOV     DPTR,#PWM1CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM2/P2.2
;           MOV     A,#08H          ;PWM2_2/P1.2
;           MOV     A,#10H          ;PWM2_3/P6.2
            MOV     DPTR,#PWM2CR
            MOVX    @DPTR,A
            MOV     A,#00H          ;PWM3/P2.3
;           MOV     A,#08H          ;PWM3_2/P1.3
;           MOV     A,#10H          ;PWM3_3/P6.3
```

```
        MOV     DPTR,#PWM3CR
        MOVX    @DPTR,A
        MOV     A,#00H                  ;PWM4/P2.4
;       MOV     A,#08H                  ;PWM4_2/P1.4
;       MOV     A,#10H                  ;PWM4_3/P6.4
        MOV     DPTR,#PWM4CR
        MOVX    @DPTR,A
        MOV     A,#00H                  ;PWM5/P2.5
;       MOV     A,#08H                  ;PWM5_2/P1.5
;       MOV     A,#10H                  ;PWM5_3/P6.5
        MOV     DPTR,#PWM5CR
        MOVX    @DPTR,A
        MOV     A,#00H                  ;PWM6/P2.6
;       MOV     A,#08H                  ;PWM6_2/P1.6
;       MOV     A,#10H                  ;PWM6_3/P6.6
        MOV     DPTR,#PWM6CR
        MOVX    @DPTR,A
        MOV     A,#00H                  ;PWM7/P2.7
;       MOV     A,#08H                  ;PWM7_2/P1.7
;       MOV     A,#10H                  ;PWM7_3/P6.7
        MOV     DPTR,#PWM7CR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

#define PWM0CR  (*(unsigned char volatile xdata *)0xff14)
#define PWM1CR  (*(unsigned char volatile xdata *)0xff1c)
#define PWM2CR  (*(unsigned char volatile xdata *)0xff24)
#define PWM3CR  (*(unsigned char volatile xdata *)0xff2c)
#define PWM4CR  (*(unsigned char volatile xdata *)0xff34)
#define PWM5CR  (*(unsigned char volatile xdata *)0xff3c)
#define PWM6CR  (*(unsigned char volatile xdata *)0xff44)
#define PWM7CR  (*(unsigned char volatile xdata *)0xff4c)

sfr     P_SW2       =   0xba;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
        P0M0 = 0x00;
```

```
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    PWM0CR = 0x00;          //PWM0/P2.0
//  PWM0CR = 0x08;          //PWM0_2/P1.0
//  PWM0CR = 0x10;          //PWM0_3/P6.0
    PWM1CR = 0x00;          //PWM1/P2.1
//  PWM1CR = 0x08;          //PWM1_2/P1.1
//  PWM1CR = 0x10;          //PWM1_3/P6.1
    PWM2CR = 0x00;          //PWM2/P2.2
//  PWM2CR = 0x08;          //PWM2_2/P1.2
//  PWM2CR = 0x10;          //PWM2_3/P6.2
    PWM3CR = 0x00;          //PWM3/P2.3
//  PWM3CR = 0x08;          //PWM3_2/P1.3
//  PWM3CR = 0x10;          //PWM3_3/P6.3
    PWM4CR = 0x00;          //PWM4/P2.4
//  PWM4CR = 0x08;          //PWM4_2/P1.4
//  PWM4CR = 0x10;          //PWM4_3/P6.4
    PWM5CR = 0x00;          //PWM5/P2.5
//  PWM5CR = 0x08;          //PWM5_2/P1.5
//  PWM5CR = 0x10;          //PWM5_3/P6.5
    PWM6CR = 0x00;          //PWM6/P2.6
//  PWM6CR = 0x08;          //PWM6_2/P1.6
//  PWM6CR = 0x10;          //PWM6_3/P6.6
    PWM7CR = 0x00;          //PWM7/P2.7
//  PWM7CR = 0x08;          //PWM7_2/P1.7
//  PWM7CR = 0x10;          //PWM7_3/P6.7
    P_SW2 = 0x00;

    while (1);
}
```

3.2.7 PCA/CCP/PWM 切换

汇编代码

```
P_SW1   DATA    0A2H

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
```

```
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW1,#00H          ;ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5,CCP3/P1.4
;       MOV     P_SW1,#10H          ;ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5,CCP3_2/P2.6
;       MOV     P_SW1,#20H          ;ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2,CCP3_3/P7.3
;       MOV     P_SW1,#30H          ;ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1,CCP3_4/P3.0

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr     P_SW1       =   0xa2;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
```

```
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        P_SW1 = 0x00;           //ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5,CCP3/P1.4
//      P_SW1 = 0x10;           //ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5,CCP3_2/P2.6
//      P_SW1 = 0x20;           //ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2,CCP3_3/P7.3
//      P_SW1 = 0x30;           //ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1,CCP3_4/P3.0

        while (1);
}
```

3.2.8 I2C 切换

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW2       =   0xba;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        P_SW2 = 0x00;                   //SCL/P1.5, SDA/P1.4
//      P_SW2 = 0x10;                   //SCL_2/P2.5, SDA_2/P2.4
//      P_SW2 = 0x20;                   //SCL_3/P7.7, SDA_3/P7.6
//      P_SW2 = 0x30;                   //SCL_4/P3.2, SDA_4/P3.3

        while (1);
```

```
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#00H          ;SCL/P1.5, SDA/P1.4
;           MOV     P_SW2,#10H          ;SCL_2/P2.5, SDA_2/P2.4
;           MOV     P_SW2,#20H          ;SCL_3/P7.7, SDA_3/P7.6
;           MOV     P_SW2,#30H          ;SCL_4/P3.2, SDA_4/P3.3

            SJMP    $

            END
```

3.2.9 比较器输出切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
```

```
sfr     P_SW2   =   0xba;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;                   //CMPO/P3.4
//  P_SW2 = 0x08;                   //CMPO_2/P4.1

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
```

```
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#00H          ;CMPO/P3.4
;       MOV     P_SW2,#08H          ;CMPO_2/P4.1

        SJMP    $

        END
```

3.2.10 主时钟输出切换

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

#define   MCLKOCR   (*(unsigned char volatile xdata *)0xfe00)

sfr     P_SW2     =   0xba;

sfr     P0M1      =   0x93;
sfr     P0M0      =   0x94;
sfr     P1M1      =   0x91;
sfr     P1M0      =   0x92;
sfr     P2M1      =   0x95;
sfr     P2M0      =   0x96;
sfr     P3M1      =   0xb1;
sfr     P3M0      =   0xb2;
sfr     P4M1      =   0xb3;
sfr     P4M0      =   0xb4;
sfr     P5M1      =   0xc9;
sfr     P5M0      =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    MCLKOCR = 0x04;                 //HIRC/4 output via MCLKO/P5.4
//  MCLKOCR = 0x84;                 //HIRC/4 output via MCLKO_2/P1.6
    P_SW2 = 0x00;

    while (1);
}
```

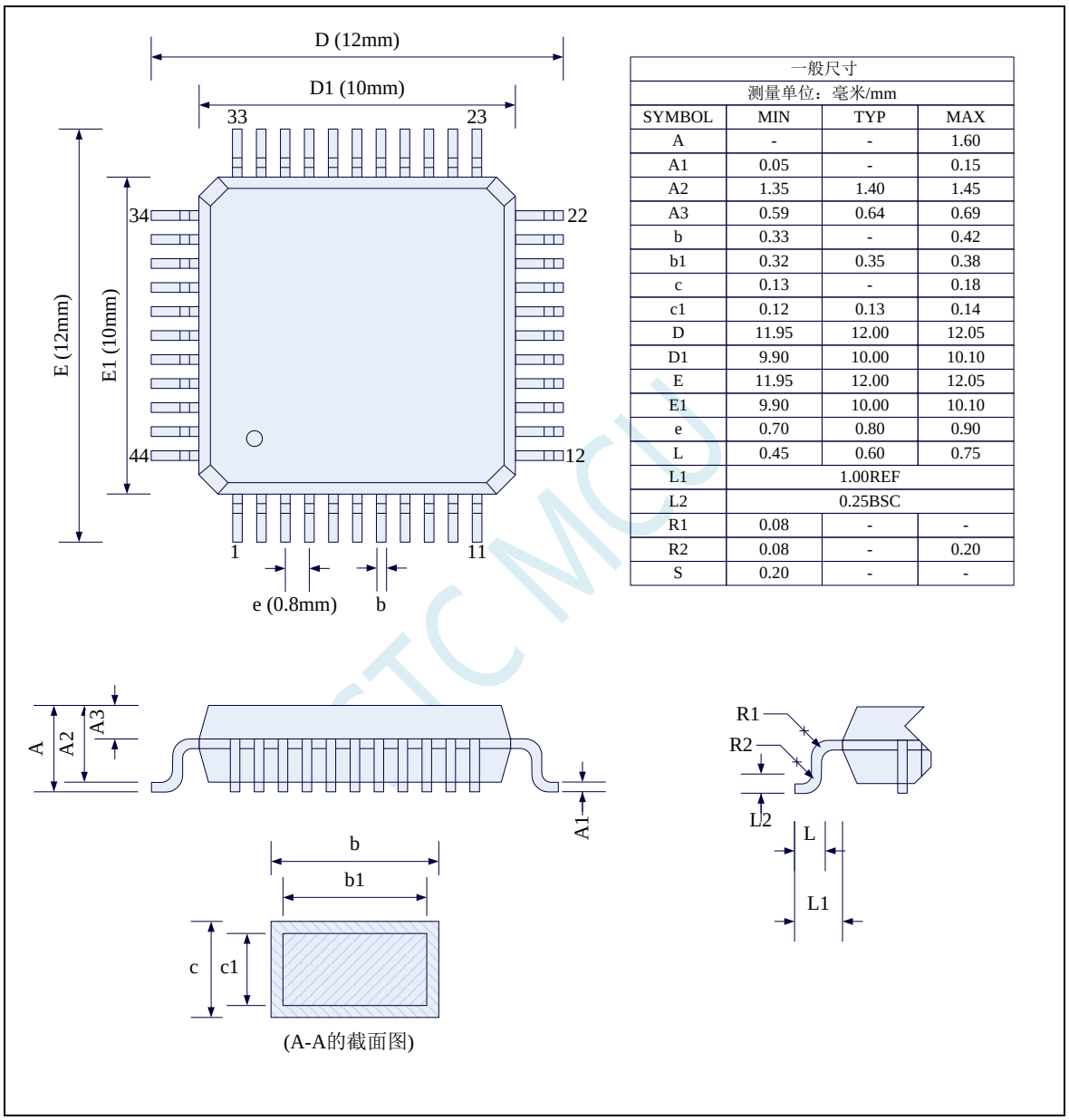
汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

MCLKOCR     EQU     0FE05H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H
            MOV     A,#04H          ;HIRC/4 output via MCLKO/P5.4
;           MOV     A,#84H          ;HIRC/4 output via MCLKO_2/P1.6
            MOV     DPTR,#MCLKOCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
```

```
        SJMP    $

        END
```

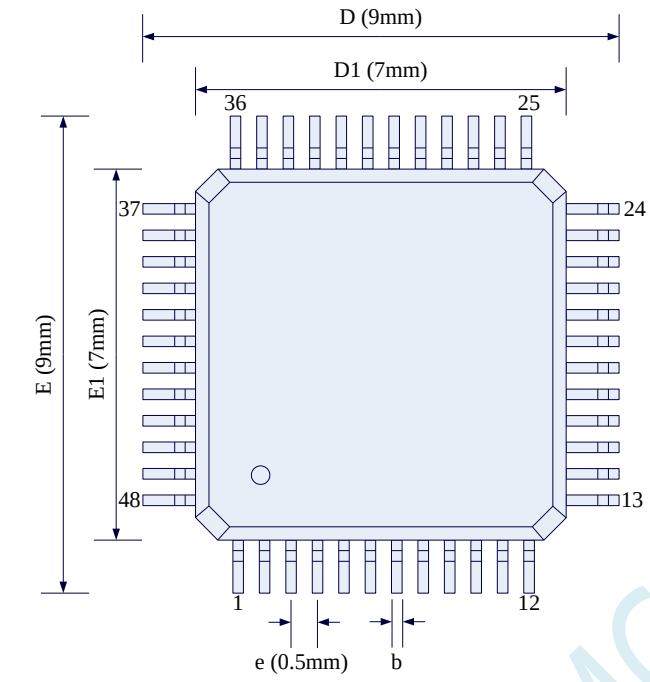
4 封装尺寸图

4.1 LQFP44 封装尺寸图（12mm*12mm）

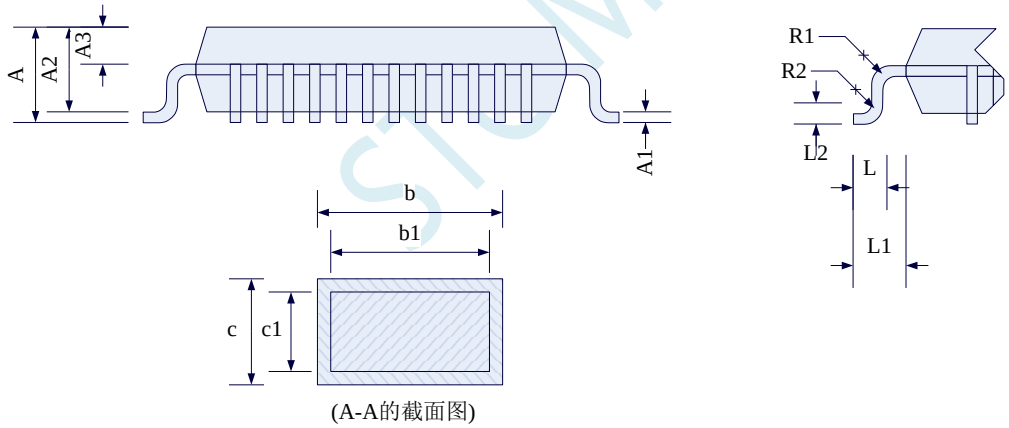


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.33	-	0.42
b1	0.32	0.35	0.38
c	0.13	-	0.18
c1	0.12	0.13	0.14
D	11.95	12.00	12.05
D1	9.90	10.00	10.10
E	11.95	12.00	12.05
E1	9.90	10.00	10.10
e	0.70	0.80	0.90
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

4.2 LQFP48 封装尺寸图（9mm*9mm）

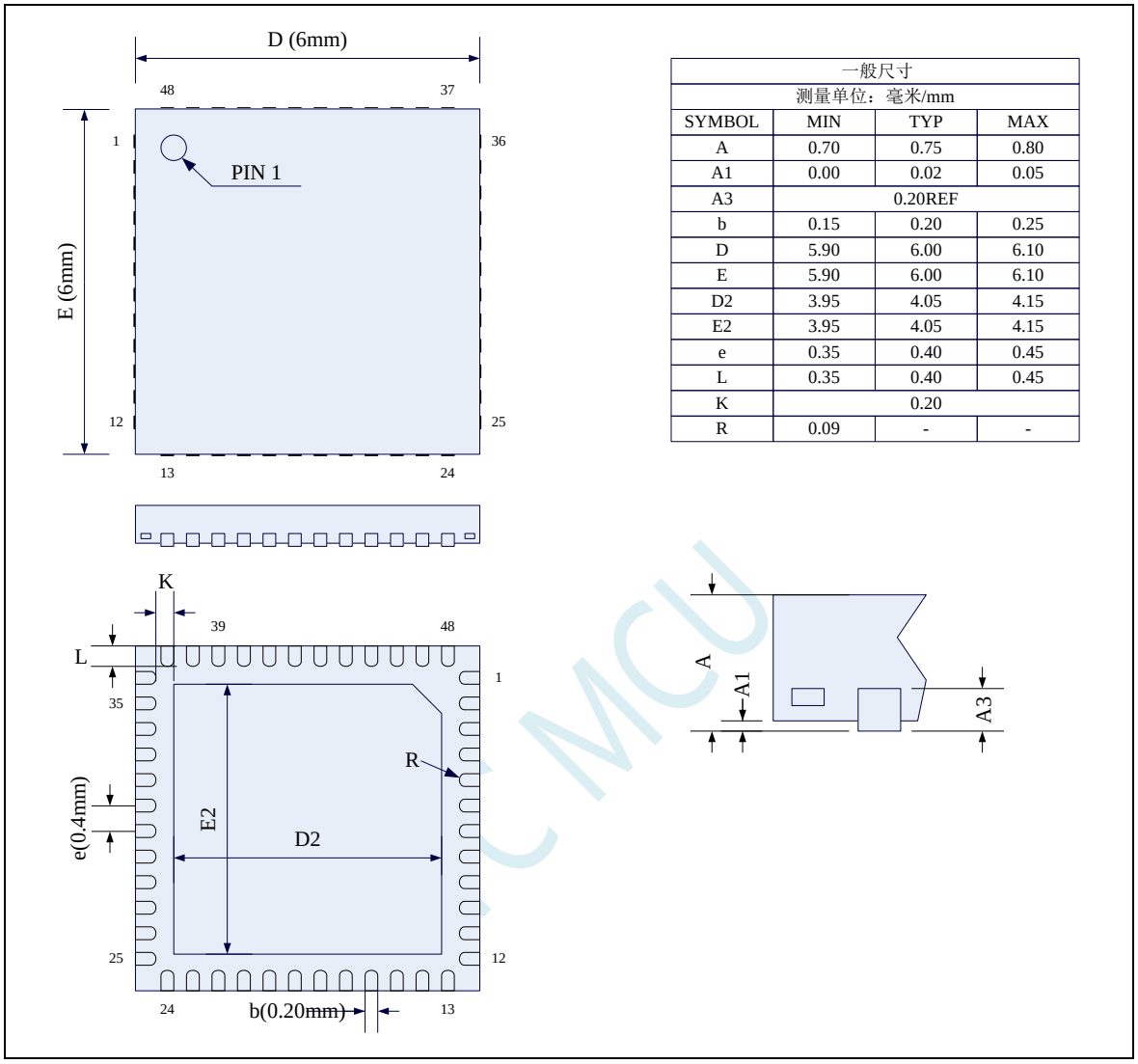


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.45	0.50	0.55
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-



(A-A的截面图)

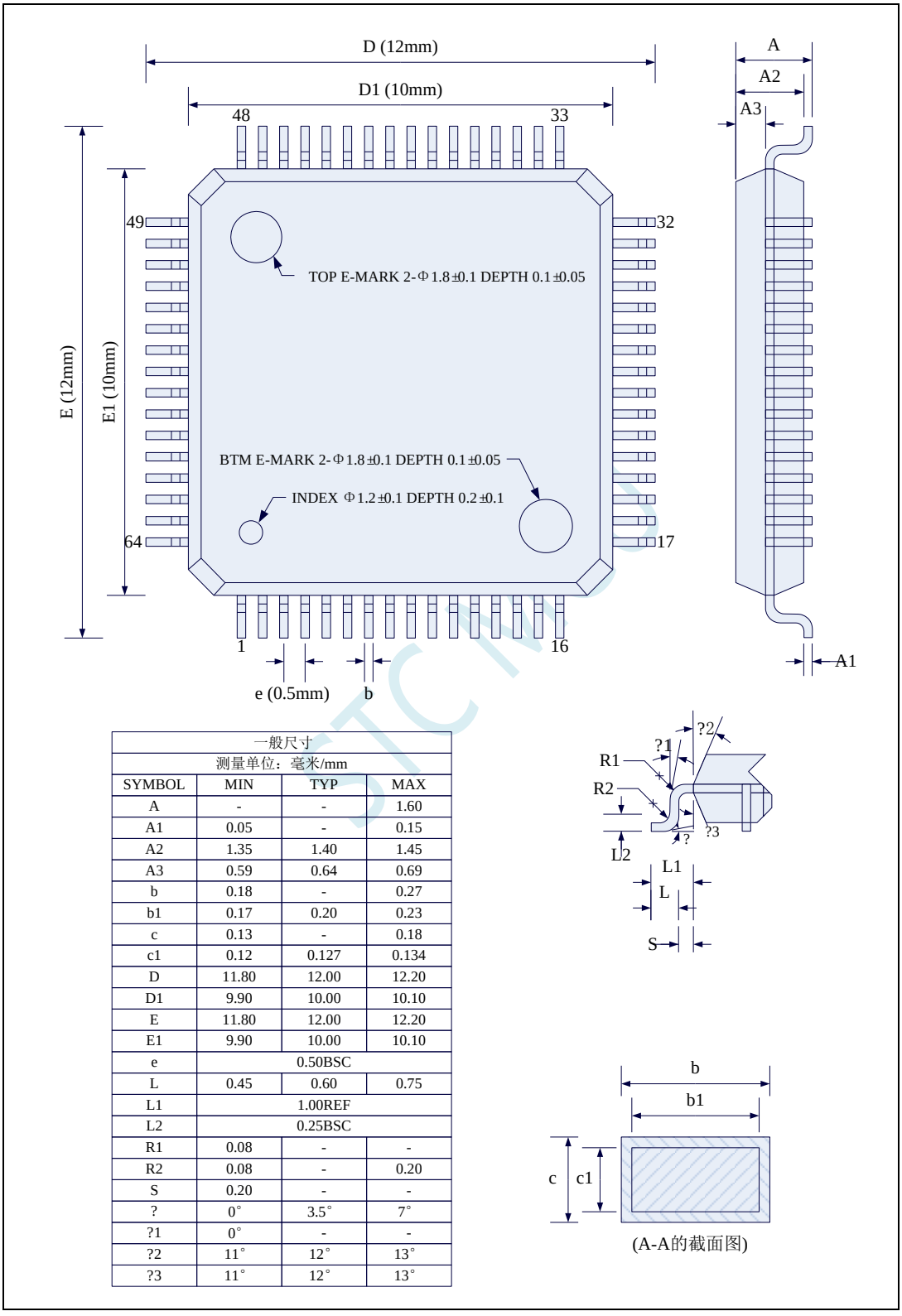
4.3 QFN48 封装尺寸图（6mm*6mm）



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3		0.20REF	
b	0.15	0.20	0.25
D	5.90	6.00	6.10
E	5.90	6.00	6.10
D2	3.95	4.05	4.15
E2	3.95	4.05	4.15
e	0.35	0.40	0.45
L	0.35	0.40	0.45
K		0.20	
R	0.09	-	-

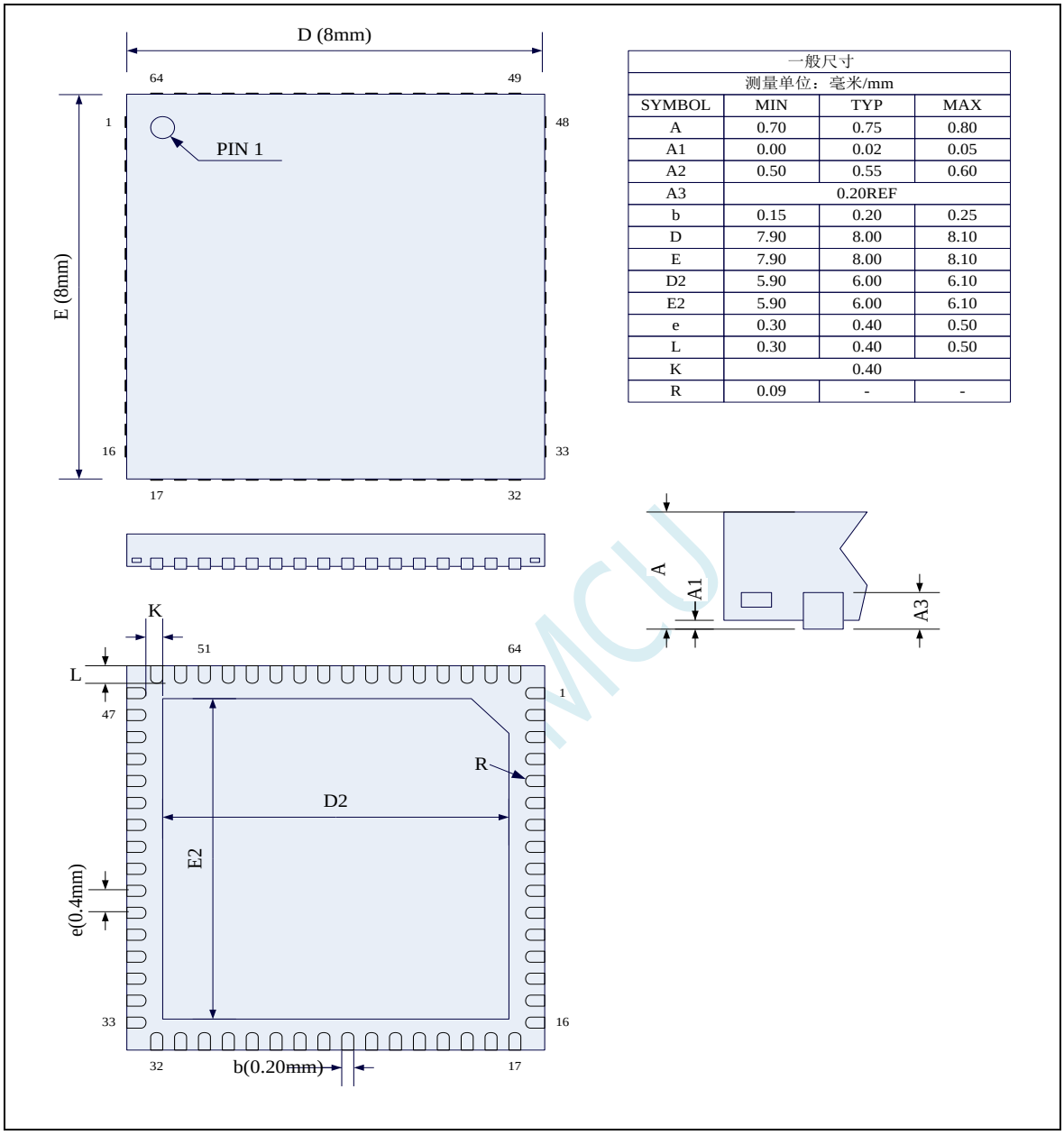
STC 现有 QFN48 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

4.4 LQFP64 封装尺寸图（12mm*12mm）



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
?	0°	3.5°	7°
?1	0°	-	-
?2	11°	12°	13°
?3	11°	12°	13°

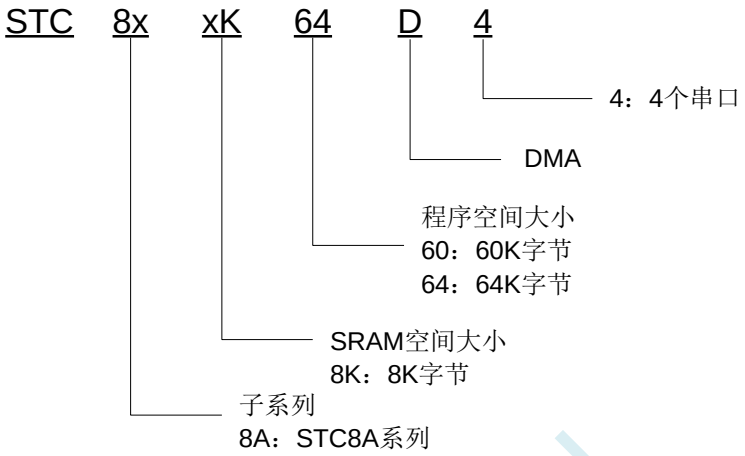
4.5 QFN64 封装尺寸图（8mm*8mm）



一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A2	0.50	0.55	0.60
A3		0.20REF	
b	0.15	0.20	0.25
D	7.90	8.00	8.10
E	7.90	8.00	8.10
D2	5.90	6.00	6.10
E2	5.90	6.00	6.10
e	0.30	0.40	0.50
L	0.30	0.40	0.50
K		0.40	
R	0.09	-	-

STC 现有 QFN64 封装芯片的背面金属片（衬底），在芯片内部并未接地，在用户的 PCB 板上可以接地，也可以不接地，不会对芯片性能造成影响

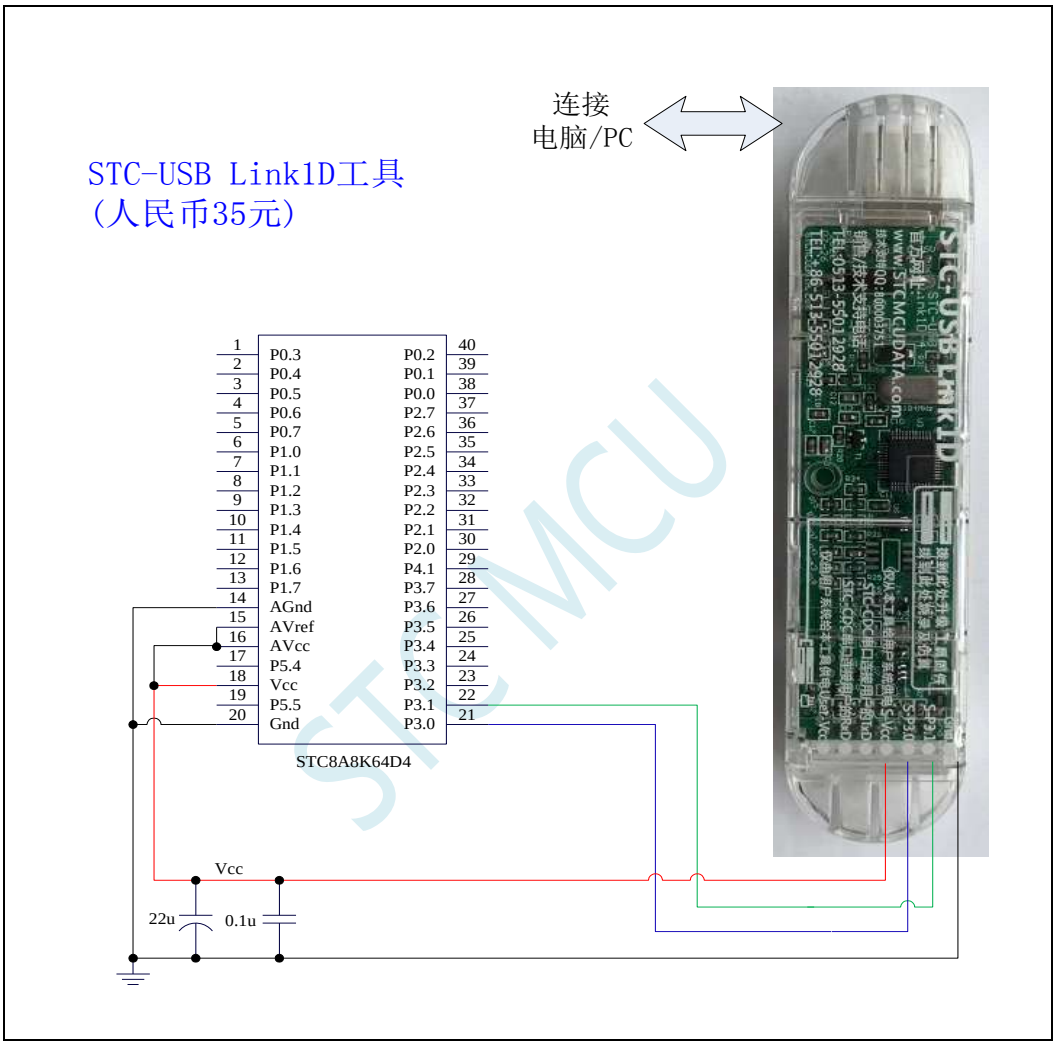
4.6 STC8A8K64D4 系列单片机命名规则



5 ISP 下载及典型应用线路图

5.1 STC8A8K64D4 系列 ISP 下载应用线路图

5.1.1 使用 STC-USB Link1D 工具下载，支持在线和脱机下载



ISP 下载步骤：

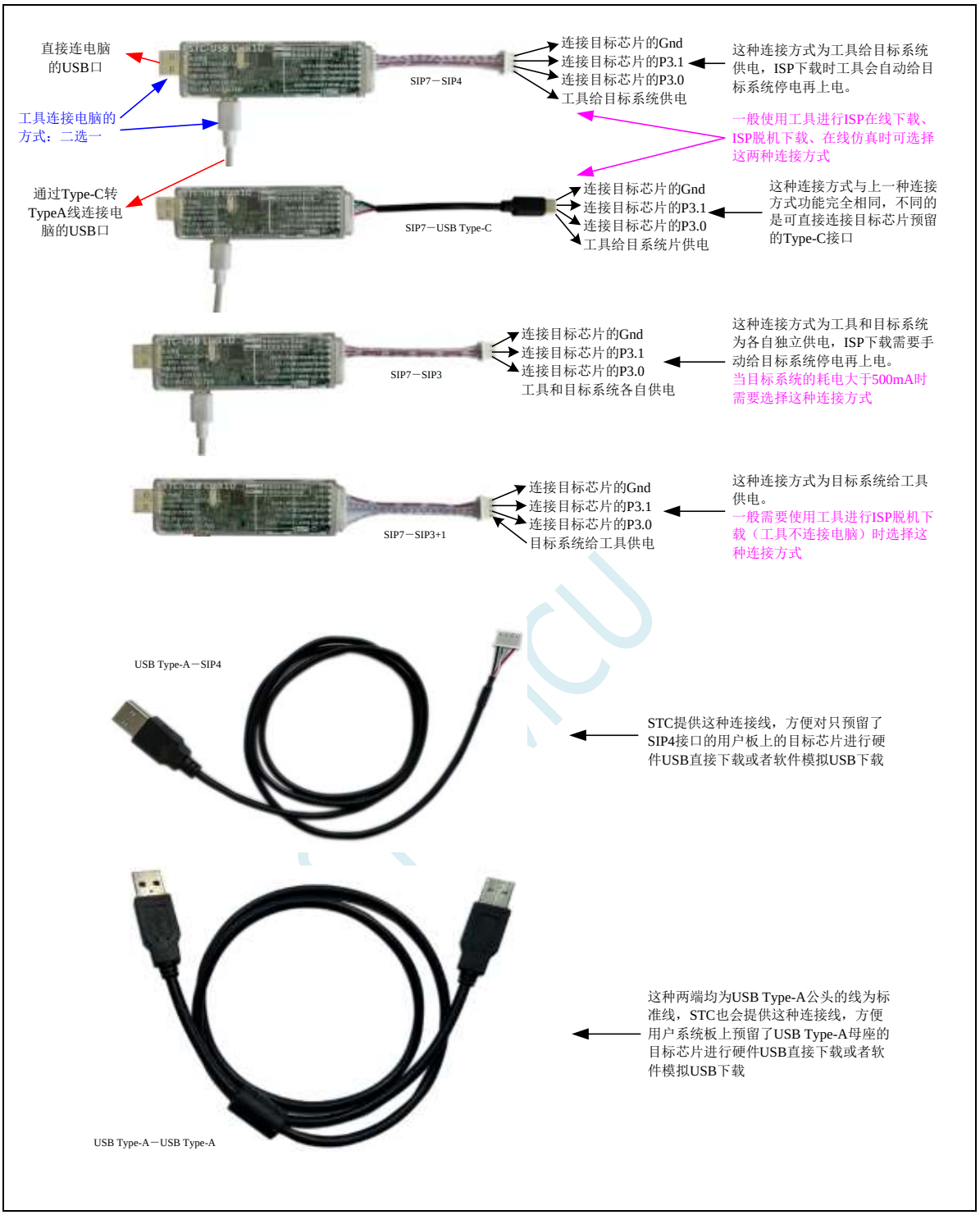
1、 按照如图所示的连接方式将 STC-USB Link1D 和目标芯片连接

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

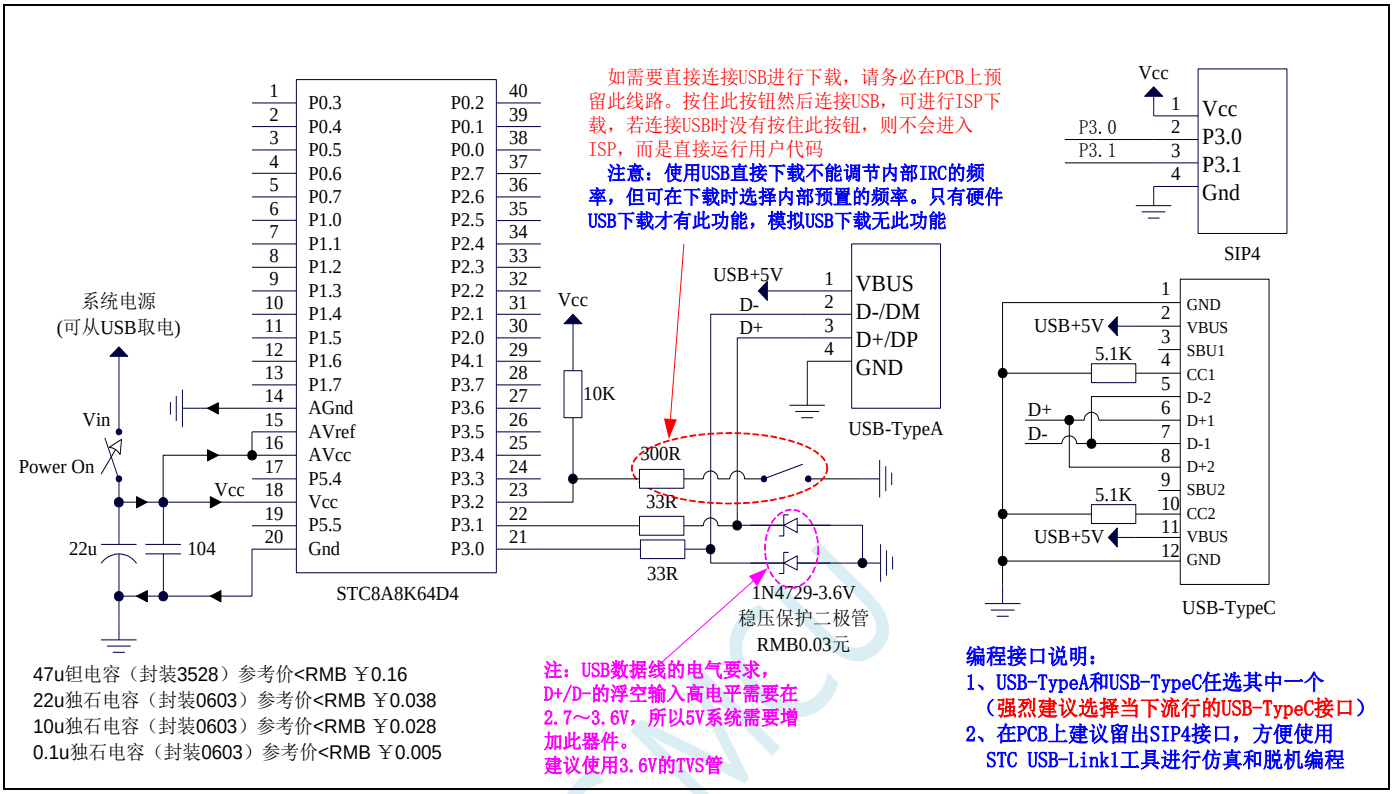
3、 开始 ISP 下载

注意：若是使用 STC-USB Link1D 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

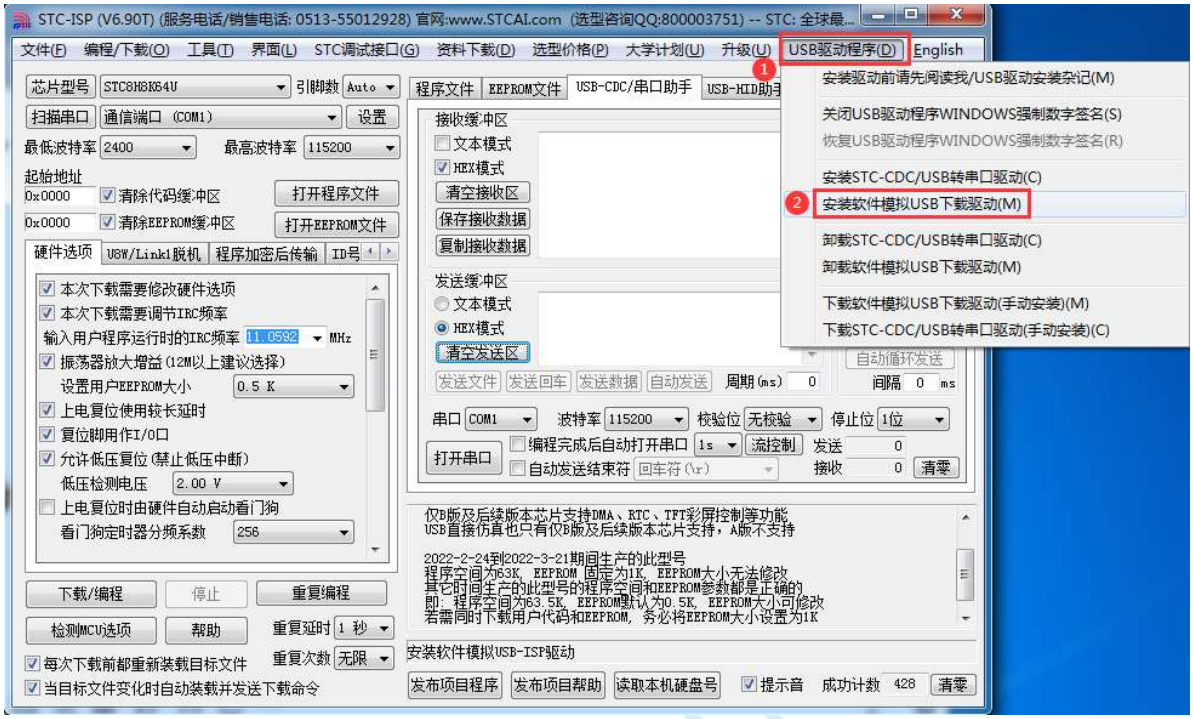
注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。


直接连电脑的USB口
工具连接电脑的方式：二选一
SIP7－SIP4
连接目标芯片的Gnd
连接目标芯片的P3.1
连接目标芯片的P3.0
工具给目标系统供电
这种连接方式为工具给目标系统供电，ISP下载时工具会自动给目标系统停电再上电。
一般使用工具进行ISP在线下载、ISP脱机下载、在线仿真时可选择这两种连接方式
通过Type-C转TypeA线连接电脑的USB口
SIP7－USB Type-C
连接目标芯片的Gnd
连接目标芯片的P3.1
连接目标芯片的P3.0
工具给目标系统片供电
这种连接方式与上一种连接方式功能完全相同，不同的是可直接连接目标芯片预留的Type-C接口
SIP7－SIP3
连接目标芯片的Gnd
连接目标芯片的P3.1
连接目标芯片的P3.0
工具和目标系统各自供电
这种连接方式为工具和目标系统为各自独立供电，ISP下载需要手动给目标系统停电再上电。
当目标系统的耗电大于500mA时需要选择这种连接方式
SIP7－SIP3+1
连接目标芯片的Gnd
连接目标芯片的P3.1
连接目标芯片的P3.0
目标系统给工具供电
这种连接方式为目标系统给工具供电。
一般需要使用工具进行ISP脱机下载（工具不连接电脑）时选择这种连接方式
USB Type-A－SIP4
STC提供这种连接线，方便对只预留了SIP4接口的用户板上的目标芯片进行硬件USB直接下载或者软件模拟USB下载
USB Type-A－USB Type-A
这种两端均为USB Type-A公头的线为标准线，STC也会提供这种连接线，方便用户系统板上预留了USB Type-A母座的目标芯片进行硬件USB直接下载或者软件模拟USB下载

5.1.2 软件模拟 USB 直接 ISP 下载，建议尝试，不支持仿真（5V 系统）



现在 STC 的不带硬件 USB 的 STC8G/STC8H 的 MCU，基本都支持用软件模拟 USB 下载用户程序，因为走的是 USB-SCAN 通信协议，不管任何版本操作系统，都要安装驱动。在 STC-ISP 下载软件如下图所示的地方安装驱动。



ISP 下载步骤：

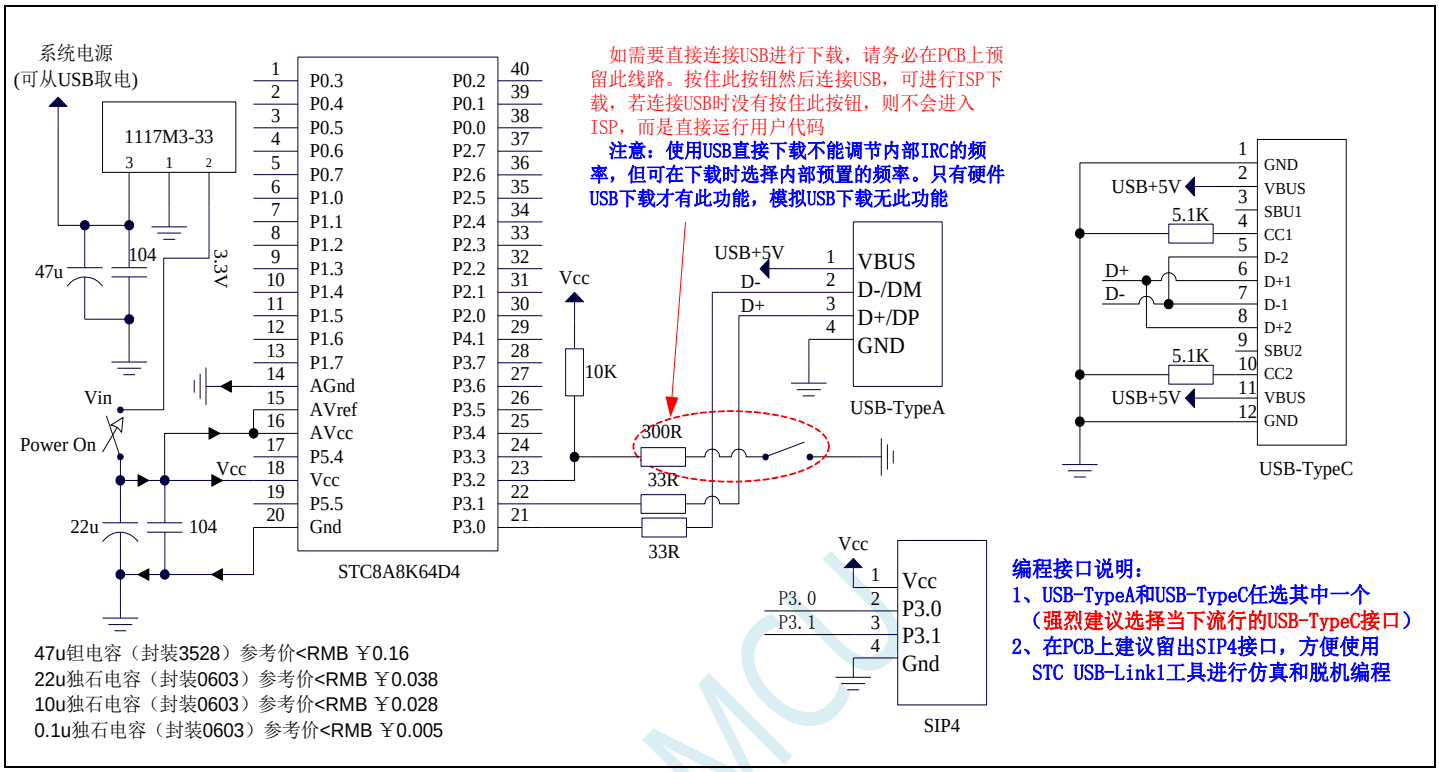
1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好

2、将 P3.2 与 GND 短接，实验箱板子上的 P3.2/INT0 按键按下

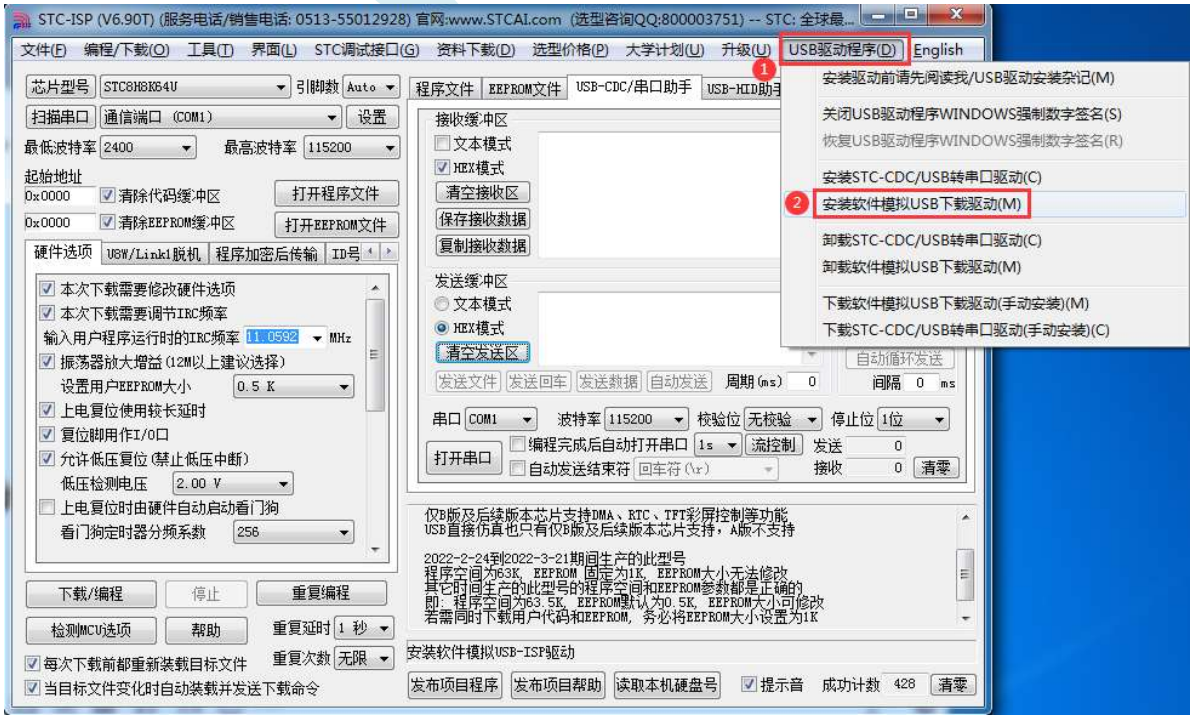
3、给目标芯片重新上电。若目标芯片已经停电，直接上电即可；若目标芯片处于通电状态，则需给目标芯片停电再上电（冷启动）。等待 STC-ISP 下载软件中自动识别出“STC USB Writer (HID1)”识别出来后,就与 P3.2 状态无关了（此时就不需要一直按着 P3.2 口了，一直按着把手累坏了不要紧，将按键按坏就麻烦了）。

4、点击下载软件中的“下载/编程”按钮（注意：USB 下载与串口下载的操作顺序不同，**千万千万不要先点下载按钮，一定到等到电脑端识别出“STC USB Writer (HID1)”设备后，才能点下载按钮开始下载**）

5.1.3 软件模拟 USB 直接 ISP 下载，建议尝试，不支持仿真（3.3V 系统）



现在 STC 的不带硬件 USB 的 STC8G/STC8H 的 MCU，基本都支持用软件模拟 USB 下载用户程序，因为走的是 USB-SCAN 通信协议，不管任何版本操作系统，都要安装驱动。在 STC-ISP 下载软件如下图所示的地方安装驱动。



ISP 下载步骤：

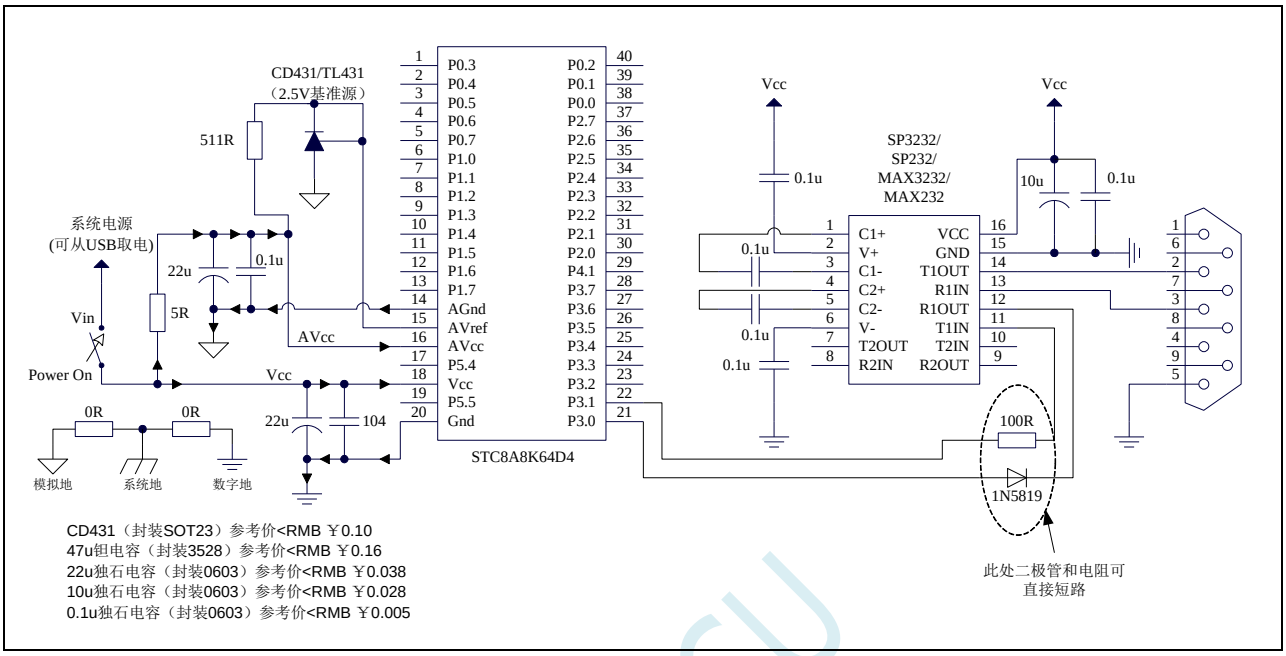
1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好

2、将 P3.2 与 GND 短接，实验箱板子上的 P3.2/INT0 按键按下

3、给目标芯片重新上电。若目标芯片已经停电，直接上电即可；若目标芯片处于通电状态，则需给目标芯片停电再上电（冷启动）。等待 STC-ISP 下载软件中自动识别出“STC USB Writer (HID1)”识别出来后,就与 P3.2 状态无关了（此时就不需要一直按着 P3.2 口了，一直按着把手累坏了不要紧，将按键按坏就麻烦了）。

4、点击下载软件中的“下载/编程”按钮（注意：USB 下载与串口下载的操作顺序不同，**千万千万不要先点下载按钮，一定到等到电脑端识别出“STC USB Writer (HID1)”设备后，才能点下载按钮开始下载**）

5.1.4 使用 RS-232 转换器下载，也可支持仿真（使用高精度 ADC）



ISP 下载步骤：

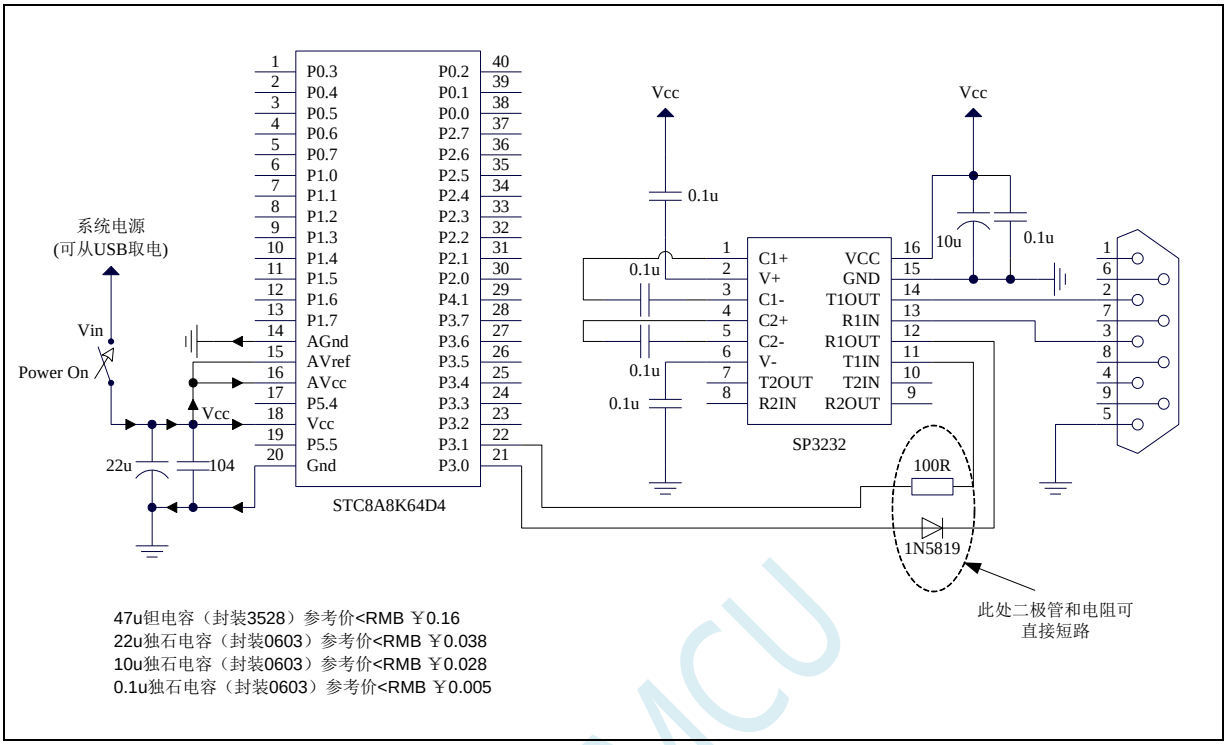
1、给目标芯片停电

2、点击 STC-ISP 下载软件中的“下载/编程”按钮

3、给目标芯片上电

4、开始 ISP 下载

5.1.5 使用 RS-232 转换器下载，也可支持仿真（使用一般精度 ADC）



ISP 下载步骤：

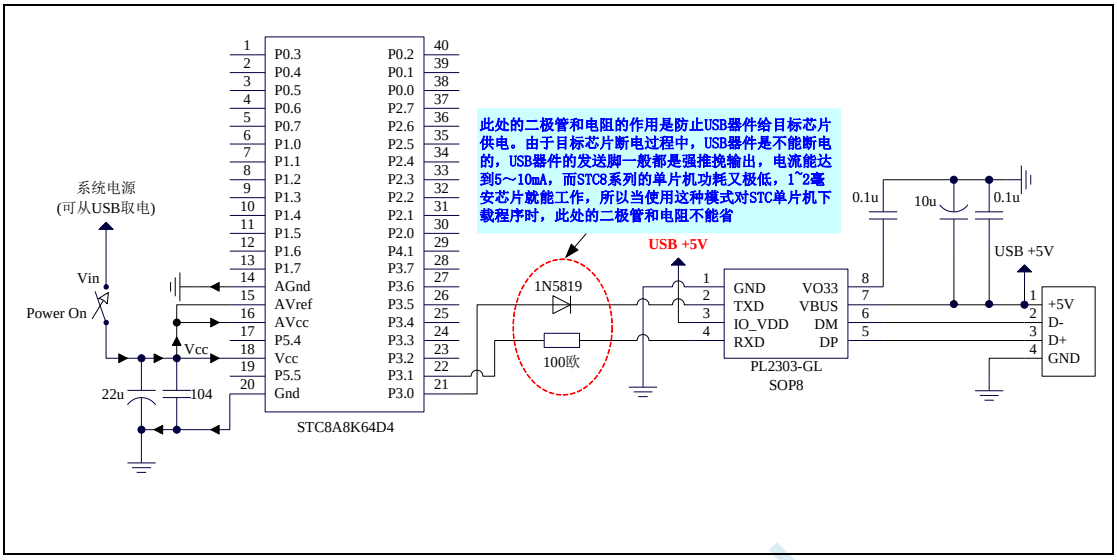
1、 给目标芯片停电

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 给目标芯片上电

4、 开始 ISP 下载

5.1.6 使用 PL2303-GL 下载，也可支持仿真



ISP 下载步骤：

1、 给目标芯片停电，注意不能给 USB 转串口芯片停电（如：CH340、PL2303-GL 等）
注意：PL2303-SA 的部分波特率误差非常大，建议使用 PL2303-GL

2、 由于 USB 转串口芯片的发送脚一般都是强推挽输出，必须在目标芯片的 P3.0 口和 USB 转串口芯片的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

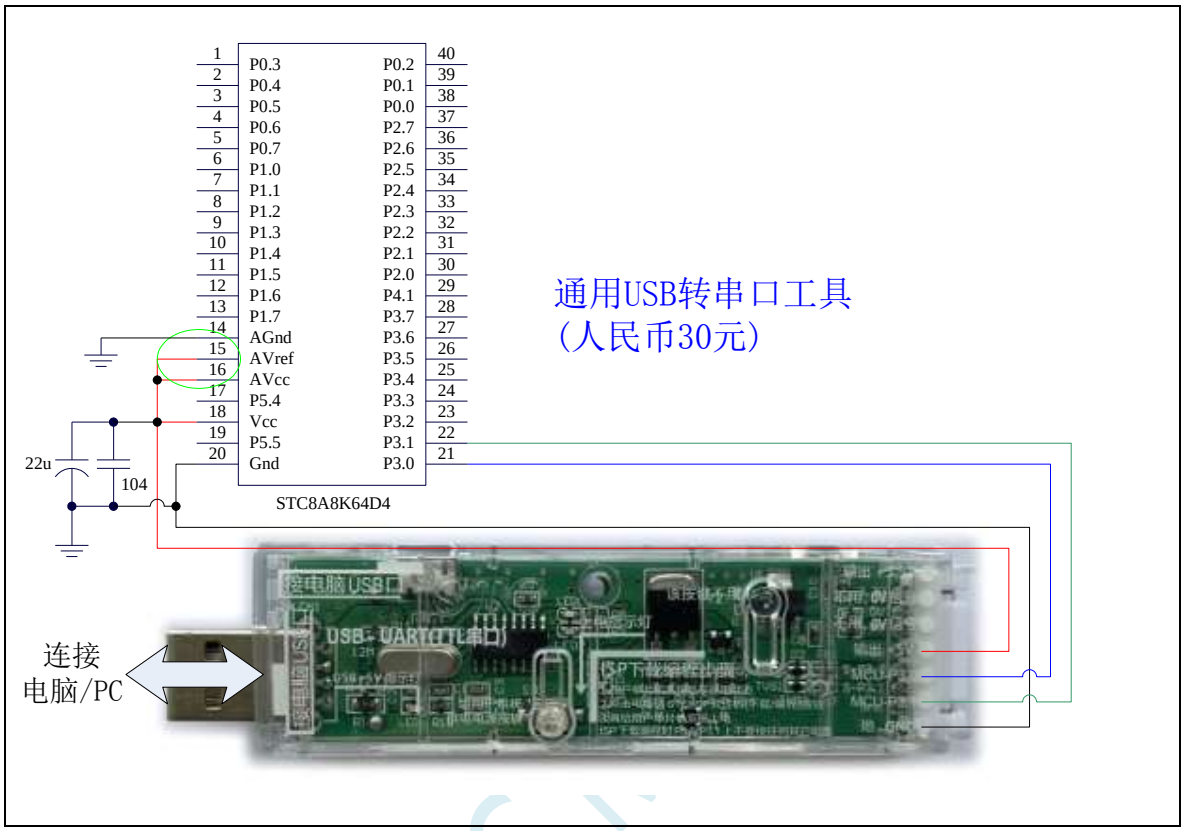
3、 点击 STC-ISP 下载软件中的“下载/编程”按钮

4、 给目标芯片上电

5、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

5.1.7 使用通用 USB 转串口工具下载，支持 ISP 在线下载，也可支持仿真



ISP 下载步骤：

1、 按照如图所示的连接方式将通用 USB 转串口工具和目标芯片连接

2、 按下电源按钮，确定目标芯片处于**停电状态**（上电指示灯为灭的状态）。**注意：工具第一次上电时是不对外供电的，因此若是第一次上电使用此工具，可跳过此步。**

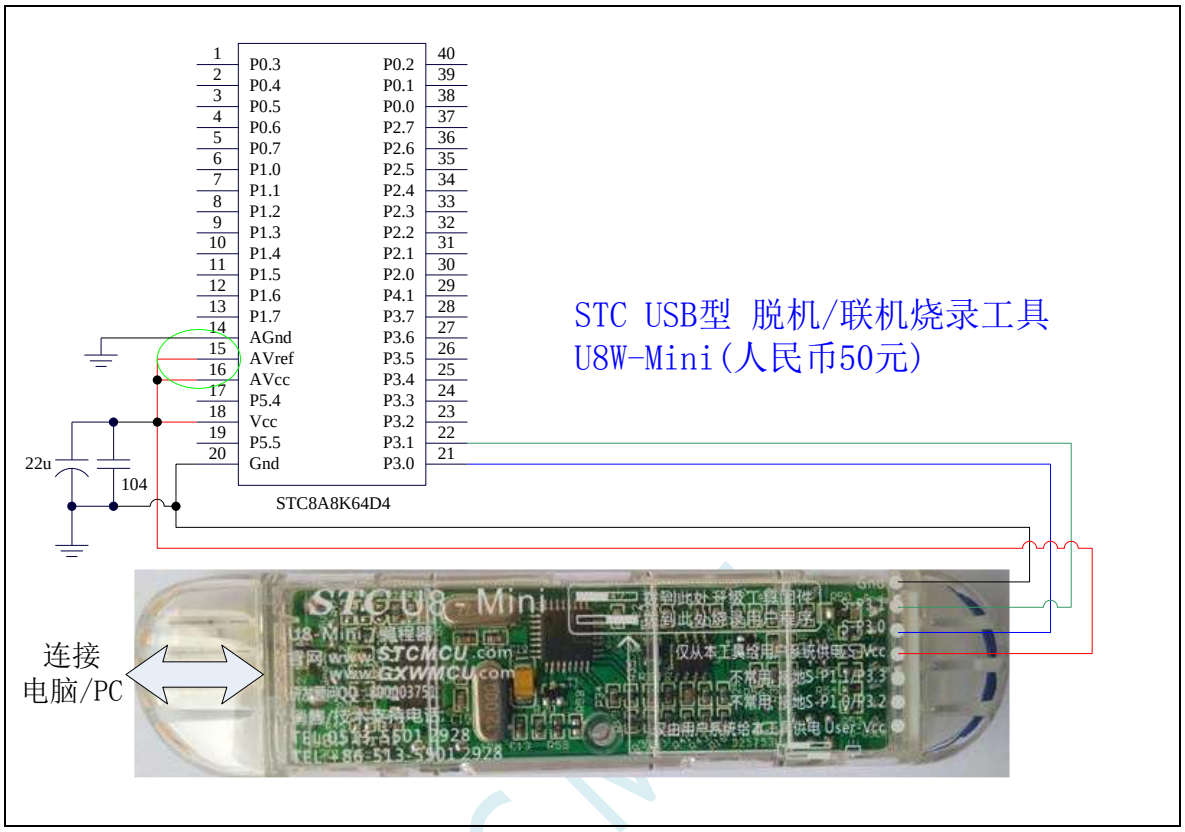
3、 点击 STC-ISP 下载软件中的“下载/编程”按钮

4、 再次按下电源按钮，给目标芯片上电（上电指示灯为亮的状态）

5、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

5.1.8 使用 U8-Mini 工具下载，支持 ISP 在线和脱机下载，也可支持仿真



ISP 下载步骤：

1、 按照如图所示的连接方式将 U8-Mini 和目标芯片连接

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

注意：若是使用 U8-Mini 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

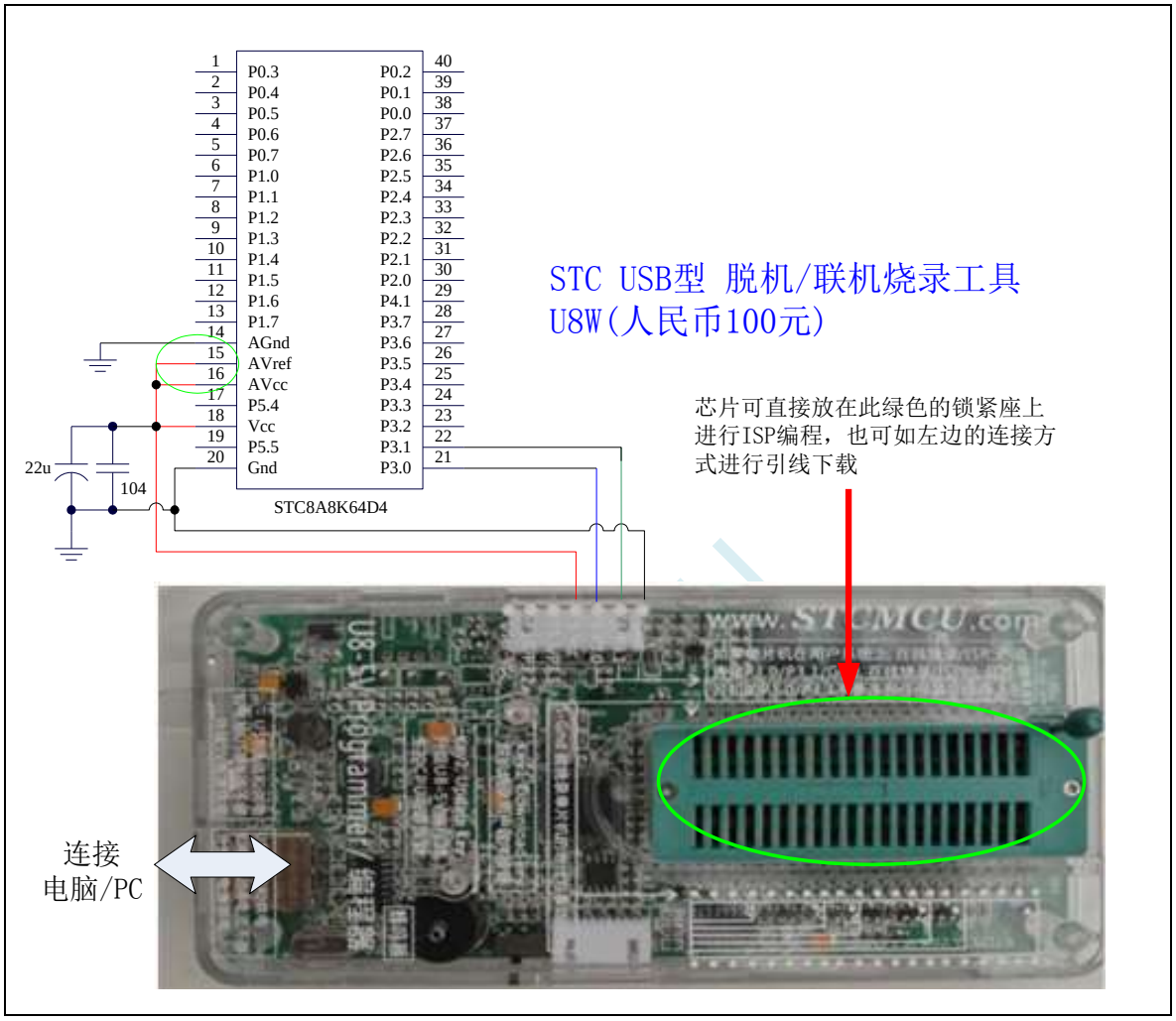
若要使用 U8-Mini 进行仿真，首先必须将 U8-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

1、 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本

2、 U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（**按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1**）

3、 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

5.1.9 使用 U8W 工具下载，支持 ISP 在线和脱机下载，也可支持仿真



ISP 下载步骤（连线方式）：

1、 按照如图所示的连接方式将 U8W 和目标芯片连接

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

注意：若是使用 U8W 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

ISP 下载步骤（在板方式）：

1、 将芯片按照 1 脚靠近锁紧扳手、管脚向下靠齐的方向放置好目标芯片

2、 点击 STC-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

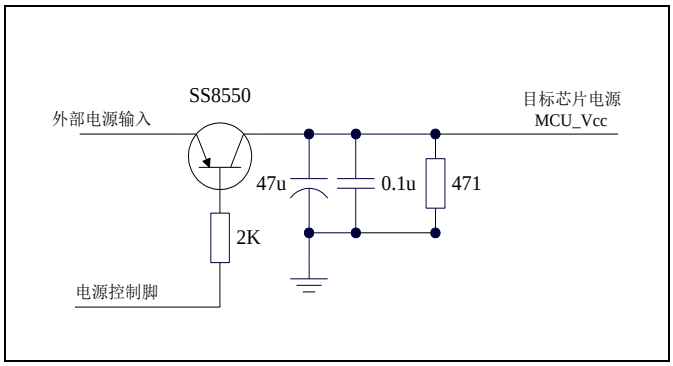
若要使用 U8W 进行仿真，首先必须将 U8W 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

1、 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本

2、 U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（**按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1**）

3、 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

5.1.10 单片机电源控制参考电路



5.2 STC-ISP 下载软件高级应用

5.2.1 发布项目程序

发布项目程序功能主要是将用户的程序代码与相关的选项设置打包成为一个可以直接对目标芯片进行下载编程的超级简单的用户自己界面的可执行文件。

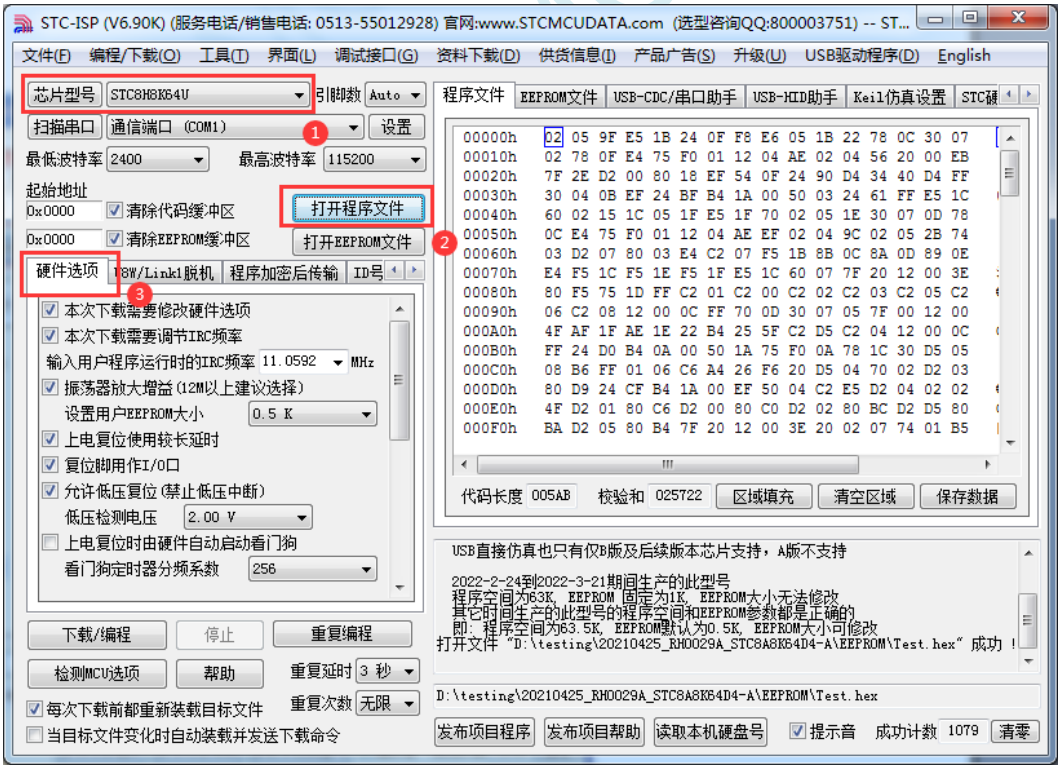
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走,如通过网络发走,如通过U盘烤走,防不胜防,当然盗走你的电脑那就没办法那,所以STC的脱机下载工具比电脑烧录安全,能限制可烧录芯片数量,让前台文员小姐烧,让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---坦克,可以发给客户自己升级,不需冒着生命危险跑到战火纷飞的伊拉克升级软件啦），对于 ID 号不一致的其它芯片，不能进行下载编程。

发布项目程序详细的操作步骤如下：

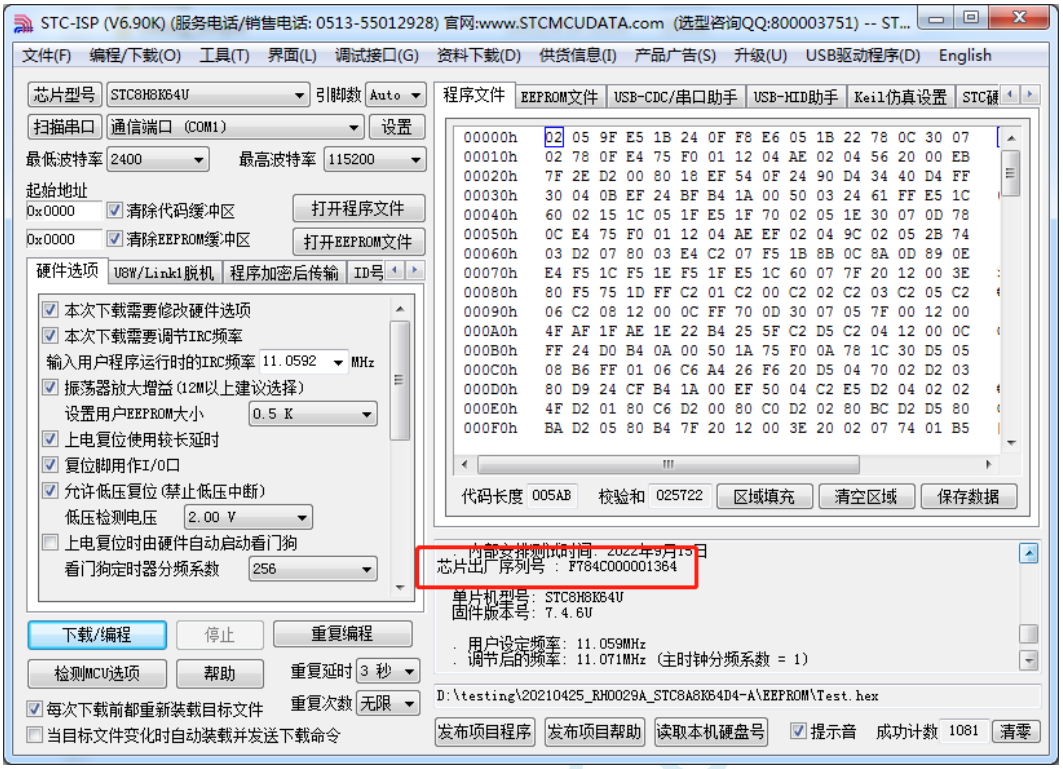
1、首先选择目标芯片的型号

2、打开程序代码文件

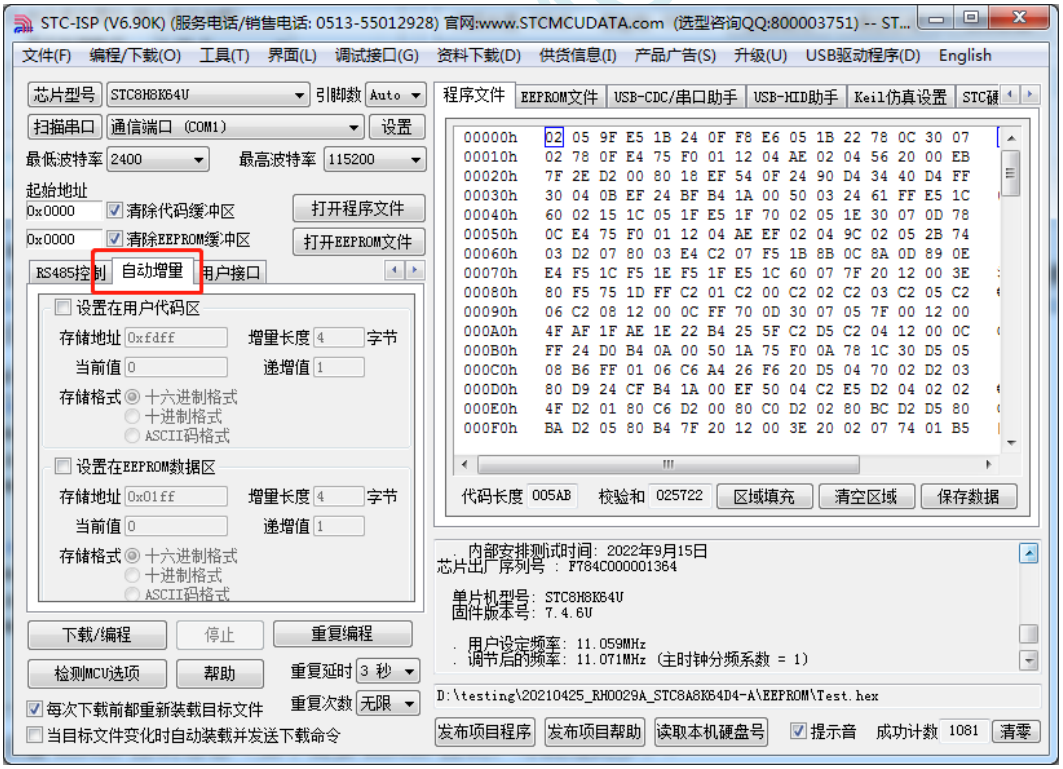
3、设置好相应的硬件选项



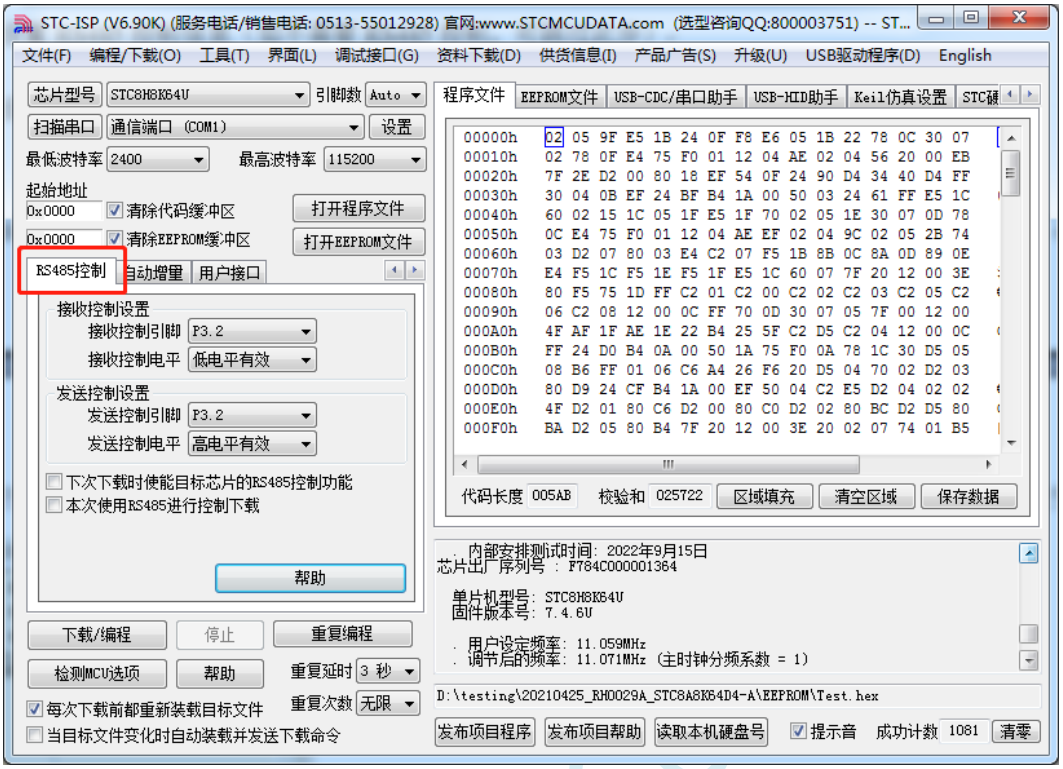
4、试烧一下芯片，并记下目标芯片的 ID 号，如下图所示，该芯片的 ID 号即为“F784C000001364”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



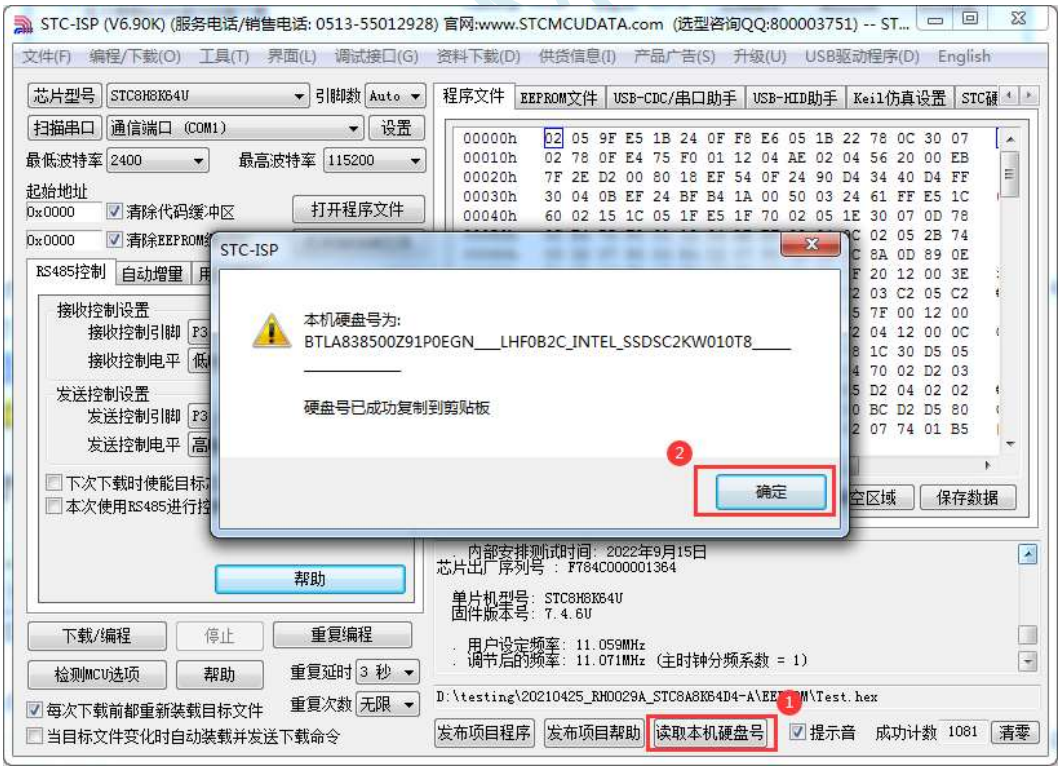
5、设置自动增量（如不需要自动增量，可跳过此步）



6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



7、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）



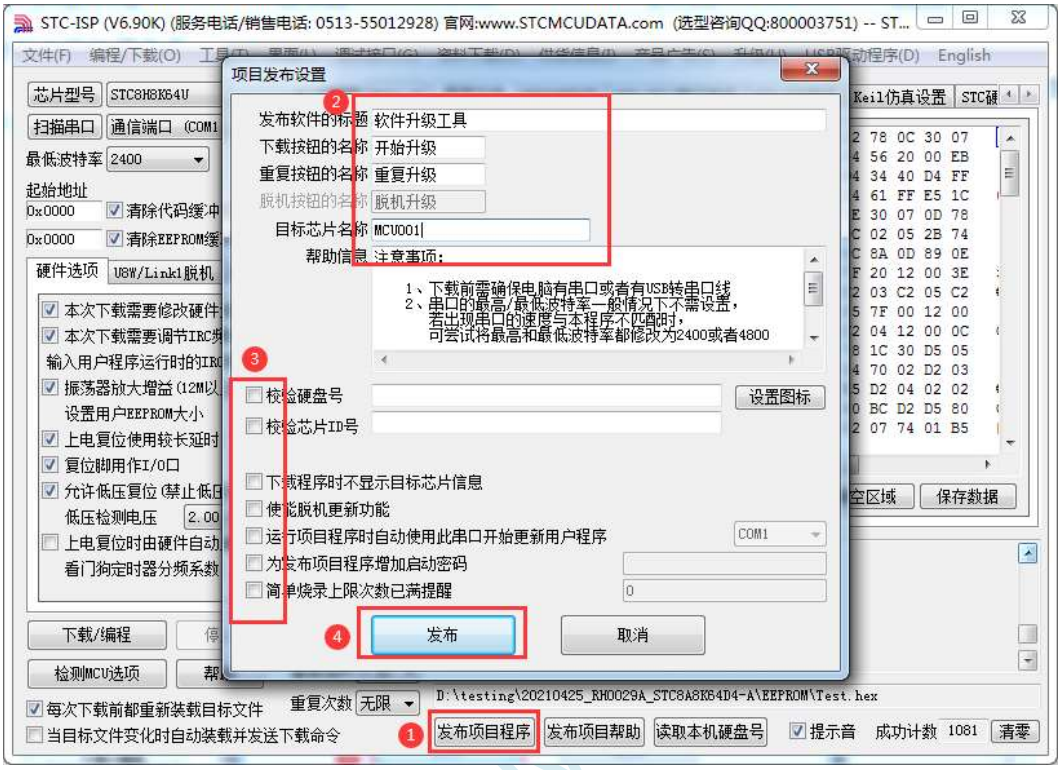
8、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

9、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

10、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记

下的目标电脑的硬盘号

11、若需要校验目标芯片的 ID 号，则需要勾选上“校验芯片 ID 号”，并在后面的文本框内输入前面所记下的目标芯片的 ID 号



12、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。发布的项目程序打界面如下图

5.2.2 程序加密后传输（防烧录时串口分析出程序）

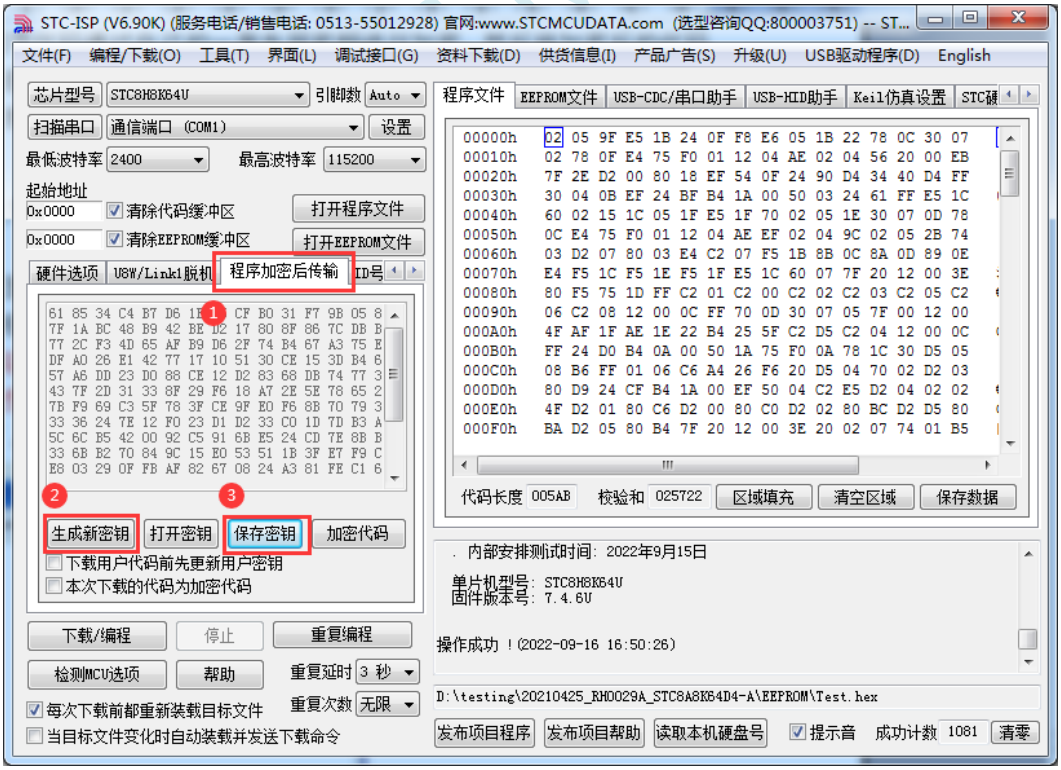
目前，所有的普通串口下载烧录编程都是采用明码通信的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。当然用 STC 的脱机下载板烧程序总比用电脑烧程序强（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘烤走，防不胜防，当然盗走你的电脑那就没办法那，所以 STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以）。即使是 STC 全球首创的脱机下载工具，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据分析出实际的用户程序代码，也是没有办法达到要求的，这就需要用到最新的 STC 单片机所提供的程序加密后传输功能。

程序加密后传输下载是用户先将程序代码通过自己的一套专用密钥进行加密，然后将加密后的代码再通过串口下载，此时下载传输的是加密文件，通过串口分析出来的是加密后的乱码，如不通过派人潜入你公司盗窃你电脑里面的加密密钥，就无任何价值，便可起到防止在烧录程序时被烧录人员通过监测串口分析出代码的目的。

程序加密后传输功能的使用需要如下的几个步骤：

1、生成并保存新的密钥

如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。

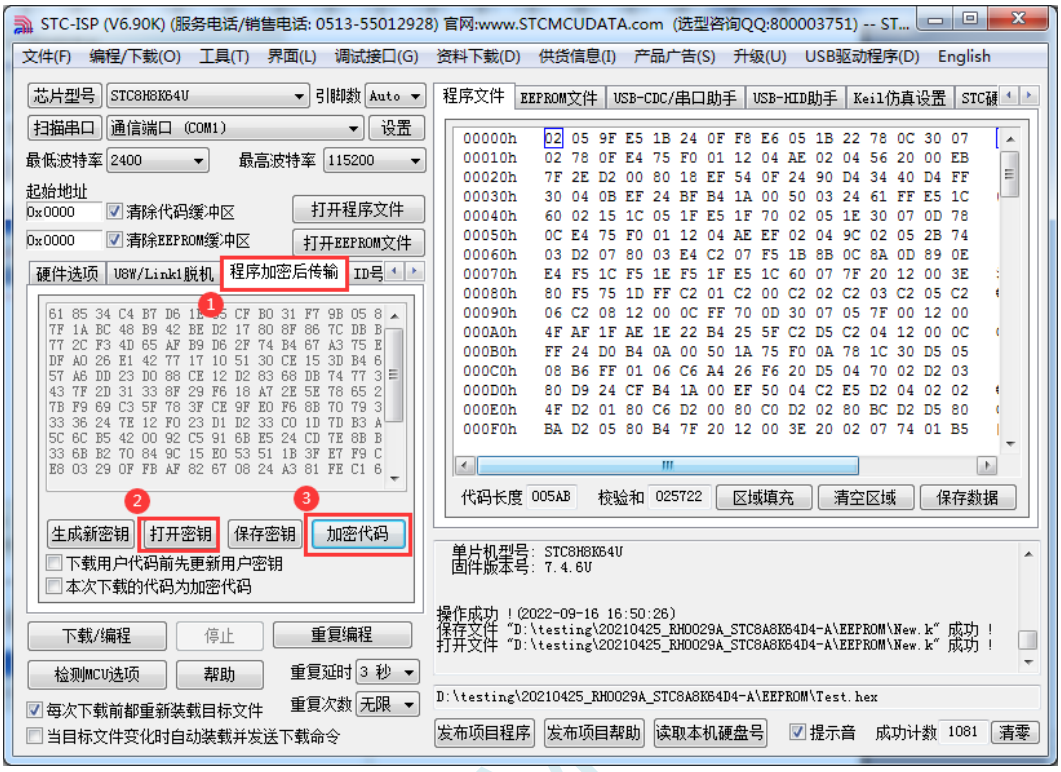


2、对代码文件加密

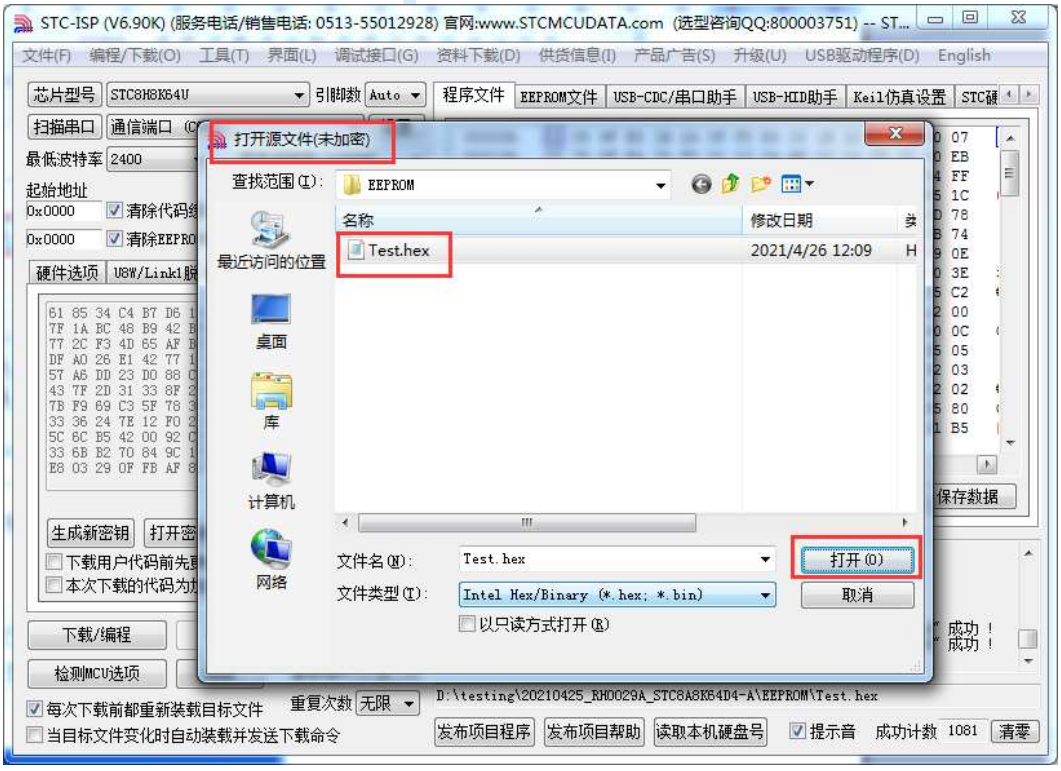
加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。

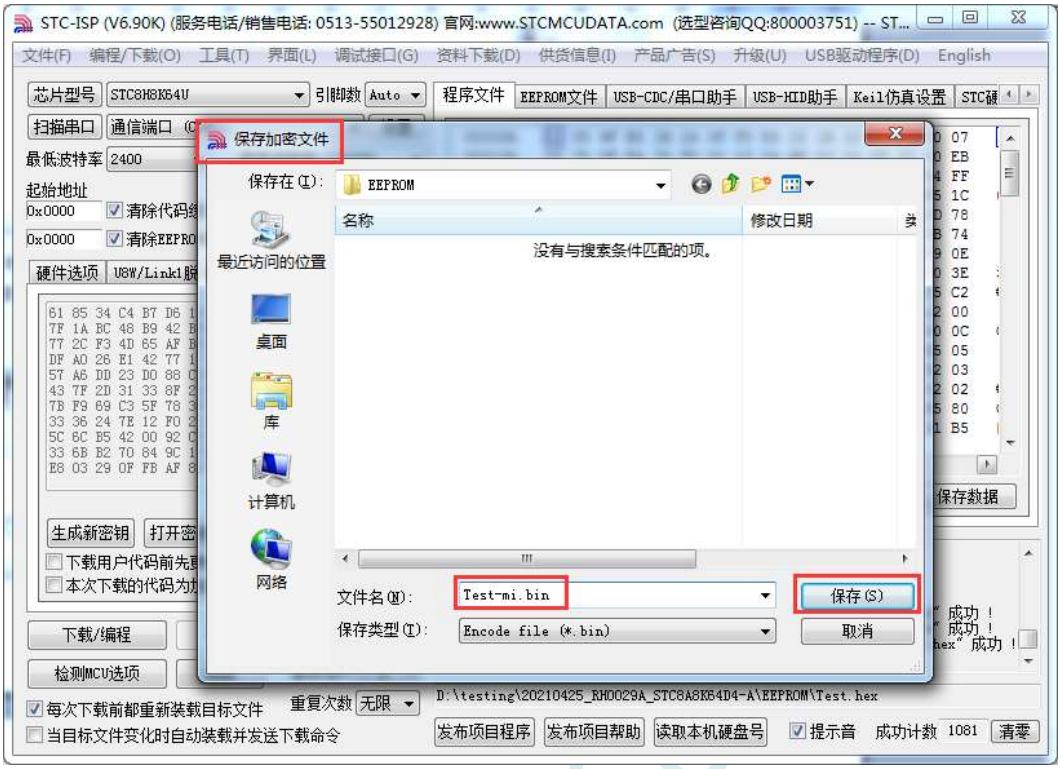
如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，

例如“New.k”，然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件



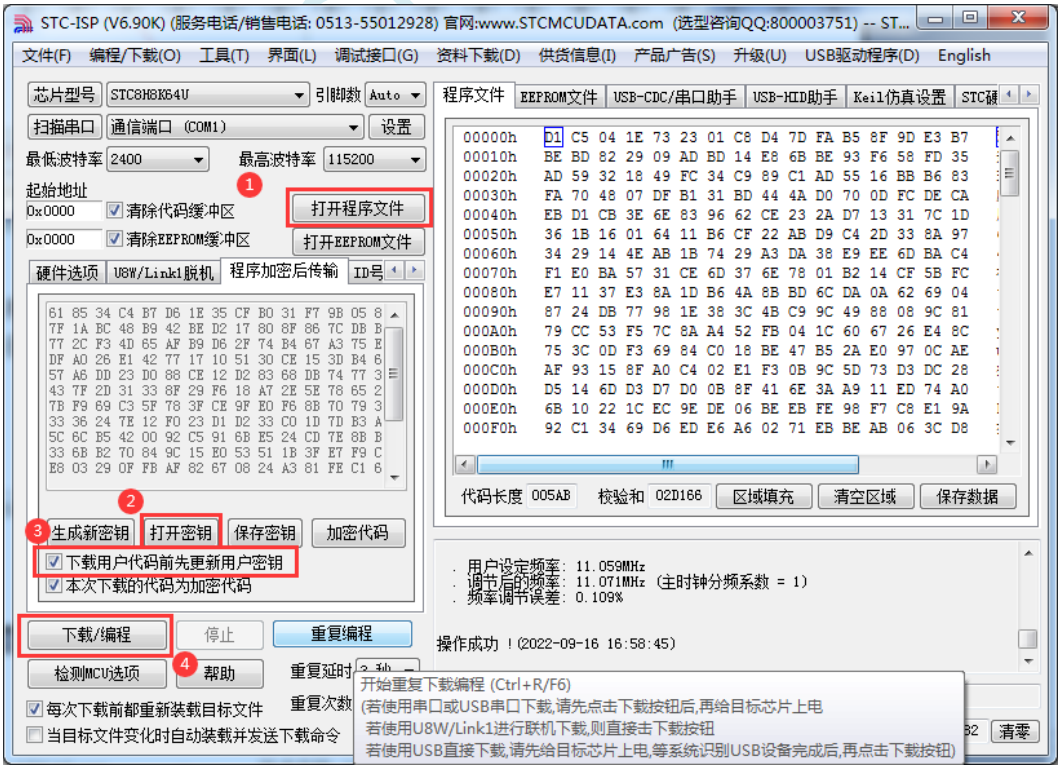
点击打开按钮后，马上有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。





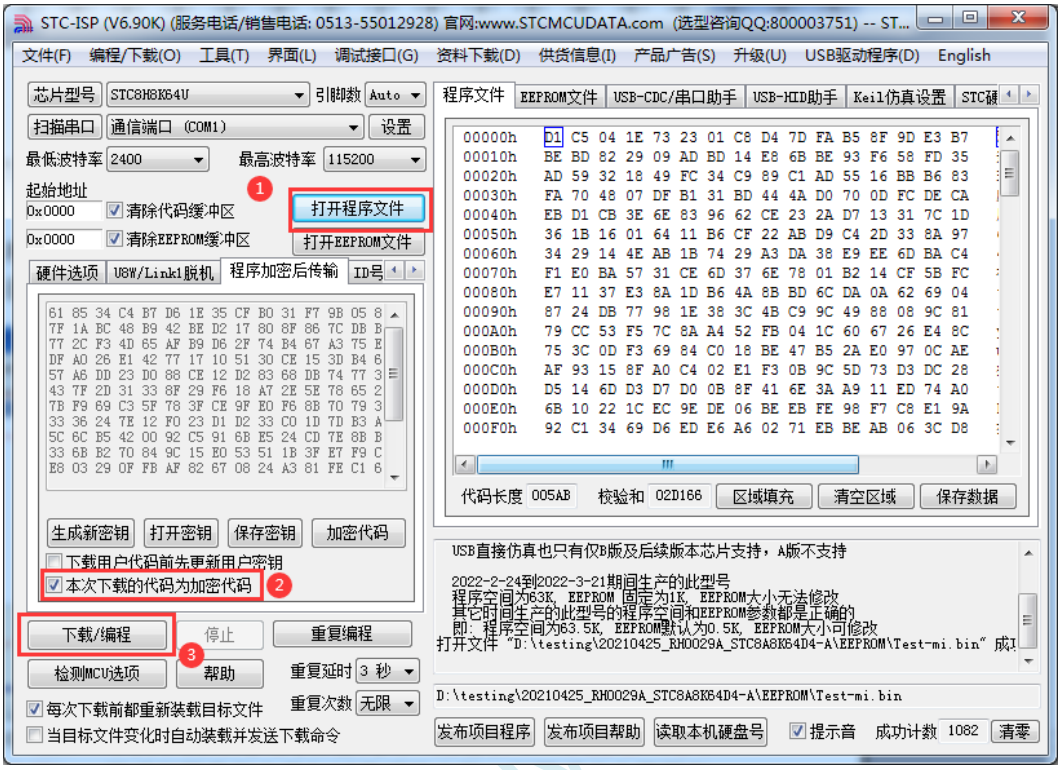
3、将用户密钥更新到目标芯片中

更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



4、加密更新用户代码

密钥更新成功后，目标芯片便具有接收加密代码并还原的功能。此时若需要再次升级/更新代码，则只需要参考第二步的方法，将目标代码进行加密，然后如下图



对于一片新的 STC 单片机，可将步骤 3 和步骤 4 合并完成，即将密钥更新到目标单片机的同时也可将加密后的代码一并下载到单片机中，若已经执行过步骤 3（即已经将密钥更新到目标芯片中了），则后续的代码更新就只需要按照步骤 4，只需要在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项（“下载用户代码前先更新用户密钥”选项不需要选了），然后打开我们之前加过密后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的（防止在烧录程序时被烧录人员通过监测串口分析出代码的目的）。

5.2.3 发布项目程序+程序加密后传输结合使用

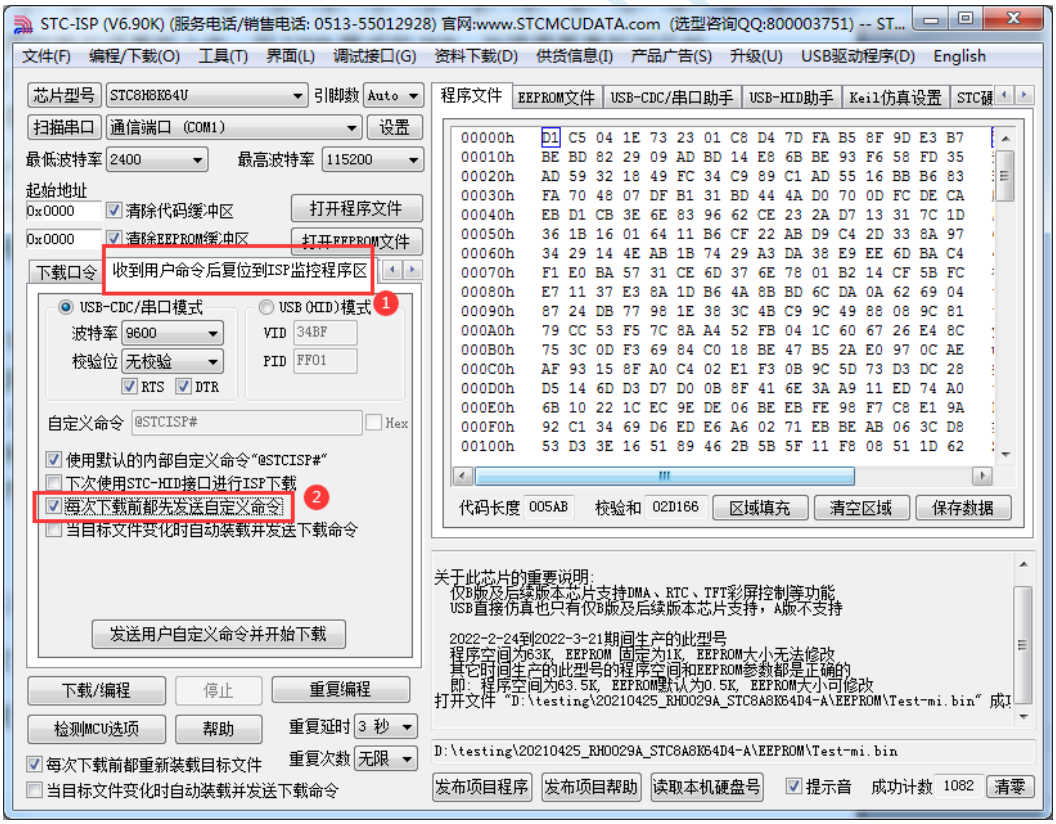
发布项目程序与程序加密后传输两项新的特殊功能可以结合在一起使用。首先程序加密后传输可以确保用户代码在烧录编程时串口通信传输过程当中的保密性，而发布项目程序可实现让最终使用者远程升级功能（方案公司的人员不需要亲自到场）。所以两项功能结合起来使用，非常适用于方案公司/生产商在软件需要更新时，让最终使用者自己对终端产品进行软件更新的目的，又确保现场烧录人员无法通过串口分析出有用程序，强烈建议方案公司使用。

5.2.4 用户自定义下载（实现不停电下载）

将用户的目标程序下载到STC单片机是通过执行单片机内部的ISP系统代码和上位机进行串口或者USB通讯来实现的。但STC单片机内部的ISP系统代码只有在每次重新停电再上电时才会被执行，这就要求用户每次需要对目标单片机更新程序时就必须重新上电，而USB模式的ISP，处理需要重新对目标芯片上电外，还需要在上电时将P3.2口下拉到GND。对于处于开发阶段的项目，需要频繁的修改代码、更新代码，每次下载都需要重新上电会导致操作非常麻烦。

STC单片机在硬件设计时，增加了一个软复位寄存器（IAP_CONTR），让用户可以通过设置此寄存器来决定CPU复位后重新执行用户代码还是复位到ISP区执行ISP系统代码。当向IAP_CONTR寄存器写入0x20时，CPU复位后重新执行用户代码；当向IAP_CONTR寄存器写入0x60时，CPU复位后复位到ISP区执行ISP系统代码。

要实现不停电进行ISP下载，用户可以在程序中设计一段代码，例如检测一个特殊的按键、或者监控串口等待一个特殊的串口命令，当检测到满足下载条件时，就通过软件触发软复位寄存器复位到ISP区执行ISP系统代码，从而实现不停电ISP下载。当触发条件是外部按键时，则在用户代码中实时监控按键状态即可。若要实现STC-ISP软件和用户触发软复位完全同步，则需要使用STC-ISP软件中所提供的“收到用户命令后复位到ISP监控程序区”这个功能。



实现不停电 ISP 下载的步骤如下：

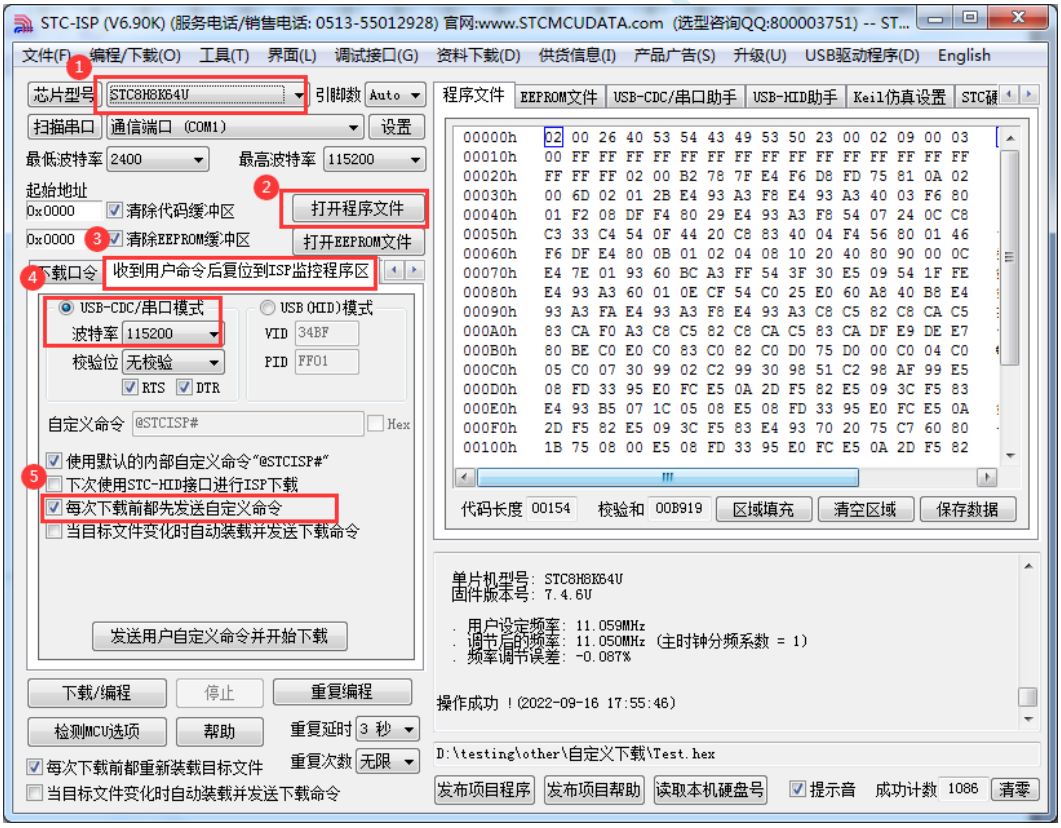
1、编写用户代码，并在用户代码中添加串口命令监控程序

（参考代码如下，测试单片机型号为 STC8H8K64U）

```
#include "stc8h.h"

#define    FOSC        11059200UL
#define    BAUD        (65536 - FOSC/4/115200)

char code *STCISPCMD = "@STCISP#";          //自定义下载命令
char index;

void uart_isr()   interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;                        //接收串口数据

        if (dat == STCISPCMD[index])        //判断接收的数据和当前的命令字符是否匹配
        {
            index++;                        //若匹配则索引+1
            if (STCISPCMD[index] == '\0')   //判断命令是否配完成
                IAP_CONTR = 0x60;           //若匹配完成则软复位到 ISP
        }
        else
        {
            index = 0;                      //若不匹配,则需要从头开始
            if (dat == STCISPCMD[index])
                index++;
        }
    }
}

void main()
{
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
```

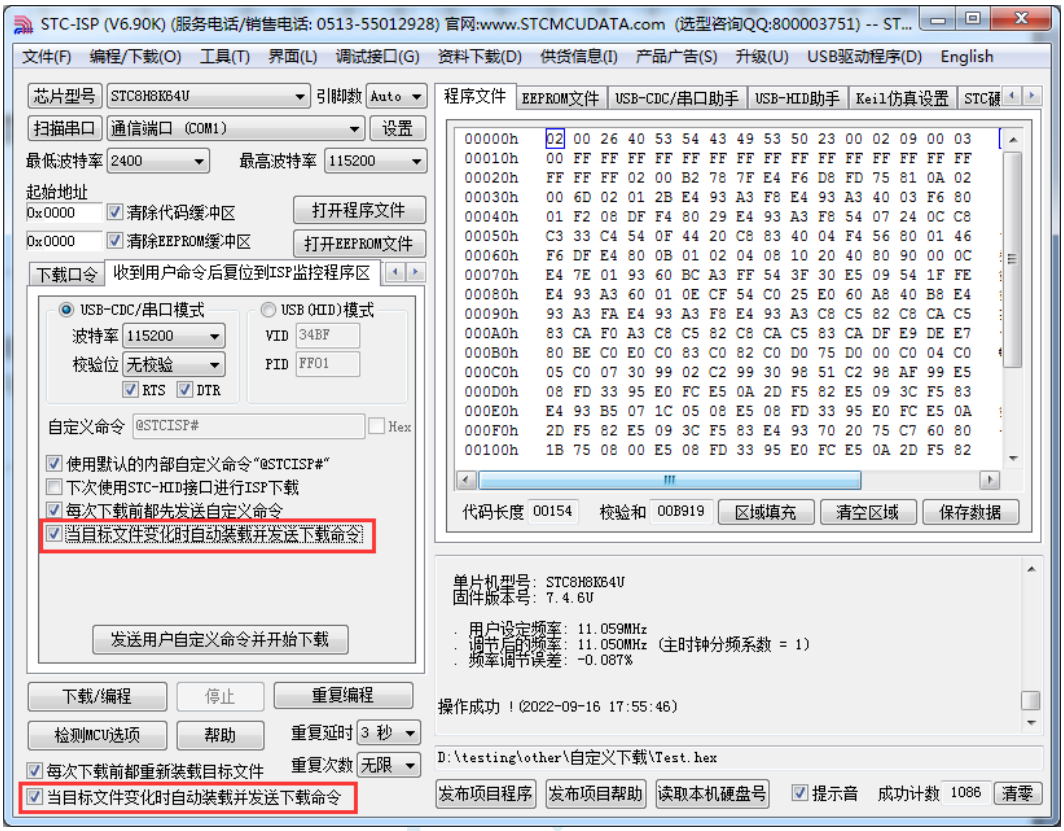
```
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;

    SCON = 0x50;                    //串口初始化
    AUXR = 0x40;
    TMOD = 0x00;
    TH1 = BAUD >> 8;
    TL1 = BAUD;
    TR1 = 1;
    ES = 1;
    EA = 1;

    index = 0;                      //初始化命令

    while (1);
}
```

2、按下图所示的步骤进行设置自定义下载命令（范例使用 STC 默认命令“@STCISP#”）



3、第一次下载时需要对目标单片机重新上电，之后的每次更新只需要点击下载软件中的“下载/编程”按钮，下载软件自动将下载命令发送给目标单片机，目标单片机接收到命令后自动复位到系统 ISP 区，即可实现不停电更新用户代码。

4、STC-ISP 还可实现项目开发阶段，完全自动下载功能，即当下载软件侦测到目标代码被更新了，就会自动发送下载命令。要实现这个功能只需要勾选下图中的两个选项中的任意一个即可

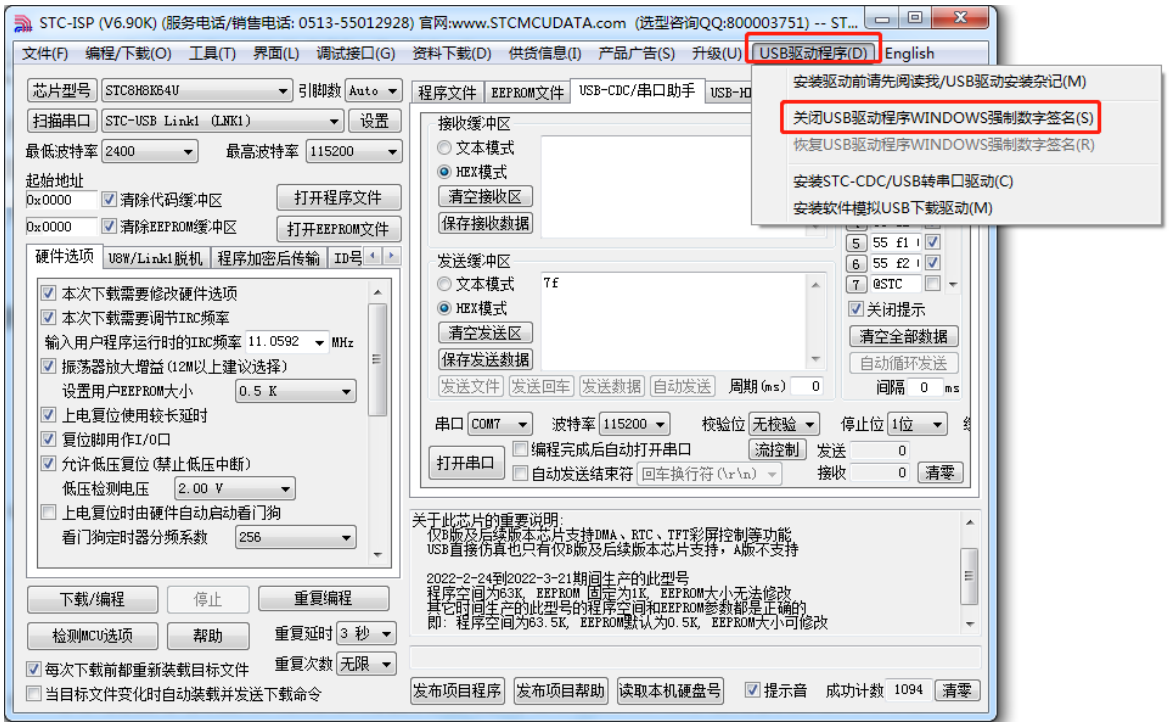


5.3 驱动程序数字签名的相关说明

5.3.1 关于驱动程序强制数字签名

Windows 系统从 Vista 版本开始的 64 位 Windows 系统就对驱动程序强制需要数字签名才能安装，否则驱动无法安装，USB 设备也无法正常使用。目前 STC 提供的驱动程序暂时没有做 WHQL 认证(微软徽标认证)，在没有关闭驱动程序强制数字签名前，在从 Vista 系统开始的 64 位 Windows 系统中可能无法安装成功。

STC-ISP 最新的下载软件提供了一个简单关闭驱动程序强制数字签名的功能，菜单目录如下图所示：

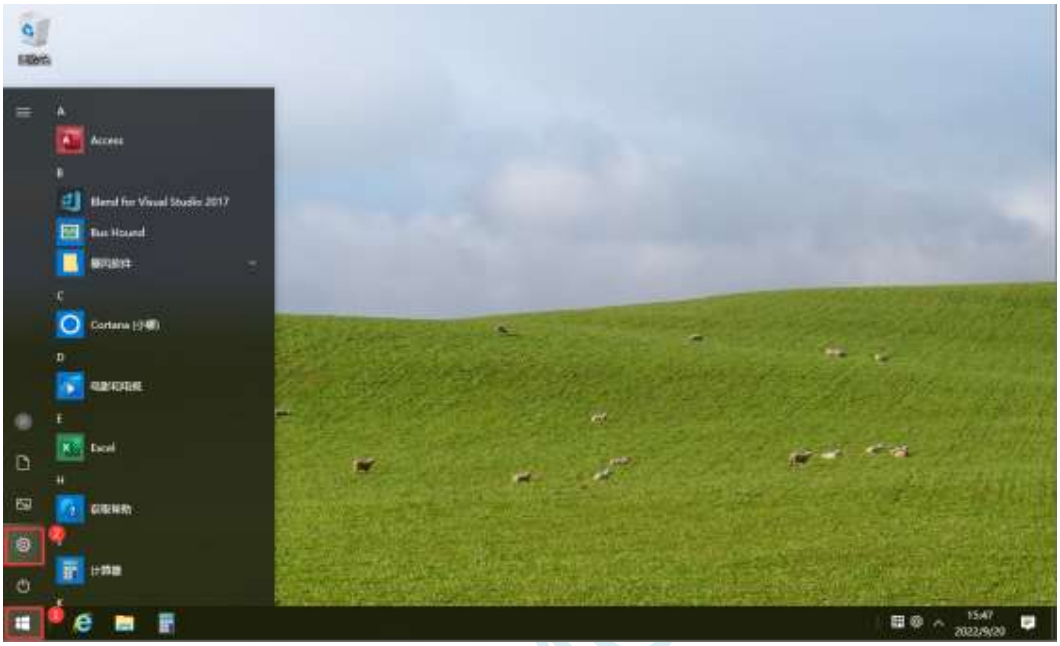


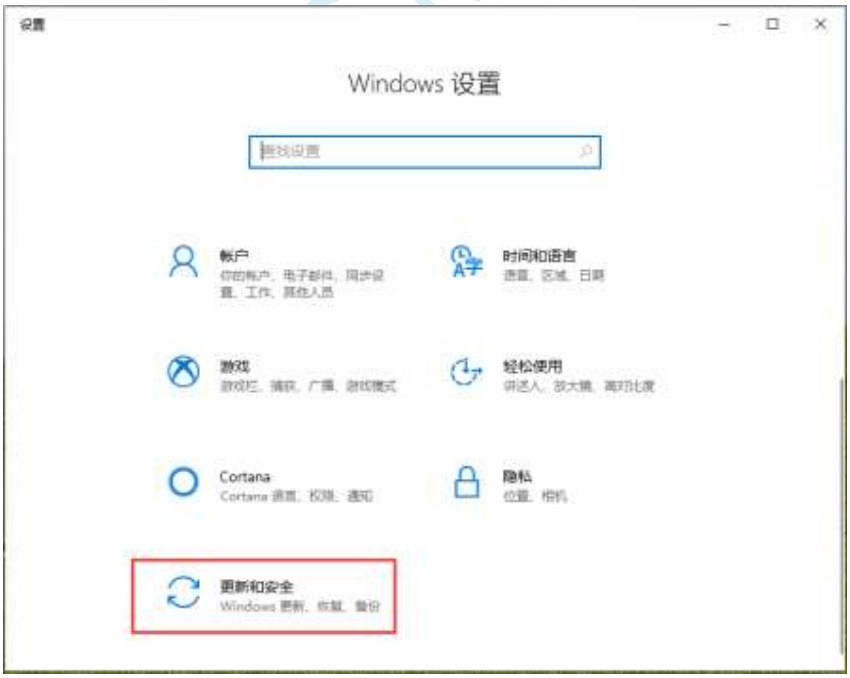
本菜单项是使用系统的 BCDEDIT 工具自动设置的 TestSigning 和 NoIntegrityChecks 两项参数以达到临时关闭驱动程序强制数字签名的目的。对于使能了安全启动的系统会无法使用 BCDEDIT 工具，需要在 BIOS 中将安全启动关闭（将 Secure Boot 设置为 Disable）

目前测试 WinXP/Win7-32/Win7-64 系统均可正常关闭数字签名，但 Win10 及后续系统无效。要在 Win10 及后续系统中关闭数字签名仍然需要从系统启动菜单中选择禁止强制数字签名来实现。详细步骤请参考下一小节

5.3.2 Windows 10 关闭驱动程序强制数字签名步骤

在“开始”菜单中选择“设置”功能





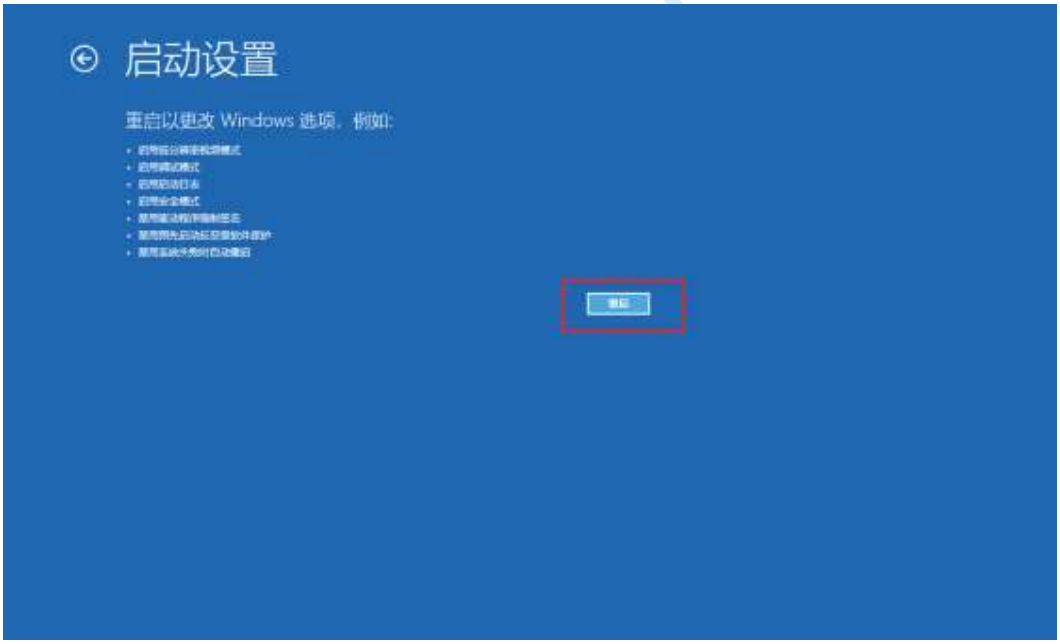

设置
主页
查找设置
更新和安全
Windows 更新
传递优化
Windows 安全中心
备份
疑难解答
恢复
激活
查找我的设备
开发者选项
恢复
重置此电脑
如果电脑未正常运行，重置电脑可能会解决问题。重置时，可以选择保留个人文件或删除个人文件，然后重新安装 Windows。
开始
高级启动
通过设备或磁盘(如 U 盘或 DVD)启动，更改你的电脑固件设置，更改 Windows 启动设置，或者从系统映像还原 Windows。 这将重新启动电脑。
立即重新启动
更多恢复选项
了解如何进行 Windows 的全新安装以便开始全新的体验
备份文件


选择一个选项
继续
关闭电脑
使用设备
疑难解答


疑难解答
重置此电脑
高级选项


高级选项
系统还原
启动修复
卸载更新
命令提示符
系统映像恢复
UEFI 固件设置
查看更多恢复选项


高级选项
启动设置
查看更多恢复选项


启动设置
重启以更改 Windows 选项，例如:

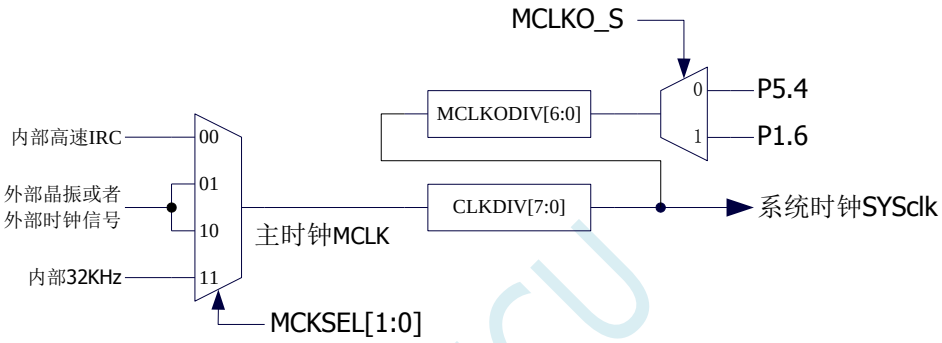
系统重新启动后，驱动程序的强制数字签名即可临时被关闭

6 时钟、复位、省电模式与系统电源管理

6.1 系统时钟控制

系统时钟控制器为单片机的 CPU 和所有外设系统提供时钟源，系统时钟有 3 个时钟源可供选择：内部高精度 IRC、内部 32KHz 的 IRC（误差较大）和外部晶振。用户可通过程序分别使能和关闭各个时钟源，以及内部提供时钟分频以达到降低功耗的目的。

单片机进入掉电模式后，时钟控制器将会关闭所有的时钟源



系统时钟结构图

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	时钟选择寄存器	FE00H	-						MCKSEL[1:0]		xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H									nnnn,nnnn
HIRCCR	内部高速振荡器控制寄存器	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	XCFILTER[1:0]		NMXCG	-	-	XOSCST	0000,0xx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	内部 IRC 起振去抖控制	FE06H	IRCDB[7:0]								1000,0000

6.1.1 系统时钟选择寄存器（CLKSEL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKSEL	FE00H	-						MCKSEL[1:0]	

MCKSEL[1:0]：主时钟源选择

MCKSEL[1:0]	主时钟源
00	内部高精度 IRC
01	外部晶体振荡器或外部输入时钟信号
10	-
11	内部 32KHz 低速 IRC

6.1.2 时钟分频寄存器（CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

CLKDIV：主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
x	MCLK/x
…	…
255	MCLK/255

注意：用户程序复位后，系统会自动根据上次 ISP 下载时所设定工作频率所需的分频系数来设置此寄存器的初始值

6.1.3 内部高速高精度 IRC 控制寄存器（HIRCCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HIRCCR	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST

ENHIRC：内部高速高精度 IRC 使能位

0：关闭内部高精度 IRC

1：使能内部高精度 IRC

HIRCST：内部高精度 IRC 频率稳定标志位。（只读位）

当内部的 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 HIRCST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 IRC 时，首先必须设置 ENHIRC=1 使能振荡器，然后一直查询振荡器稳定标志位 HIRCST，直到标志位变为 1 时，才可进行时钟源切换。

6.1.4 外部振荡器控制寄存器（XOSCCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	FE03H	ENXOSC	XITYPE	XCFILTER[1:0]		NMXCG	-	-	XOSCST

ENXOSC：外部晶体振荡器使能位

0：关闭外部晶体振荡器

1：使能外部晶体振荡器

XITYPE：外部时钟源类型

0：外部时钟源是外部时钟信号（或有源晶振）。信号源只需连接单片机的 XTALI（P1.7）**（此时 P1.6 口固定为高阻输入模式，可用于读取外部数字信号或当作 ADC 输入，但一般不建议使用，因为旁边的 P1.7 口有高频振荡信号会对 P1.6 的信号有影响）**

1：外部时钟源是晶体振荡器。信号源连接单片机的 XTALI（P1.7）和 XTALO（P1.6）

XOSCST：外部晶体振荡器频率稳定标志位。（只读位）

当外部晶体振荡器从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器

频率稳定后，时钟控制器会自动将 XOSCST 标志位置 1。所以当用户程序需要将时钟切换到使用外部晶体振荡器时，首先必须设置 ENXOSC=1 使能振荡器，然后一直查询振荡器稳定标志位 XOSCST，直到标志位变为 1 时，才可进行时钟源切换。

XCFILTER[1:0]：外部晶体振荡器抗干扰控制寄存器

00：外部晶体振荡器频率在 48M 及以下时可选择此项

01：外部晶体振荡器频率在 24M 及以下时可选择此项

1x：外部晶体振荡器频率在 12M 及以下时可选择此项

NMXCG：外部晶体振荡器增益控制

0：低增益

1：高增益

6.1.5 内部 32KHz 低速 IRC 控制寄存器（IRC32KCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC32KCR	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

ENIRC32K：内部 32K 低速 IRC 使能位

0：关闭内部 32K 低速 IRC

1：使能内部 32K 低速 IRC

IRC32KST：内部 32K 低速 IRC 频率稳定标志位。（只读位）

当内部 32K 低速 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 IRC32KST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 32K 低速 IRC 时，首先必须设置 ENIRC32K=1 使能振荡器，然后一直查询振荡器稳定标志位 IRC32KST，直到标志位变为 1 时，才可进行时钟源切换。

6.1.6 主时钟输出控制寄存器（MCLKOCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	FE05H	MCLKO_S	MCLKODIV[6:0]						

MCLKODIV[6:0]：主时钟输出分频系数

（**注意：主时钟分频输出的时钟源是经过 CLKDIV 分频后的系统时钟**）

MCLKODIV[6:0]	系统时钟分频输出频率
0000000	不输出时钟
0000001	SYSClk/1
0000010	SYSClk /2
0000011	SYSClk /3
...	...
1111110	SYSClk /126
1111111	SYSClk /127

MCLKO_S：系统时钟输出管脚选择

0：系统时钟分频输出到 P5.4 口

1：系统时钟分频输出到 P1.6 口

6.2 STC8A8K64D4 系列内部 IRC 频率调整

STC8A8K64D4 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时，ISP 下载软件会根据用户所选择/设置的频率自动进行调整，一般频率值可调整到±0.3%以下，调整后的频率在全温度范围内（－40℃～85℃）的温漂可达-1.35%～1.30%。

STC8A8K64D4 系列内部 IRC 有 4 个频段，频段的中心频率分别为 6MHz、10MHz、27MHz 和 44MHz，6M 频段的调节范围约为 4.5MHz～8MHz，10M 频段的调节范围约为 7.5MHz～13.5MHz，27M 频段的调节范围约为 20.5MHz～36MHz，44M 频段的调节范围约为 29MHz～55MHz（注意：不同的芯片以及不同的生成批次可能会有约 5%左右的制造误差）。建议用户在 ISP 下载时设置 IRC 频率不要高于 45MHz。

注意：对于一般用户，内部 IRC 频率的调整可以不用关心，因为频率调整工作在进行 ISP 下载时已经自动完成了。所以若用户不需要自行调整频率，那么下面相关的 4 个寄存器也不能随意修改，否则可能会导致工作频率变化。

若用户需要在自己的代码中动态选择芯片预置的频率，请参考预置频率列表以及“用户自定义内部 IRC 频率”的范例程序

内部 IRC 频率调整主要使用下面的 4 个寄存器进行调整

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IRCBAND	IRC 频段选择	9DH	-	-	-	-	-	-	SEL[1:0]		0000,00nn
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	-	LIRTRIM	0000,000n
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	时钟分频寄存器	FE01H	CLKDIV[7:0]								nnnn,nnnn

6.2.1 IRC 频段选择寄存器（IRCBAND）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	9DH	-	-	-	-	-	-	SEL[1:0]	

SEL[1:0]：频段选择

00：选择 6MHz 频段

01：选择 10MHz 频段

10：选择 27MHz 频段

11：选择 44MHz 频段

6.2.2 内部 IRC 频率调整寄存器（IRTRIM）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRTRIM	9FH	IRTRIM[7:0]							

IRTRIM[7:0]：内部高精度 IRC 频率调整寄存器

IRTRIM 可对 IRC 频率进行 256 个等级的调整，每个等级所调整的频率值在整体上呈线性分布，局部会有波动。宏观上，每一级所调整的频率约为 0.24%，即 IRTRIM 为（n+1）时的频率比 IRTRIM 为（n）时的频率约快 0.24%。但由于 IRC 频率调整并非每一级都是 0.24%（每一级所调整频率的最大值约为 0.55%，最小值约为 0.02%，整体平均值约为 0.24%），所以会造成局部波动。

6.2.3 内部 IRC 频率微调寄存器（LIRTRIM）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LIRTRIM	9EH	-	-	-	-	-	-	-	LIRTRIM

LIRTRIM：内部高精度 IRC 频率微调寄存器

6.2.4 时钟分频寄存器（CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

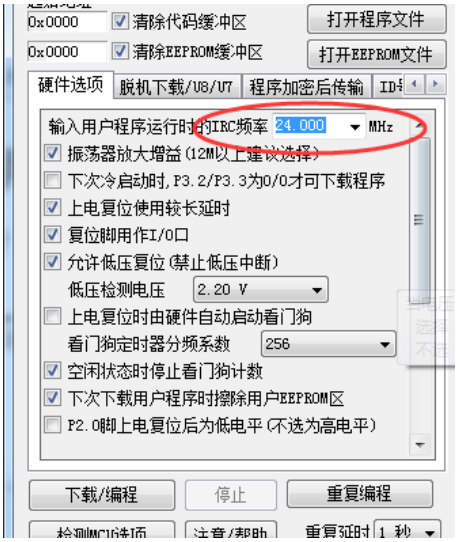
CLKDIV：主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
x	MCLK/x
…	…
255	MCLK/255

6.2.5 分频出 3MHz 用户工作频率，并用户动态改变频率追频示例

为得到 3MHz 的频率，可使用 24MHz÷8 的方法。

首先在进行 ISP 下载时选择内部 IRC 工作频率为 24MHz，如下图所示，



然后在代码中选择时钟源为内部 IRC，并使用 CLKDIV 寄存器进行 8 分频。

C 语言代码

```
//测试工作频率为24MHz

#include "reg51.h"
#include "intrins.h"

#define    CLKSEL      (*(unsigned char volatile xdata *)0xfe00)
#define    CLKDIV      (*(unsigned char volatile xdata *)0xfe01)
#define    HIRCCR      (*(unsigned char volatile xdata *)0xfe02)
#define    XOSCCR      (*(unsigned char volatile xdata *)0xfe03)
#define    IRC32KCR    (*(unsigned char volatile xdata *)0xfe04)

sfr        P_SW2       =   0xba;
sfr        IRTRIM      =   0x9f;

sfr        P0M1        =   0x93;
sfr        P0M0        =   0x94;
sfr        P1M1        =   0x91;
sfr        P1M0        =   0x92;
sfr        P2M1        =   0x95;
sfr        P2M0        =   0x96;
sfr        P3M1        =   0xb1;
sfr        P3M0        =   0xb2;
sfr        P4M1        =   0xb3;
sfr        P4M0        =   0xb4;
sfr        P5M1        =   0xc9;
sfr        P5M0        =   0xca;

void main()
{
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    CLKSEL = 0x00;                  //选择内部 IRC ( 默认 )
    CLKDIV = 0x08;                  //时钟 8 分频
    P_SW2 = 0x00;

    IRTRIM++;                       //IRC 频率向上 3‰进行微调（注意判断边界）
//  IRTRIM--;                       //IRC 频率向下 3‰进行微调（注意判断边界）

    while (1);
}
```

汇编代码

```
;测试工作频率为 24MHz

P_SW2       DATA    0BAH
IRTRIM      DATA    09FH

CLKSEL      EQU     0FE00H
CLKDIV      EQU     0FE01H
HIRCCR      EQU     0FE02H
XOSCCR      EQU     0FE03H
IRC32KCR    EQU     0FE04H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
```