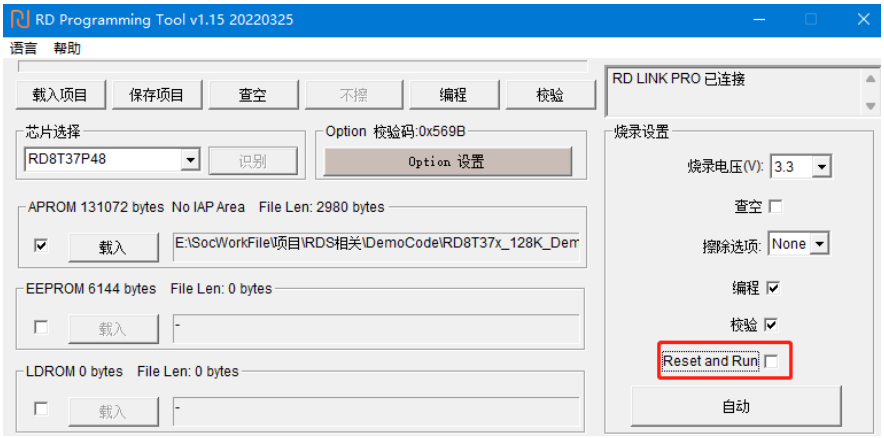
目录

1 新定义 RD8X3X 系列 MCU 电气参数注意项

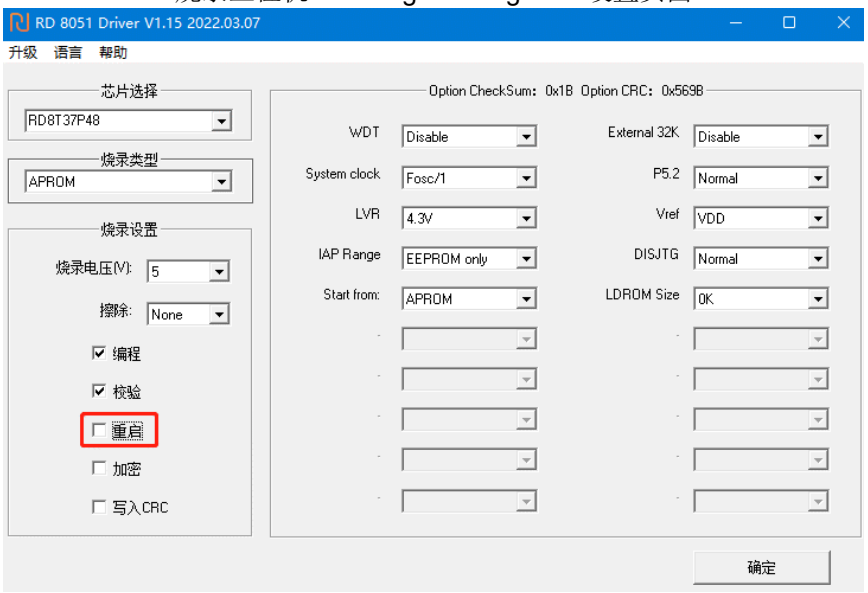
工作电压：2.0V~5.5V
工作温度：-40 ~ 105℃
内核：超高速 1T 8051；双 DPTR
Flash ROM：可重复写入>10 万次
LDROM：可通过 Code Option 设置项将 LDROM 设为 0/1/2/4k
EEPROM：独立的 6Kbyte，可重复写入 10 万次，100 年以上保存寿命
系统时钟：内建高频振荡器频率误差： 跨越 (2.0V~5.5V) 及 (-40 ~ 105℃) 应用环境，不超过 ±2%

2 新定义 RD8X3X 系列 MCU 烧写注意事项

1. 新定义 RD8X3X 系列芯片的 CLK 或 DIO 管脚对 GND 的电容不得超过 100pF，VDD 对 GND 的电容不可超过 1000uF。
2. 烧录引出点与芯片之前尽量不要串电阻，如无法避免，应保证串接电阻的阻值不超过 100R，且烧录时要尽量缩短烧录排线。
3. 电路设计时应避免将芯片的 CLK 和 DIO 连到同一个数码管上。
4. RD8X3X 系列芯片脱机烧录时，如果程序上电有 IAP 写的操作，则会校验失败，原因为 RD8X3X 系列脱机校验位 CRC 校验，程序跑起来导致 ROM 数据改变，从而校验失败，解决方法两种：
   （1） 在初始化之前加一段时间(建议 100ms)的延时才能烧录成功。
   （2） 在烧录上位机和 Keil 插件的配置页面中，取消勾选“Reset and Run”：



烧录上位机 RD Programming Tool 设置页面

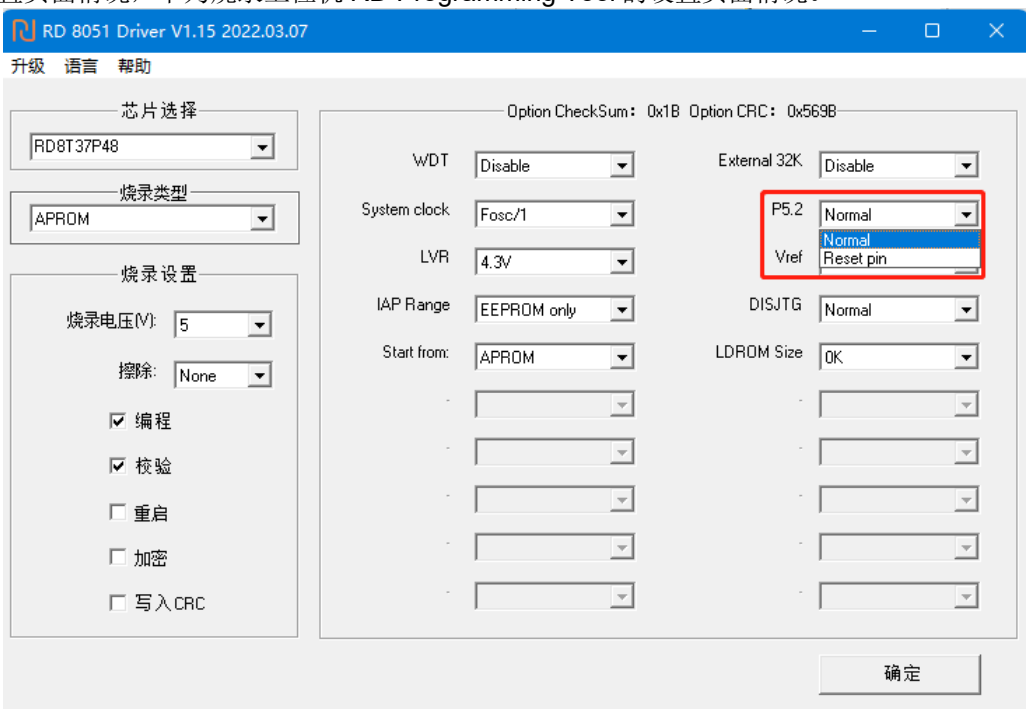


KEIL 插件的设置页面

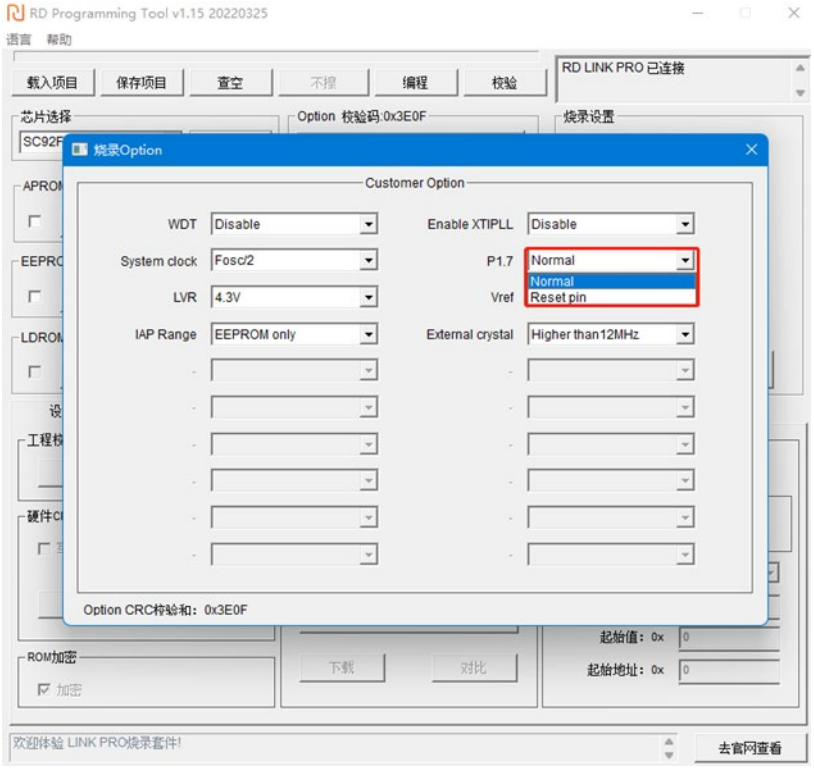
3 电路设计的注意事项

新定义 RD8X3X 系列 MCU 的 GPIO 上电默认模式为高阻输入模式。

新定义 RD8X3X 系列 MCU 的 RST 管脚，通过 Option 选项可设置为 Reset pin 或 GPIO。RST 管脚作为 Reset pin 时，低电平使能；作为 GPIO 时，管脚低电平不会产生复位。Option 的设置如图（以 RD8G37x 为例）。上为 KEIL 插件的设置页面情况，下为烧录上位机 RD Programming Tool 的设置页面情况。



KEIL 插件的设置页面情况



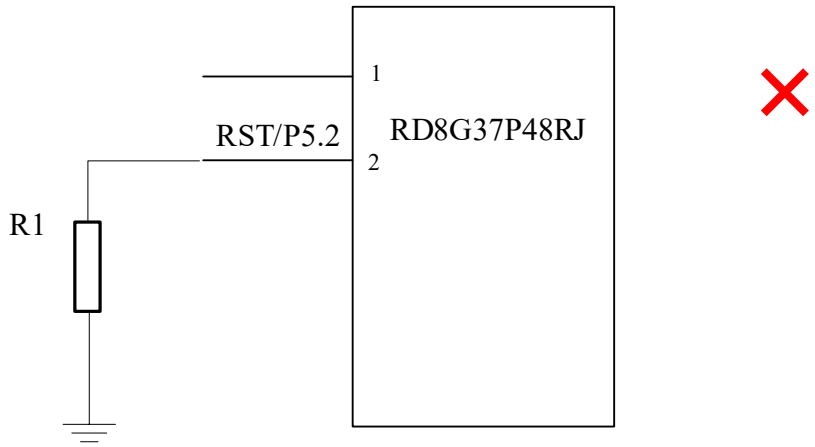
烧录上位机 RD Programming Tool 的设置页面情况

3.1 电路设计实例

3.1.1 RST 管脚电路

新定义 RD8X3X 系列 MCU 的 RST 引脚，与 I/O 复用，有别于传统 MCU 的 RST 引脚（传统的 RST 的引脚只能做输入，不能做输出），既可以做输入，又可以做输出。当将 IO 口设置为复位口时，上电后，用户电路的 RST 口不能一直为低，否则会一直复位，无法正常工作。因此，用户设计电路时，需要注意：

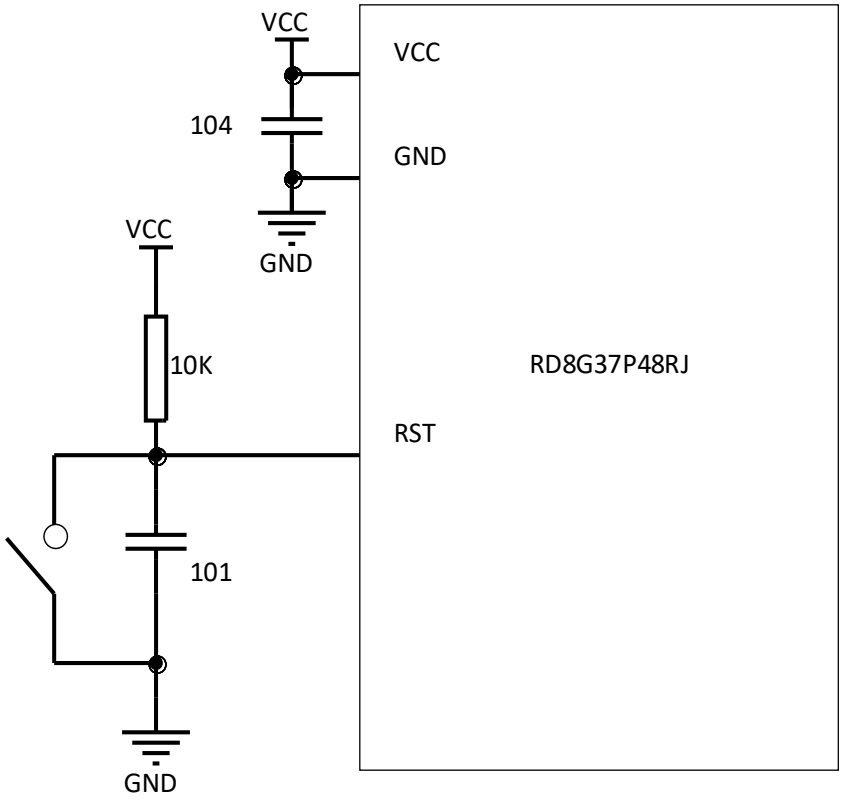
错误接法：



错误接法图示

说明：以上电路，若 RST 外接一个电阻 R1，系统在上电时与内部上拉判为低电平，造成系统一直复位，无法正常工作。

建议接法：

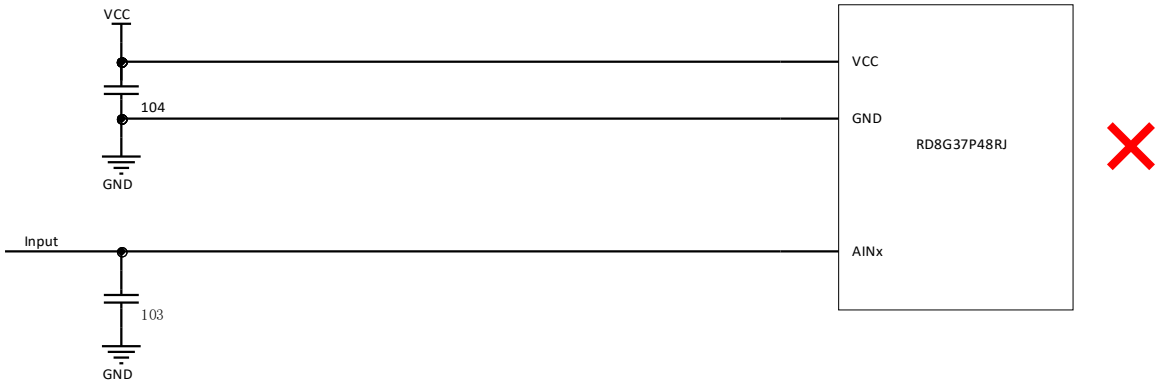


建议接法图示

3.1.2 ADC 采样管脚电路

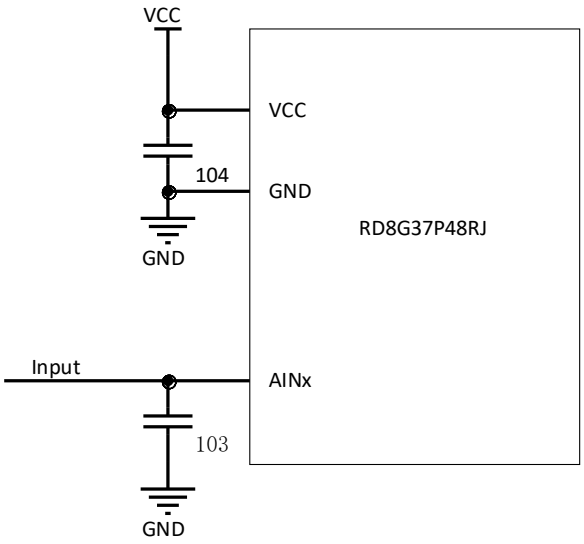
新定义 RD8X3X 系列 MCU 的 ADC 采样口需要在靠近管脚处加 103 电容， ADC 转换需要让电源电压稳定，所以在使用 ADC 功能时，请在靠近 IC 的 VCC 和 GND 处加 104 电容，以保证转换结果准确。

错误接法：104 电容离电源脚太远， 103 电容离 ADC 采样口太远。



错误接法图示

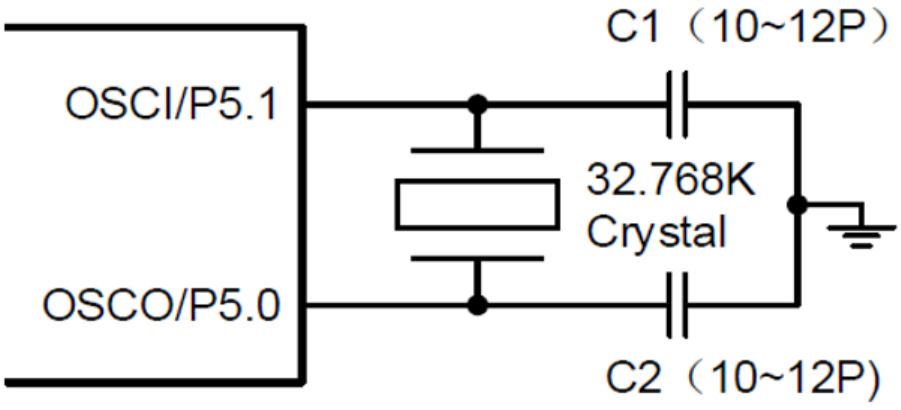
建议接法：104 电容靠近电源脚， 103 电容靠近 ADC 采样口。



建议接法图示

3.1.3 外部晶振电路

新定义 RD8X3X 系列 MCU 提供了低频外部接口，用户如需使用外部晶振，匹配电容请根据所选晶振的要求进行选择，请将晶振电路靠近芯片引脚处。如图：



32.768K 外部晶振连接图

3.1.4 TOUCHKEY 电路

新定义触控 MCU 的触控架构为高灵敏度触控模式。
高灵敏度触控模式外接的 CMOD 电容容值范围为 472~104，推荐使用 103 电容，电容材质无特殊要求。
CMOD 电容需要尽量靠近芯片管脚。

3.2 IO 口各模式设置注意事项

新定义 RD8X3X 系列 MCU 的 GPIO，有三种工作模式：

1. 带上拉输入模式
2. 高阻输入模式
3. 强推挽输出模式

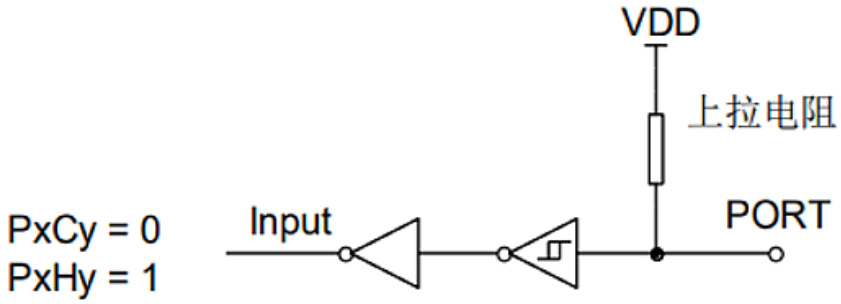
注意：未使用及封装未引出的 IO 口均要设置为强推挽输出模式。

3.2.1 I/O 设为高阻，实现电路设计

通常来说，对于某些特定场合的应用，譬如电压检测，过零检测，LCD 的应用等，都是采用高阻工作模式来实现的。因此说，用户可以从新定义 MCU 体系中按需选择。

3.2.2 带上拉输入模式

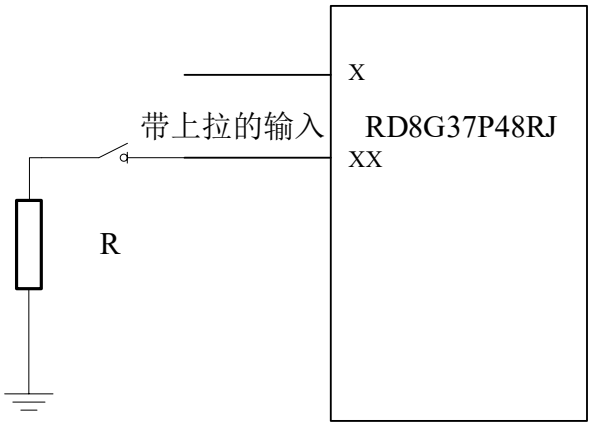
带上拉的输入模式下，输入口恒定接一个上拉电阻，仅当输入口上拉电平被拉低时，才会检测到低电平信号。带上拉的输入模式的端口结构示意图如下：



带上拉的输入模式

3.2.3 带上拉输入模式检测按键

I/O 口作按键输入时，串接的下拉电阻 R 需小于 10K。



上拉模式的按键输入接法图示

3.2.4 I/O 开漏输出模式的实现方式

新定义 RD8X3X 系列 MCU 没有开漏输出设置项，若用户想让 IO 口实现开漏输出功能，需要通过切换模式以达到开漏输出的效果，需要引脚输出低时切换为强推挽输出模式，需要引脚保持悬空状态时，则将 IO 口切换为高阻输入模式即可。

代码示例如下：

```
P0PH &= 0XFE;   //去除 P00 的上拉电阻
P00 = 0;            //将 P00 输出 0
P0CON &= 0xfe；//P0 设置为输入模式,等效为开漏输出的开漏状态
P0CON |= 0x01；//P0 设置为输出模式,等效为开漏输出低
```

3.2.5 I/O 与或操作注意事项

新定义 RD8X3X 的 CPU 执行指令速度快，在执行下一条指令时 GPIO 上的电平可能还没来得及完成改变，所以请不要对一个 GPIO 寄存器通过做连续的与或操作的方式来改变 IO 口的输出状态，如下：

```
P0|=0x04;
P0|=0x08;
```

以上操作请合并为一条实现，如下：

```
P0|=0x0C;
```

或者改为位操作，如下：

```
P02=1;
P03=1;
```

3.2.6 I/O READ IO 功能注意事项

新定义 RD8X3X 系列 MCU 具有 Read IO 功能，Read IO 功能是默认开启的，即使 GPIO 处于输出模式，读端口数据寄存器，也会以 IO 管脚上的电平为依据返回高低值；若把 Read IO 功能关闭则输出模式下读端口数据寄存器则不会以 IO 管脚上的电平为依据，读取的是寄存器的值。输入模式不受影响。

代码示例如下：

```
OPINX = 0X86;      //关闭 Read IO 功能
OPREG&=0XF7 ;
OPINX = 0X86;      //开启 Read IO 功能
OPREG|=0X08 ;
```

4 软件编写的注意事项

新定义 RD8X3X 系列 MCU 内含有丰富的外设，只要配置相应的寄存器即可对其实现操作，但一些操作需要按要求进行，用户编程过程中需要注意以下几点。

4.1 TIMER2/3/4 使用注意事项

新定义 RD8X3X 系列 MCU 内含寄存器地址共用的三个独立 16-bit Timer，分别为 Timer2/3/4。所有 Timer2 有 4 种工作模式； TIM3/4 具有 3 种模式。用户配置指针寄存器 TXINX 使 TimerX 寄存器组指向 Timer2/3/4，从而实现一组寄存器配置三个独立 Timer 的功能。

用户操作 TimerX 寄存器组前，必须先配置 TXINX 指向用户指定的 Timer2/3/4，否则对 TimerX 寄存器组的操作未必对 TimerX 生效！用户在同时使用 Timer2/3/4 时尤需注意此点。

当使用多个 Timer2/3/4 时，中断服务函数编写需要注意以下两点：

1. 在中断操作 TimerX 寄存器组时，在首行添加 TXINX=0X02/ TXINX=0X03/ TXINX=0X04 指向 Timer2/3/4，以配置 TXINX 寄存器，确保操作的正确性。
2. 当开启中断时，因为 TIM2/3/4 中断函数中都需要操作到 TXINX 寄存器，可能会出现低优先级中断被高优先级中断打断开后，进入高优先级中修改了 TXINX，重新进入低优先级中断函数时 TimerX 寄存器组就已改变。为了避免上述情况的发生，在中断服务函数配置 TXINX 寄存器前，需要通过一个变量把 TXINX 的值保存起来，等待相应 TXINX 寄存器组操作完成后，再把变量的值赋给 TXINX 寄存器。

Timer2/3/4 同时使用代码示例如下：

```
TXINX = 0X02;                          //将 TimerX 寄存器组指向 Timer2
                                       //设置 Timer2
TXMOD = TXMOD & 0X7F | (1<<7);         //设置 Timer2 时钟频率为 FSYS/1
TXCON = 0X00;                          //设置 Timer2 为 16 位自动重载定时器(向上计数)
THX = (65535-32000) / 256;             //1ms@32M
TLX = (65535-32000) % 256;
RCAPXH = (65535-32000) / 256;
RCAPXL = (65535-32000) % 256;
TRX = 1;                               //启动 Timer2

TXINX = 0X03;                          //将 TimerX 寄存器组指向 Timer3
                                       //设置 Timer3
TXMOD = TXMOD & 0X7F | (1<<7);         //设置 Timer3 时钟频率为 FSYS/1
THX = (65535-3200) / 256;              //100us@32M
TLX = (65535-3200) % 256;
RCAPXH = (65535-3200) / 256;
RCAPXL = (65535-3200) % 256;
TRX = 1;                               //启动 Timer3

TXINX = 0X04;                          //将 TimerX 寄存器组指向 Timer4
                                       //设置 Timer4
TXMOD = TXMOD & 0X7F | (1<<7);         //设置 Timer4 时钟频率为 FSYS/1
THX = (65535-32000) / 256;             //1ms@32M
TLX = (65535-32000) % 256;
RCAPXH = (65535-32000) / 256;
RCAPXL = (65535-32000) % 256;
TRX = 1;                               //启动 Timer4

void Timer2_ISR() interrupt 5          //Timer2 中断服务函数
{
    unsigned char TXINX_Stack = TXINX;
    TXINX = 0X02;                      //进入 Timer2 中断服务函数首先将 TimerX 寄存器组指向 Timer2
    ......
    TFX = 0;                           //清除 Timer2 中断标志位 TFX
    ......
     TXINX = TXINX_Stack;
```

```
    }
    void Timer3_ISR() interrupt 13                //Timer3 中断服务函数
    {
        unsigned char TXINX_Stack = TXINX;
        TXINX = 0X03;                             //进入 Timer3 中断服务函数首先将 TimerX 寄存器组指向 Timer3
        ......
        TFX = 0;                                  //清除 Timer3 中断标志位
        ......
        TXINX = TXINX_Stack;
    }
    void Timer4_ISR() interrupt 14                //Timer4 中断服务函数
    {
        unsigned char TXINX_Stack = TXINX;
        TXINX = 0X04;                             //进入 Timer4 中断服务函数首先将 TimerX 寄存器组指向 Timer4
        ......
        TFX = 0;                                  //清除 Timer4 中断标志位
        ......
        TXINX = TXINX_Stack;
}
```

4.2 常规脉冲宽度调制计数器 PWM2/3/4 使用注意事项

新定义 RD8X3X 系列 MCU 拥有 6 路常规 PWM，分为 3 组：PWM2、PWM3、PWM4。

注意：这三组 PWM 的周期寄存器分别与 Timer2，Timer3，Timer4 的 RCAPXL 和 RCAPXH 共用，因此一旦用户使用了 PWM2、PWM3、PWM4 资源，就不能再更改 Timer2，Timer3，Timer4 的定时/计数值，否则会导致 PWM 周期输出异常！

当 PWM2/3/4 输出波形时，若需改变占空比，可通过改变高电平设置寄存器 PDTxy（x=2~4，y=0~1）的值实现。但需要注意:更改 PDTxy 的值，占空比不会立即改变，而是等待本周期结束，在下个周期改变；若需改变占空比，可通过改变高电平设置寄存器 PDTxy（x=2~4，y=0~1）的值实现。但需要注意:更改 PDTxy 的值，占空比不会立即改变，而是等待本周期结束，在下个周期改变。

使用 PWM2/3/4 前请先查阅 TIMER2/3/4 使用注意事项。

4.3 PWM 设置及使用注意事项

新定义 RD8X3X 系列 MCU PWM 在互补模式下，PWM40/PWM41, PWM42/PWM43, PWM50/PWM51 和 PWM52/PWM53 分为四组，分别通过 PDT40[16:0]、PDT42[16:0]、PDT50[16:0]和 PDT52[16:0]调节占空比；此时寄存器 PDT41[16:0]、PDT43[16:0]、PDT51[16:0]和 PDT53[16:0]无效；但各 PWMxy（x=4~5，y=0~3）输出和反向依然由相应的 ENPxy 和 INVxy 独立控制。

新定义 RD8X3X 系列 MCU PWM 提供故障检测功能。用户使能故障检测功能后，不能悬空 FLT 管脚，否则 PWM 输出异常！

当故障发生时，PWM 停止输出，PWM 口处于高阻状态。故障检测模式分为立即模式和锁存模式。锁存模式下，故障信号满足失能条件后，硬件不会自动清除故障检测状态标志位，用户可通过 PWMFLT &= 0x7f 软件清零。

4.4 PCON 寄存器设置注意事项

新定义 RD8X3X 系列 MCU 提供了电源管理功能，可让芯片进入到省电模式，操作 PCON 寄存器的对应项即可，但在操作 PCON 寄存器后，请在其配置指令后接至少 8 个 NOP 指令，否则程序可能出错。

使用示例如下：

```
#include "intrins.H"
PCON |= 0X02;                    //进入 STOP 模式，后需接 8 个 NOP
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
```

请不要同时置起 STOP 位和 IDLE 位！

4.5 UART0 设置及使用注意事项

新定义 RD8 系列的 MCU RD8X3X 单片机在使用 UART0 时，如果选择 TIMER1 做波特率发生器，定时器 1 必须停止计数，即 TR1 = 0。用户在选择 Timer2 做波特率发生器时，需先配置 TXINX=0X02 使 TimerX 寄存器组指向 Timer2，再配置 TXCON 选择波特率发生器。使用示例如下：

```
SCON   = 0X50;     //设置通信方式为模式一，允许接收
TXINX = 0X02;      //将 TimerX 寄存器组指向 Timer2
TXCON &= 0XCF;     //选择 T2 做波特率发生器
TRX = 1;           //用定时器 2 作为波特率发生器，定时器 2 必须使能计数
RCAPXH= 0x0d;      //在 32M 时，波特率为 9600；定时器初值[RCAPXH,RCAPXL] = Fsys/波特率
RCAPXL= 0x05;      //在 32M 时，波特率为 9600
EUART = 1;         //开启 Uart0 中断
```

4.6 SPI/TWI/UART 三选一通用串行接口 USCI 设置及使用注意事项

新定义 RD8X3X 系列 MCU 集成了三选一串行接口电路 USCI，用户可通过 USMDx[1:0]配置 USCIx 为 SPI/TWI/UART 中任一种通信模式。

4.6.1 SPI 使用注意事项：

USCI 的 SPI 具备 8 位和 16 位传输模式；SPI 设置为 16 位模式时，必须先写入高字节 SPDH[7:0]，后写入低字节 SPDL[7:0]，低字节写入后立刻开始传送。注意：SPDH[7:0]仅用于 16 位模式。

为了避免 SPI 功能打开时误产生 CLK（此时管脚从 IO 切换到 SPI 功能），必须在打开 SPI 前根据 CLK 的空闲电平设置，先将 IO 口设置为对应的电平状态。如设置 CPOL = 0 空闲为低电平时需要将 CLK 对应的 IO 设置输出 0 再打开 SPI(SPEN = 1)，反之亦然，否则会导致通讯错位。同时，SPI 通讯时，程序中需要经常重置 SPI，以保证不会因为噪声等原因误接收 CLK 而导致一直通讯错误。

4.6.2 TWI 使用注意事项：

1. 用户可根据需要将 USCI 的 TWI 设置为主机或者从机，TWI 作从机时，通信速率最高 400kHz；
2. 使用 TWI 功能时，建议用户在中断服务函数中通过状态标志位 STATE[2:0]查询通信状态；
3. 由于 TWI 作主机时没有中断标志判断 STOP，用户需在发送 STOP 信号前后加短暂延时防止通信出错。
4. 为防止 TWI 通讯时上一次的数据未传输完成就发送 Start 信号所导致的通讯异常，请在发送 Start 之前先执以下代码：

```
US0CON0&=0xF7;          //先将 AA 置 0
Delay ();               //延时一段时间，延时时间长度要大于 1byte 数据的传输时间
US0CON0 &=0x7f;         //关闭 TWI
US0CON0 |= 0x80;        //打开 TWI
```

4.6.3 UART 使用注意事项：

由于 US0CON0、US1CON0 和 USxCON0 寄存器不能进行位操作，为了防止全双工通信过程中对 RI 和 TI 进行清零时两者相互影响，USCI 的 UART 发送中断标志 TI 和接收中断标志 RI 的清除方式为“写 1 清零”，这样可以更好的实现全双工通信，而 RI 和 TI 的清除不会相互影响。

以 USCI0 为例，当 UART 工作于全双工的状态下，可能会出现发送 TI（发送中断标志位）和 RI（接收中断标志位）同时置起的情况，即 US0CON0 = XXXX XX11（X 代表其他配置值）。在发送中断标志位清零时，需要注意给 RI 赋值为 0，防止清除 TI 时误清除 RI，即 US0CON0 = XXXX XX10（X 代表其他配置值），以防 RI 被清零，导致中断丢失的风险。接收标志位清零同理。

错误写法如下：

```
US0CON0 |= 0X02;          //TI 清零，但 RI 置起时也会被误清，导致中断丢失的风险
US0CON0 |= 0X01;          //RI 清零，但 TI 置起时也会被误清，导致中断丢失的风险
```

正确写法如下，可确保 TI 和 RI 的清除不会相互影响：

```
US0CON0 &= 0XFE;          // TI 清零，并确保 RI 赋值为 0
US0CON0 &= 0xFD;          // RI 清零，并确保 RI 赋值为 0
```

4.7 USCI2/3/4/5 配置注意事项

新定义 RD8X0X 系列 MCU 拥有 USCI2/3/4/5，具备的功能与 USCI1 一致。USIC 的模式配置需要先配置 USINX，使 USMDX[1:0] 指向用户指定的 USCI2/3/4/5。通过配置 USMDX[1:0]把指定的 USCI2/3/4/5 配置为 SPI、TWI 和 UART 中任意一种通信模式。。

值得注意的是：USCI2/3/4/5 的控制寄存器共用同一组地址，用户可以通过 USINX[2:0]将 USCIX 寄存器组(USXCON0~3)指向 USCI2/3/4,从而实现一组寄存器配置四个独立 USCI 接口的有效操作。只有 USCIX[2:0]配置成功后，USCIX 寄存器组才会指向用户指定的 USCI2/3/4/5,此时操作 USCIX 寄存器组才是对应 USCI 接口的有效操作，否则对 USCIX 寄存器组的操作未必对 USCIX 生效！用户在同时使用 USCI2/3/4/5 时尤需注意此点。

当使用 USCI2/3/4 中断时，中断服务函数编写需要注意以下两点：

1. 在中断操作 USCI 寄存器，请先配置 USINX 寄存器，确保 USCIX 寄存器操作的正确性。
2. 当开启中断时，因为中断需要操作到 USINX 寄存器，可能会出现低优先级中断被高优先级中断打断开，进入高优先级中断时修改了 USINX，这个低优先级中断操作 USCIX 组就已改变。忘了避免上述情况的发生，中断服务函数配置 USINX 寄存器前，需要通过一个变量把 USINX 的值保存起来，等待相应 USCIX 寄存器组操作完成 后，再把变量的值赋给 USINX 寄存器。

USCI2/3/4 中断服务示例如下：

```
void USCI2_ISR() interrupt 16        // USCI2 中断服务函数
{
unsigned char USINX_Stack = USINX;
USINX = 0X02;                        //进入 USCI2 中断服务函数首先将 USCIX 寄存器组指向 USCI2
......
......
USINX = USINX_Stack;
}
void USCI3_ISR() interrupt 17        // USCI3 中断服务函数
{
unsigned char USINX_Stack = USINX;
USINX = 0X03;                        //进入 USCI3 中断服务函数首先将 USCIX 寄存器组指向 USCI3
......
......
USINX = USINX_Stack;
}
void USCI4_ISR() interrupt 18        // USCI4 中断服务函数
{
unsigned char USINX_Stack = USINX;
USINX = 0X04;                        //进入 USCI4 中断服务函数首先将 USCIX 寄存器组指向 USCI4
......
......
USINX = USINX_Stack;
}
```

```
void USCI5_ISR() interrupt 19          // USCI5 中断服务函数

{
unsigned char USINX_Stack = USINX;
USINX = 0X05;                          //进入 USCI5 中断服务函数首先将 USCIX 寄存器组指向 USCI5
......
......
USINX = USINX_Stack;
}
```

4.8 多通道切换采集注意事项

新定义 RD8X3X 系列 MCU 大多数型号拥有多路 ADC 通道，但每次转换只能转换一个通道，若想实现多路通道的 ADC 信号的采集，需要在一路 ADC 通道转换完毕后将转换口切换至另一路 ADC，如此反复以实现多通道的 ADC 转换。若在 ADC 通道切换后马上进行 AD 转换，通道口线上的电压可能存在不稳定的现象，在切换通道后转换的第一个值可能会存在异常，建议用户对某个通道做连续的多次采集和转换后，将切换通道后转换的第一个值或几个值去除或将最大值及最小值去除，再将剩余的 AD 转换值求平均值得到采集结果。

使用示例如下：

```
unsigned int ADC_Value0, ADC_Value1, ADC_Value2;
unsigned int ADC_Convert(void)
{
 unsigned  int Tad = 0, MinAd = 0x0fff, MaxAd = 0x0000, TempAdd = 0;
 unsigned char t = 0;
 for(t = 0; t < 10; t++)
 {
  ADCCON |= 0X40;                                    //开始 ADC 转换
  while(!(ADCCON & 0x20));                           //等待 ADC 转换完成
  ADCCON &= ~(0X20);                                 //清中断标志位
  Tad = ((unsigned int)ADCVH << 4) + (ADCVL >> 4);   //取得一次转换值
  if(Tad > MaxAd)
  {
   MaxAd = Tad;                                      //获得当前的最大值
  }
  if(Tad < MinAd)
  {
   MinAd = Tad;                                      //获得当前的最小值
  }
  TempAdd += Tad;                                    //转换值累加
 }
 TempAdd -= MinAd;                                   //去掉最小值
 TempAdd -= MaxAd;                                   //去掉最大值
 TempAdd >>= 3;                                      //求平均值
 return(TempAdd);
}

void ADC_channel(unsigned char channel)
{
 ADCCON = ADCCON & 0xE0 | channel;    //ADC 输入选择为 ADCchannel 口
}

void ADC_Multichannel()
{
 ADCCFG0 = 0x07;                 //设置 AIN0、AIN1、AIN2 设置为 ADC 口，并自动将上拉电阻移除。
 ADCCFG2 = 0x10;                 //ADCCFG 配置出错会影响 ADC 采集精度
 ADCCON  |= 0X80;                //开启 ADC 模块电源
 ADC_channel(0);                 //ADC 入口切换至 AIN0 口
 ADC_Value0 = ADC_Convert();     //启动 ADC 转换，获得转换值
 ADC_channel(1);                 //ADC 入口切换至 AIN1 口
```

```
    ADC_Value1 = ADC_Convert();     //启动 ADC 转换，获得转换值
    ADC_channel(2);                 //ADC 入口切换至 AIN2 口
    ADC_Value2 = ADC_Convert();     //启动 ADC 转换，获得转换值
}
```

4.9 外部中断设置注意事项

新定义 RD8X3X 系列 MCU 在使用外部中断功能时，请将对应的 IO 口设置为输入模式！IO 口需要先设置，再设置相应的外部中断配置。反过来操作有可能会误产生一次边沿中断。

同组外部中断共用一个中断向量，用户需要在中断服务函数内读取 IO 口电平，判断中断的来源，再执行对应的操作。不建议将多个双边沿中断设置在同一组外部中断内。

使用示例如下：

```
P4CON &= 0XFC;  //将 INT10（P40）口和 INT11(P41)设置为输入模式
P4PH |= 0X03;   //打开 P40 和 P41 的上拉电阻
INT1F = 0X03;   //使能 INT10、INT11 下降沿触发
EINT1 = 1;      //使能外部中断 1
EA = 1;         //开总中断
void Interrupt_work() interrupt 2
{
if(P40==0)      //判断外部中断是否来自于 INT10
    {
                //执行代码
    }
    if(P41==0)  //判断外部中断是否来自于 INT11
    {
                //执行代码
    }
}
```

4.10 软件操作 CODE OPTION 的注意事项

新定义 RD8X3X 系列的 MCU 内部有单独的一块 Flash 区域用于保存客户的上电初始值设置，此区域称为 Code Option 区域。用户在烧写 IC 时将此部分代码写入 IC 内部，IC 在复位初始化时，就会将此设置调入 SFR 作为初始设置。

Option 相关 SFR 的读写操作由 OPINX 和 OPREG 两个寄存器进行控制，各 Option SFR 的具体位置由 OPINX 确定，各 Option SFR 的写入值由 OPREG 确定：

符号	地址	说明		上电初始值
OPINX	FEH	**Option 指针**	OPINX[7:0]	00000000b
OPREG	FFH	**Option 寄存器**	OPREG[7:0]	nnnnnnnnb

操作 Option 相关 SFR 时 OPINX 寄存器存放相关 OPTION 寄存器的地址，OPREG 寄存器存放对应的值。

例如：要将 OP_HRCR 配置为 0x01，具体操作方法如下：

C 语言例程：

```
EA = 0;                 //关总中断
OPINX = 83H;            //将 OP_HRCR 的地址写入 OPINX 寄存器
OPREG = 0x01;           //对 OPREG 寄存器写入 0x01（待写入 OP_HRCR 寄存器的值）
EA=1;                   //开总中断
```

汇编例程：

```
CPL EA;
MOV OPINX,#83H;         //将 OP_HRCR 的地址写入 OPINX 寄存器
MOV OPREG,#01H;         //对 OPREG 寄存器写入 0x01（待写入 OP_HRCR 寄存器的值）
```

注意：禁止向 OPINX 寄存器写入 Customer Option 区域 SFR 地址之外的数值！否则会造成系统运行异常！

```
SETB EA;
```

4.11 TOUCHKEY 设置注意事项

应用程序初始化时需要将 TK 对应的 IO 口设置为强推挽输出模式，输出高电平。在 TK 扫描过程中，不能去操作使用 TK 对应的 IO。另外，如果同一块 PCB 上有两颗以上的触摸芯片，为了防止同频干扰，建议开启“抗干扰设置”。

特别说明：例如 RD8X3X 的 TK9/TK11 与 TK 调试通信口（DIO/CLK）复用，若需使用 TK 调试功能，请尽量避免使用 TK9/TK11！

更多 TouchKey 的注意事项请参考《新定义 RD8 系列 TouchKey MCU 应用指南》。

4.12 CRC 使用注意事项

1. CRCDRn 写入数据和读出不是同一数据；
2. 硬件计算所得的 CRC 值是整个程序区数据（注意，这里不包括 IAP 区域！）的 32 位 CRC 校验值。若地址单元中有用户上次操作后的残留值，会导致 CRC 值与理论值不符。因此，建议用户对整片 Flash ROM 进行擦除后再烧录代码以保证 CRC 值与理论值一致；
4. CRC 启动操作语句之后务必要加上至少 8 个 NOP 指令，确保 CRC 计算完成；
5. 执行 CRC 运算时需要关闭总中断 EA，在 8 个 NOP 之后才重新打开总中断；
6. 新定义 RD8X3X 系列的 IC 内建的 CRC 模块可用来实时生成程序代码的 32 位 CRC 值，该值和理论值比较，可监测程序区的内容是否正确。CRC 理论值不需要用户计算，烧录软件会根据载入的代码及 Code 区域设置项自动完成计算并在烧录时通过烧写器将 4 bytes 的 CRC32 计算结果写入 CRC 结果存储区，**硬件 CRC 的操作需配合 RD Programming Tool、RD LINK PRO。**

硬件 CRC 程序判断及烧录操作方法如下：

1. 用户在代码中执行硬件 CRC 计算，计算结果与 CRC 结果存储区读取的内容进行比较，若比较结果一致，则说明程序代码未改变：

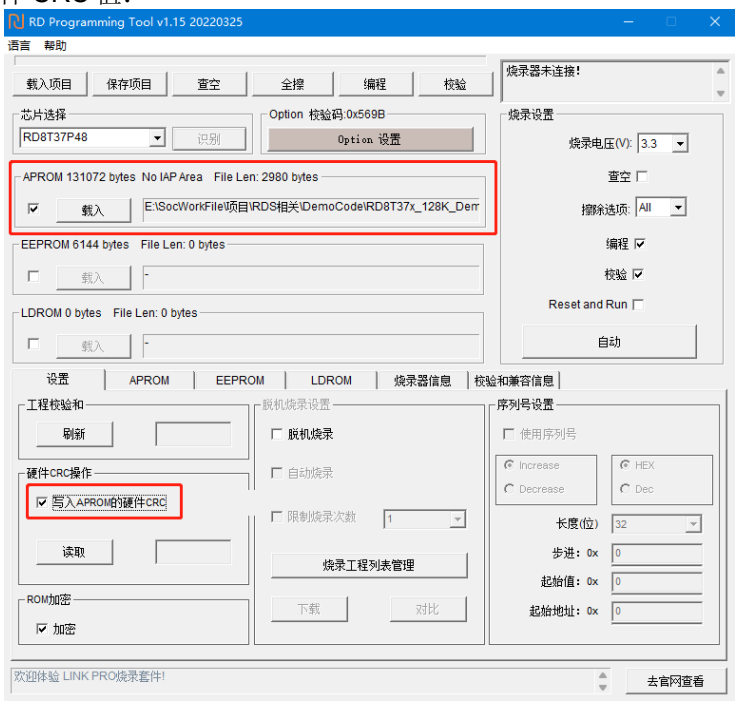
```
//执行硬件 CRC 并将计算结果与 CRC 结果存储区读取的内容进行比较
//返回值：
//0：未写入 CRC 或比较结果不一致或 CRC_SUM 校验不一致
//1：结果一致

#define CRC_Exist_Address       0x14
#define CRC_Exist_Flag          0x55
#define CRC_Result_Address      0x10
#define CRC_SUM_Address         0x15

//比较当前存储的 CRC32 值与硬件计算的 CRC32 是否相等
//返回 1:相等;        返回 0：不相等或者存储值无效
unsigned char Crc32_Check(void)
{
    unsigned long IapReadCrc32=0;
    unsigned char flag=0;
    if(IAP_Read_Crc_32bit(&IapReadCrc32)==0x01)                    // 读取保存正确的 CRC 值
    {
            if(IapReadCrc32==CRC32_Cal())
            {
                    flag=1;
            }
    }
    return flag;
}
//读取 Crc 存储区间 Crc32 值
//返回 1:存储值有效，值保存在变量*Crc32 中；
//返回 0:存储值无效。
unsigned char IAP_Read_Crc_32bit(unsigned long * Crc32)
{
```

```
    unsigned char i = 0,flag=0;
    unsigned int Crc_Cs=0;
    *Crc32=0;
    if( IAP_Read_Crc_8bit(CRC_Exist_Address) == CRC_Exist_Flag)  //判断 CRC 写入标志
    {
        for(i=0;i<5;i++)
        {
            Crc_Cs += IAP_Read_Crc_8bit(CRC_Result_Address+i);
        }
        if(Crc_Cs == (IAP_Read_Crc_8bit(CRC_SUM_Address)+(unsigned
int)((IAP_Read_Crc_8bit(CRC_SUM_Address+1))<<8)))  //校验保存的 CRC 数据是否有效
        {
            * Crc32 =IAP_Read_Crc_8bit(CRC_Result_Address);
            * Crc32+=(unsigned long)(IAP_Read_Crc_8bit(CRC_Result_Address+1))<<8;
            * Crc32+=(unsigned long)IAP_Read_Crc_8bit(CRC_Result_Address+2)<<16;
            * Crc32+=(unsigned long)IAP_Read_Crc_8bit(CRC_Result_Address+3)<<24;
            return   0x01;
        }
    }
    return flag;
}
//按照 8bit 读取 Crc 存储区间 Crc32 值
//返回 8bit 值
unsigned char IAP_Read_Crc_8bit(unsigned int OP_Address)
{
    unsigned char GetData=0;
    bit EA_Temp = EA;
    EA = 0;
    IAPADE = 0x01;          //操作 CRC 存储结果区域
    GetData = *((unsigned char code *)OP_Address);
    IAPADE = 0x00;                  //返回 ROM 区域
    EA = EA_Temp;
    return GetData;
}
//启动硬件 CRC 读取 APROM 的 CRC32 值
//返回 32bit 值
unsigned long  CRC32_Cal(void)
{
    unsigned long  Crc32_Cal_Result=0;
    bit EA_Temp = EA;
    EA = 0;
    OPERCON |= 0x01; //启动硬件 CRC;
    _nop_();                    //至少 8 个 NOP 操作
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    CRCINX = 0x00;
    Crc32_Cal_Result = CRCREG;
    Crc32_Cal_Result +=(unsigned long)CRCREG<<8;
    Crc32_Cal_Result +=(unsigned long)CRCREG<<16;
    Crc32_Cal_Result +=(unsigned long)CRCREG<<24;
    EA = EA_Temp;
    return Crc32_Cal_Result;
}
```

2. 连接好 IC 及烧录系统，烧录上位机载入代码，勾选上“APROM”编程区域（因为硬件 CRC 计算是只针对 APROM 区域），在烧界面勾选“写入 APROM 的硬件 CRC”，触发编程操作，即可通过烧写器向芯片 Custom Option 区域写入硬件 CRC 值：



4.13 软件安全加密功能注意事项

新定义 RD8X3X 系列芯片允许用户是否开启安全加密功能，该功能需要配合 RD LINK PRO 以及 RD Programming Tool 上位机软件使用，具体的操作说明见《新定义 LINK 系列量产开发工具使用手册》安全加密章节，可到新定义网站下载。

4.14 中断关闭注意事项

在实际应用中开启子中断后一般不需要再次关闭子中断，如果用户需要关闭子中断，必须在关闭子中断前先将总中断关闭，然后再关闭子中断，如下是子中断关闭步骤，请用户务必按照此步骤进行子中断的关闭

EA = 0;

子中断开关=0;(如 ET0 = 0;)

EA = 1;

4.15 提高 LCD 帧频注意事项

新定义 RD8X3X 系列 MCU 支持使用软件 LCD 库修改 LCD 帧频。当用户所使用 LCD 屏的驱动帧频不是 64Hz 时，请参考《新定义软件可调 LCD 帧频应用指南》。

4.16 128K 单片机分 Bank 应用注意事项

标准的 8051 单片机能寻址 64KB 的代码空间，而 RT8x37 拥有 128K Flash ROM，需要采用 Code Bank 的方式来扩展程序空间，具体请参考《新定义 128K 单片机分 Bank 应用说明》

5 新定义 RD8X3X 系列 MCU 的 IAP 及算法解析

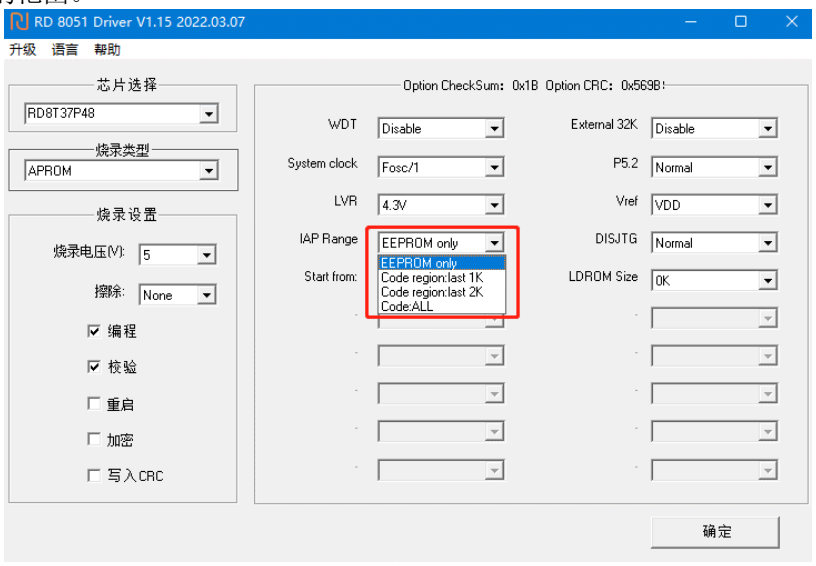
5.1 IAP 操作

以 RD8X3X 为例，说明新定义 RD8X3X 系列 MCU CODE 区 IAP 操作的方法。

RD8X3X 内部 64/128 Kbytes Flash 均可以进行 In Application Programming（IAP）操作，即允许用户程序动态的把数据写入内部的 Flash。

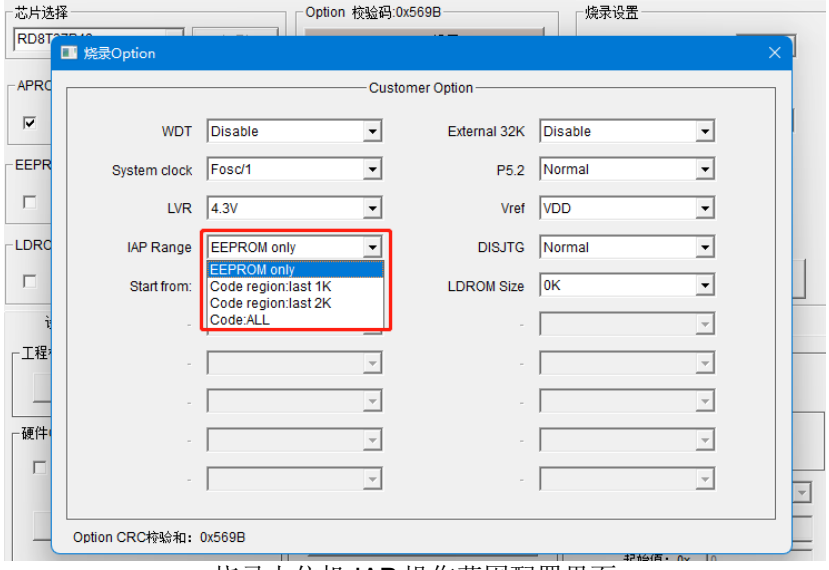
注：新定义 RD8X3X 系列 MCU 执行 IAP 的擦写操作，请参照《IAP 库使用指南》。

用户使用 IAP 时, 需要在 Option 项设置允许 IAP 操作的范围，在 Keil 中设置方法如下，在 IAP Range 的选项中选择允许 IAP 操作的范围。



KEIL IAP 操作范围配置界面

在上位机烧录时也是在 Option 项中的 IAP Range 的选项中选择允许 IAP 操作的范围。如图。



烧录上位机 IAP 操作范围配置界面

FLASH 读写特点：

1. 可进行单 Byte 读写操作；
2. Flash ROM 分为 128 个扇区（0000h~FFFFh），写前需对操作地址所属扇区进行扇区擦除（扇区擦除：512bytes），擦除写入该扇区首地址即可；

FLASH 的寿命：10 万次以上

5.2 IAP 的使用建议及注意事项

1 ROM 区域内的 IAP 操作有一定的风险，需要用户在软件中做相应的安全处理措施，如果操作不当可能会造成用户程序被改写！除非用户必需此功能(比如用于远程程序更新等)，否则不建议用户使用。

2 新定义 RD8X3X 系列 MCU 写入数据前需要对目标地址所在扇区进行扇区擦除，建议用户在擦除前将数据备份到另一个扇区，防止擦除过程中掉电而发生旧数据被擦除而新数据未写入的意外情况。

3 RD8X3X 系列芯片可以进行 In Application Programming（IAP）操作，而实际产品的应用中，只是需要把仅几个 Bytes 的数据写到 Flash，采用固定地址写入数据会使某些地址过早达到 IAP 寿命次数，同时每次写数据前需进行数据备份和扇区擦除，因此建议用户可采用分扇区、循环地址写入的方法操作 Flash。

4 RD8X3X 芯片越界 IAP 写或擦除操作 LDROM 区域会导致芯片损坏，请勿越界操作 LDROM！

5 当使用到函数指针等使用到 MOVC 操作时需要注意 ROMBANK[1:0]的值，否则可能会读值错误或者跑飞。

6 仿真注意事项

6.1 仿真状态下软复位失效

新定义 RD8X3X 系列 MCU 在仿真状态下执行会把软复位功能关闭，所以执行 PCON|=0x08；命令不会执行复位动作。

6.2 仿真状态下复位 SFR 不复位

新定义 RD8X3X 系列 MCU 在仿真状态下复位不会复位 SFR。

规格更改记录

版本	记录	日期
V1.0	初版	2022 年 6 月

声明

合肥市新定义电子有限公司（以下简称 RD）保留随时对 RD 产品、文档或服务进行变更、更正、增强、修改和改进的权利，恕不另行通知。RD 认为提供的信息是准确可信的。本文档信息于 2022 年 6 月开始使用。在实际进行生产设计时，请参阅各产品最新的数据手册等相关资料。