从 STM32F103C8T6 移植到 CH32F203C8T6

V1.0

说明

资源分配对比如下表：

表 1 资源分配对比

	STM32F103C8T6	CH32F203C8T6
Flash	64K	64K
RAM	20K	20K
主频	72MHz	144MHz
TIMER	4	4
U(S)ART	3	3
I2C	2	2
SPI	2	2
CAN	1	1
USBD	1	1
USBHD	——	1
ADC	10	10
TKey	-	10
OPA	——	2

目录

1 硬件差异

LQFP48 管脚兼容

2 内部资源差异对比

表 2 内部资源差异

	CH32F203C8T6	STM32F103C8T6
最大主频	144MHz	72MHz
USB	USBHD+USBD	USBD
OPA	2 组	无
TKey	10 组	无
CAN 和 USBD	可同时使用	不可同时使用
JTAG	不支持	支持
Standby 唤醒	支持所有 IO	仅支持 PA0

3 软件环境设置-Keil

目前市面通用的MDK for ARM版本有Keil5；建议客户使用Keil 5 V5.25及以上版本，ARM-CMSIS 5.3.0，以便支持WinUSB DAP-Link。若使用低版本Keil5，只支持HID DAPlink，下载速度相对较慢。

注：WCH-Link V2.4版本后改为 WinUSB 模式。

3.1 keil5 中添加 PACK 包

1. 从沁恒官网（http://www.wch.cn/）下载 CH32F20XEVT，打开 PUB 文件夹中的 Keil.WCH32F2xx_DFP.1.0.1 文件，如图 1 所示文件位置。

图 1 PACK 文件包位置

2. 双击解压安装至Keil 5的目录，一般都会默认选择，如若同时安装了Keil 4和Keil 5才需要手动选择。PACK包安装界面如图2所示，点击Next完成安装即可。

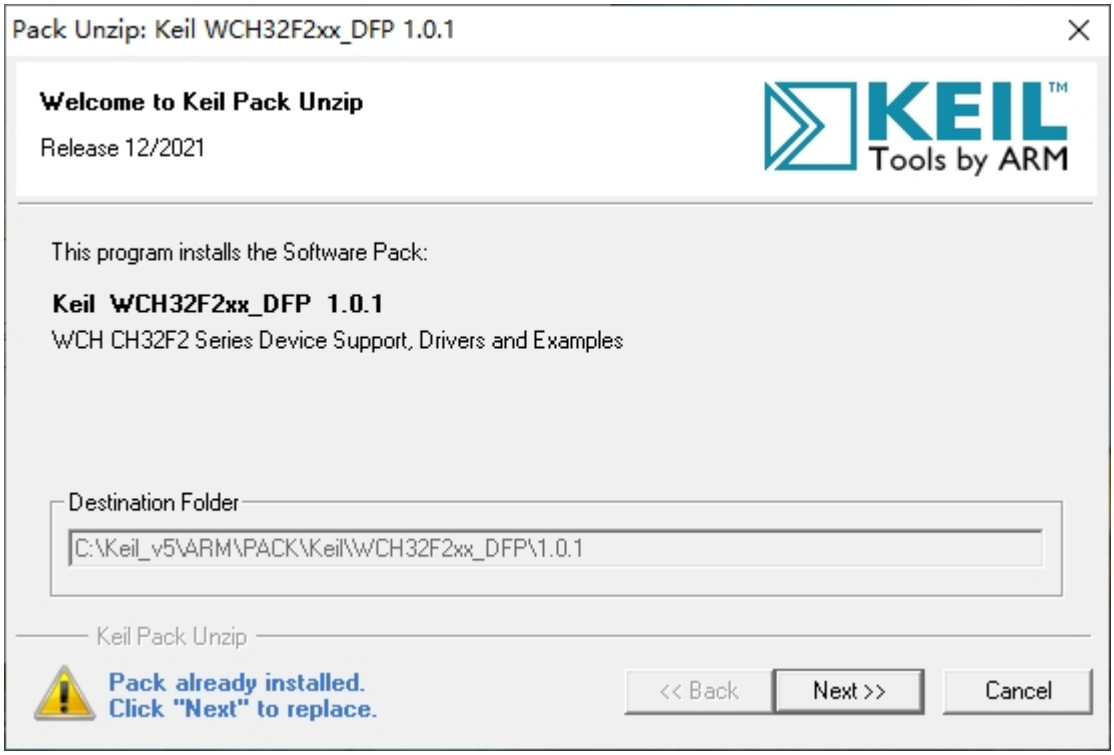


图 2 PACK 包安装界面

3. 安装成功后，重新打开Keil 5，则可以在File->Device Database中出现WCH的下拉选项，点击可以查看到相应的型号。如图3所示Pack包成功安装。

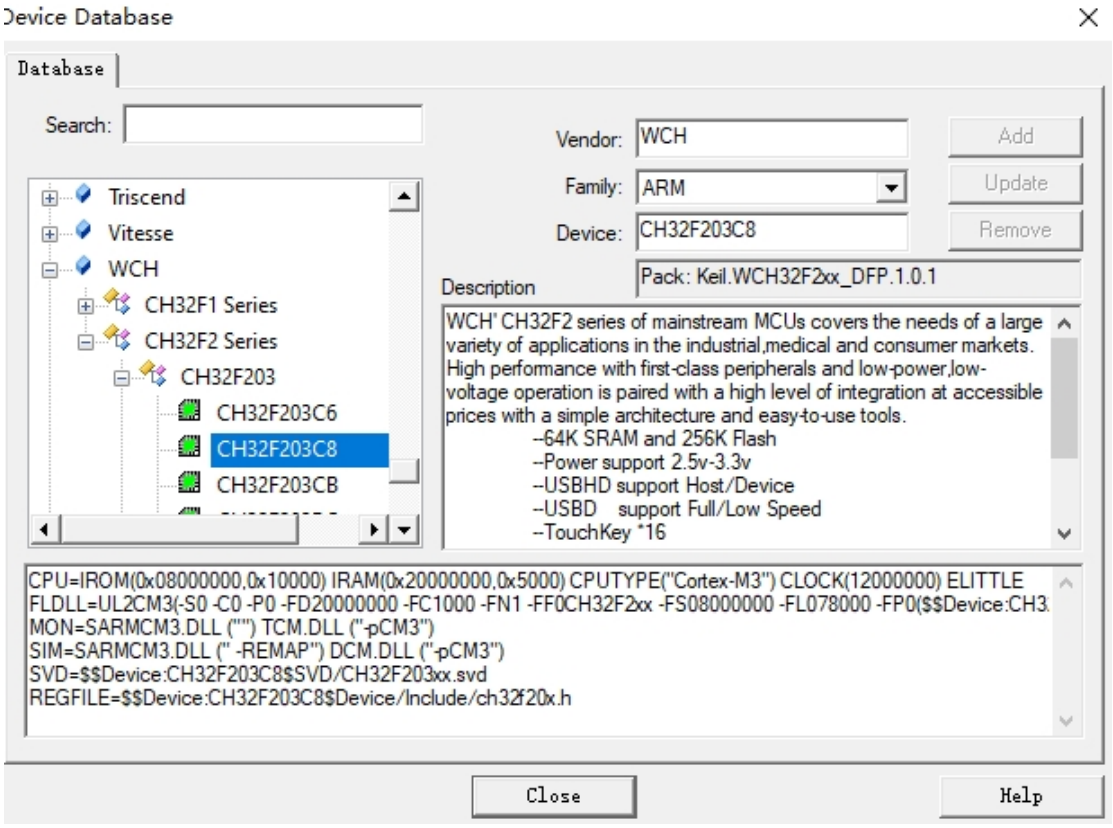


图 3 Pack 包成功安装示意图

4. 为了后续debug工作的顺利进行，建议检查一下安装路径下是否有下载算法，可以通过如下方式查看：打开一个工程，将型号选为CH32F2xx的型号，然后Options for Target -> Debug ->Settings -> Flash Download-> Add，如果下拉选项中有CH32F2xx的下载算法则完全安装成功，如图4所示。

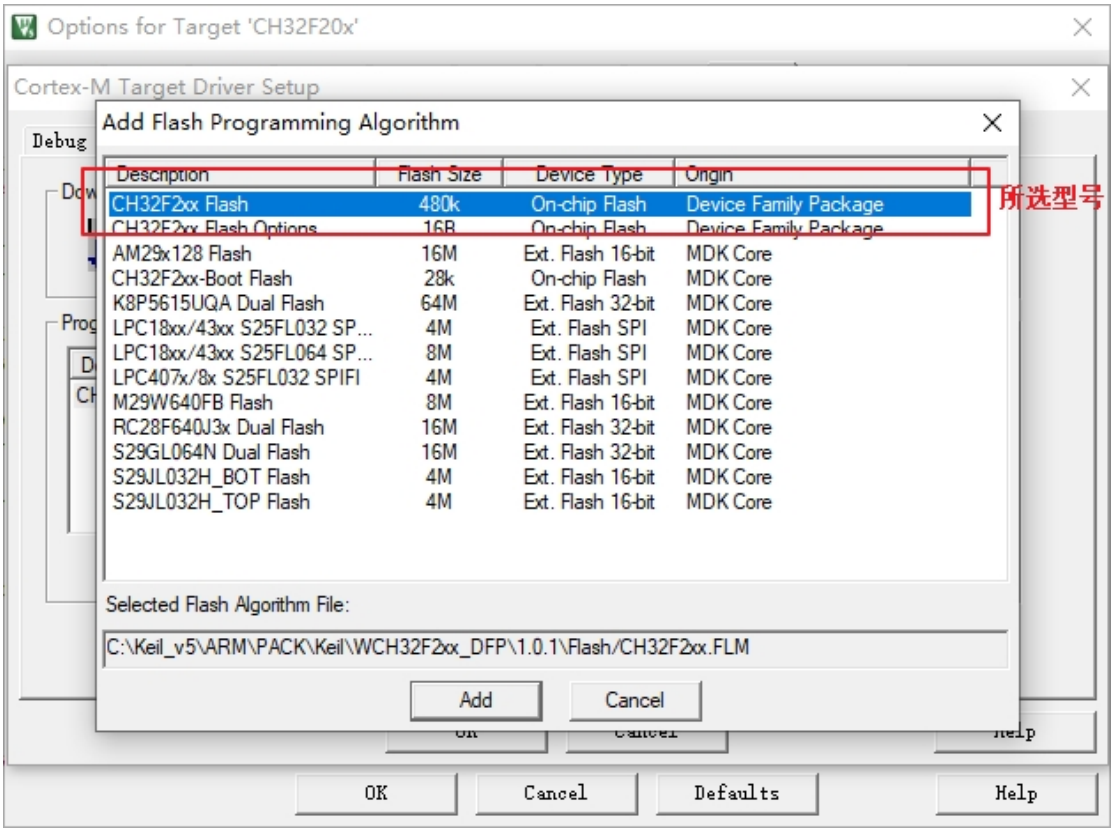


图 4 Flash 算法文件选择示意图

3.3 CH32F20xEVT.ZIP 使用注意事项

3.3.1 芯片型号对应 EVT 中函数定义

本示例针对以 CH32F203C8T6 为例，对例程更改步骤如下：

1. 更改 CH32F20x.h 文件，如图 9 所示，选用 CH32F20x_D6。（如注释所示，适用于 CH32F203K8、CH32F203C6、CH32F203C）。

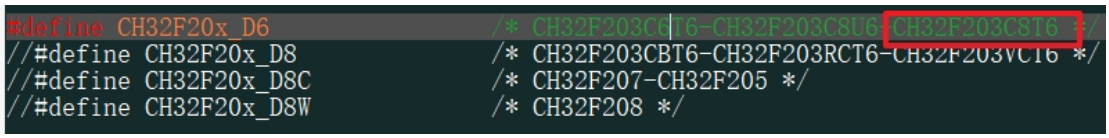


图 5 CH32F20x.h 文件更改

2. 更改 startup 文件，点击 Manage Project Items 进行文件更换，选用 startup_ch32f20x_ D6.s 文件，进行替换，文件更换见图 10（若换用其他类型芯片，选用启动文件参照图 9 注释）。

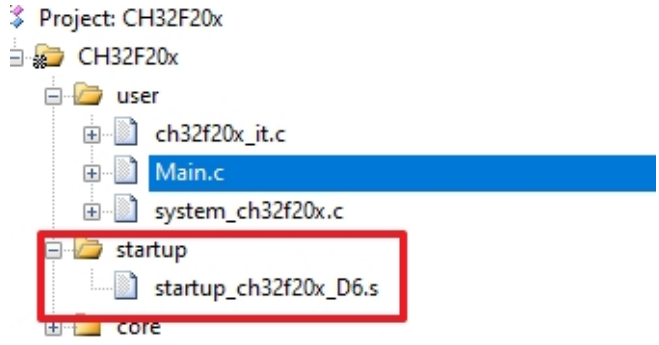


图 6 startup 文件更换

3. 选用芯片类型，点击 Options for Target…. 选用 CH32F203C8，如图 11 所示。

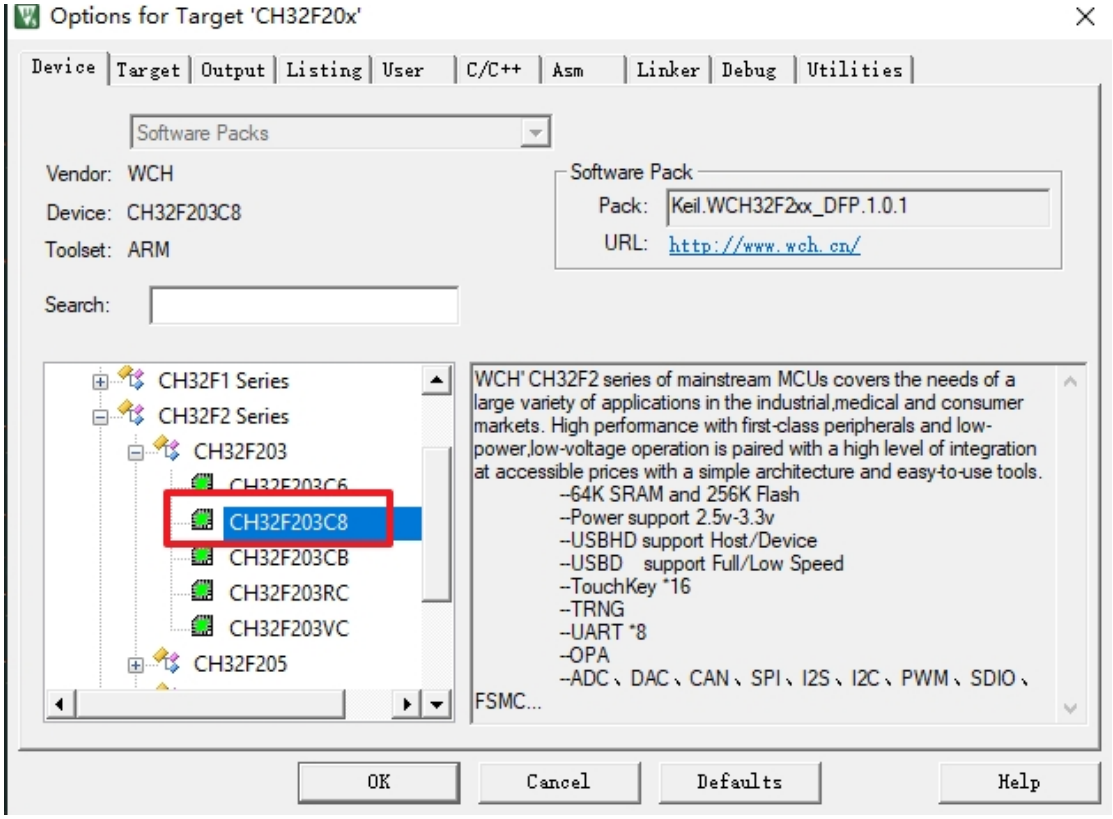
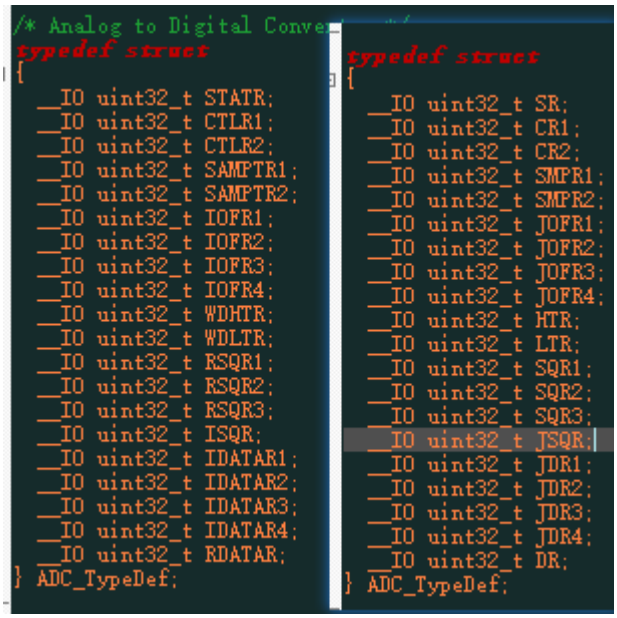


图 7 CH32F203C8 芯片选择

3.3.2 寄存器名字对比

在使用 EVT 时，CH32 寄存器名称和 STM32 不同，需注意，具体可参考底层驱动文件 ch32f20x.h。



3.2 使用 WCH-Link 开发 CH32F203C8T6

CH32F203C8T6的开发板自带WCH-Link，可以用电路板上的WCH-Link调试仿真代码，操作方法如下：

1. 在Options for Target -> Debug 中选择“CMSIS-DAP Debugger”，部分客户反馈找不到这一驱动器选项，那是因为MDK版本过低，只有Keil4.74以上的版本和Keil5才支持CMSIS-DAP Debugger选项，如图5所示。

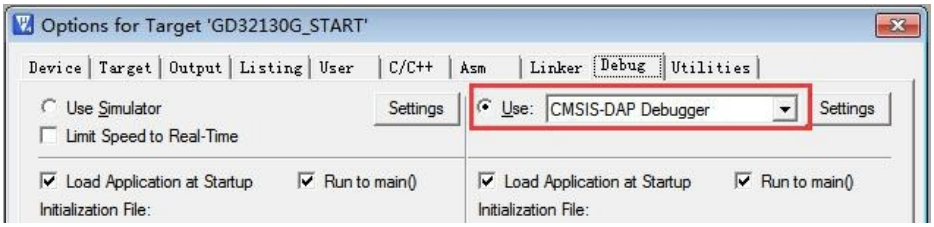


图 8 WCH-Link 选择 Debugger 类型

2. 在Options for Target -> Utilities，也要选择“CMSIS-DAP Debugger”，如图6所示进行选择。

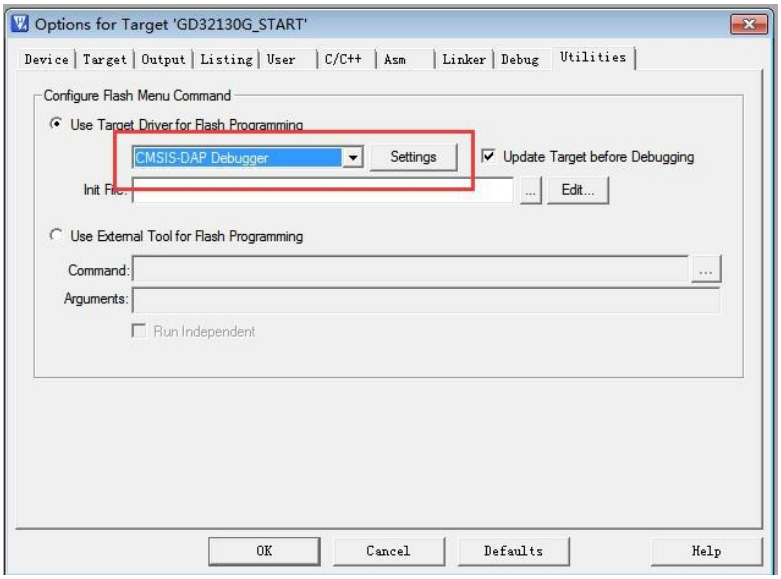


图 9 GD-Link 在 Utilities 中选择 Debugger 类型

3. 在Options for Target -> Debug ->Settings勾选SWJ、Port选择 SW。右框IDcode会出现”0xXBAXXXXX”，如图7所示进行勾选。

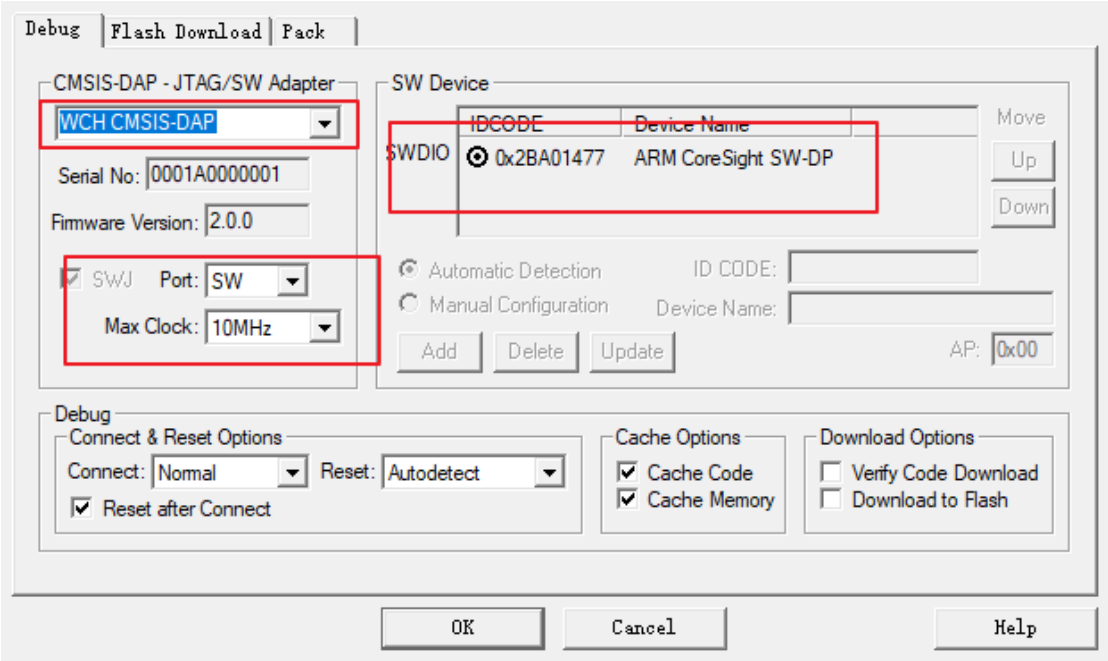


图 10 WCH-Link 成连接目标板示意图

4. 在Options for Target -> Debug ->Settings -> Flash Download中添加CH32的flash算法，如图8所示。

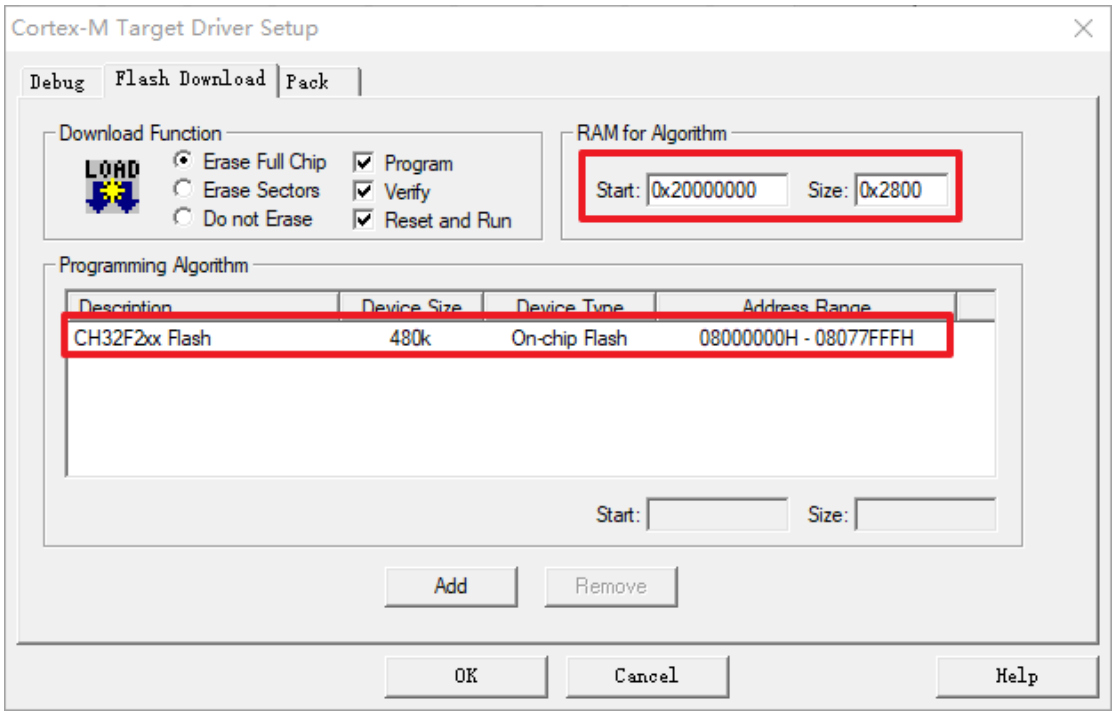


图 11 WCH-Link 添加 Flash 算法文件示意图

完成如上操作，点击 OK，即可对编译完的程序进行下载调试。

注：在使用其他 LINK（如 ST-Link、JLink 等）下载时，RAM for Algorithm 配置 Size 0x2800

4 外设移植注意点

4.1 RCC

4.1.1 HSE

CH32 的 HSE 起振时间为 2.5ms：CH32F203C8T6 的晶体超时时间配置见图 12。该配置在底层驱动文件 ch32f20x.h。

```
/* In the following line adjust the External High Speed oscillator (HSE) Startup Timeout value */
#define HSE_STARTUP_TIMEOUT    ((uint16_t)0x1000) /* Time out for HSE start up */
```

图 12 CH32F203C8T6 的晶体超时时间设置

4.1.2 PLL

CH32 的 HSI 和 HSI/2 作为 PLL 的源，STM32 只有 HSI/2 作为 PLL 的源。可通过若使用 HSI 做 PLL 源，需软件配置 EXTEN 寄存器，具体使用可参考 CH32F20xEVT 中 EXAM\RCC\HSI_PLL_Source。

```
#if (PLL_Source == HSI)
    EXTEN->EXTEN_CTR |= EXTEN_PLL_HSI_PRE;

#endif
```

图 13 EXTEN 寄存器配置

CH32 的 PLL 倍频系数更多，支持最大主频 144MHz。

```
/* for other CH32F20x */
#define  RCC_PLLMULL2                    ((uint32_t)0x00000000)        /* PLL input clock*2 */
#define  RCC_PLLMULL3                    ((uint32_t)0x00040000)        /* PLL input clock*3 */
#define  RCC_PLLMULL4                    ((uint32_t)0x00080000)        /* PLL input clock*4 */
#define  RCC_PLLMULL5                    ((uint32_t)0x000C0000)        /* PLL input clock*5 */
#define  RCC_PLLMULL6                    ((uint32_t)0x00100000)        /* PLL input clock*6 */
#define  RCC_PLLMULL7                    ((uint32_t)0x00140000)        /* PLL input clock*7 */
#define  RCC_PLLMULL8                    ((uint32_t)0x00180000)        /* PLL input clock*8 */
#define  RCC_PLLMULL9                    ((uint32_t)0x001C0000)        /* PLL input clock*9 */
#define  RCC_PLLMULL10                   ((uint32_t)0x00200000)        /* PLL input clock10 */
#define  RCC_PLLMULL11                   ((uint32_t)0x00240000)        /* PLL input clock*11 */
#define  RCC_PLLMULL12                   ((uint32_t)0x00280000)        /* PLL input clock*12 */
#define  RCC_PLLMULL13                   ((uint32_t)0x002C0000)        /* PLL input clock*13 */
#define  RCC_PLLMULL14                   ((uint32_t)0x00300000)        /* PLL input clock*14 */
#define  RCC_PLLMULL15                   ((uint32_t)0x00340000)        /* PLL input clock*15 */
#define  RCC_PLLMULL16                   ((uint32_t)0x00380000)        /* PLL input clock*16 */
#define  RCC_PLLMULL18                   ((uint32_t)0x003C0000)        /* PLL input clock*18 */
```

图 14 PLL 倍频系数

4.2 ADC

4.2.1 ADC 校准

CH32 校准后需软件处理。具体操作参考 ADC 例程。

```
/*******************************************************************************
 * @fn      Get_CalibrationValue
 *
 * @brief   Get ADCx Calibration Value.
 *
 * @param   ADCx - where x can be 1 to select the ADC peripheral.
 *
 * @return  CalibrationValue
 */
int16_t Get_CalibrationValue( ADC_TypeDef *ADCx )
{
```

图 15 ADC 校验程序

4.3 CAN

4.3.1 CAN 过滤器表

CH32 因设计上的不同，需做软件处理，过滤器表向后偏移了 256Byte。具体操作参考 CAN 例程，CH32F20xEVT 底层库 ch32f20x_can.c 已做处理。

```
#if defined (CH32F20x_D6)
    {
        uint32_t i;

        for(i = 0; i < 64; i++){
            *(__IO uint16_t *)(0x40006000 + 512 + 4 * i) = *(__IO uint16_t *)(0x40006000 + 768 + 4 * i);
            *(uint16_t *)(0x40006000 + 768 + 4 * i) = 0;
        }
    }
```

图 16 CAN 软件处理

4.4 USBD

4.4.1 System

在使用 USBD 时，STM32 可使用主频为 48MHz(1 分频)和 72MHz（1.5 分频），CH32 可使用主频为 48MHz(1 分频)、96MHz(2 分频)和 144MHz（3 分频），若用户使用 USBD，只能选上述三个主频配置。

4.4.2 USB 引脚内置电阻

STM32 需外接电阻，CH32 内置电阻。具体操作参考可参考 CH32F20xEVT 中 EXAM\RCC\USB\USBD 相关例程。

```
/*********************************************************************
 * @fn      USB_Port_Set
 *
 * @brief   Set USB IO port.
 *
 * @param   NewState - DISABLE or ENABLE.
 *          Pin_In_IPU - Enables or Disables intenal pull-up resistance.
 *
 * @return  none
 */
void USB_Port_Set(FunctionalState NewState, FunctionalState Pin_In_IPU)
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

  if(NewState) {
    _SetCNTR(_GetCNTR()&(~(1<<1)));
    GPIOA->CFGHR&=0XFFF00FFF;
    GPIOA->OUTDR&=~(3<<11); //PA11/12=0

#if defined (CH32F20x_D6) || defined (CH32F20x_D8W)
    GPIOA->CFGHR|=0X00044000; //float
#else
    GPIOA->CFGHR|=0X00088000; // IPD
#endif

  }
  else
  {
    _SetCNTR(_GetCNTR()|(1<<1));
    GPIOA->CFGHR&=0XFFF00FFF;
    GPIOA->OUTDR&=~(3<<11); //PA11/12=0
    GPIOA->CFGHR|=0X00033000; // LOW
  }
#ifdef USBD_SPEED_LS
  EXTEN->EXTEN_CTR |= EXTEN_USBD_LS
#else
```

图 17 CH32 内置电阻配置

4.4.3 程序差异点

CH32 端点状态寄存器初始化时需进行全清操作，全清操作见下图；CH32 的 USB_EPnR 寄存器，需多次写才能写入，故在使用时，需写完后需判断是否写入，CH32F20xEVT 底层库 usb_regs.c 已做处理。

```
/*******************************************************************
 * @fn      USB_init.
 *
 * @brief   init routine.
 *
 * @return  none
 */
void USBD_init(void)
{
  uint8_t i;

  pInformation->Current_Configuration = 0;
  PowerOn();
  for (i=0;i<8;i++) _SetENDPOINT(i, _GetENDPOINT(i) & 0x7F7F & EPREG_MASK);//all clear
  _SetISTR((uint16_t)0x00FF);//all clear
  USB_SIL_Init();
  bDeviceState = UNCONNECTED;
}
```

```
/*******************************************************************************
* Macro Name     : SetEPTxStatus
* Description    : sets the status for tx transfer (bits STAT_TX[1:0]).
* Input          : bEpNum: Endpoint Number.
*                  wState: new state
* Return         : None.
*******************************************************************************/
#define _SetEPTxStatus(bEpNum,wState) {\
    register uint16_t _wRegVal;        \
    _wRegVal = _GetENDPOINT(bEpNum) & EPTX_DTOGMASK;\
    /* toggle first bit ? */       \
    if((EPTX_DTOG1 & wState)!= 0)        \
      _wRegVal ^= EPTX_DTOG1;          \
    /* toggle second bit ?  */         \
    if((EPTX_DTOG2 & wState)!= 0)        \
      _wRegVal ^= EPTX_DTOG2;          \
    _SetENDPOINT(bEpNum, (_wRegVal | EP_CTR_RX|EP_CTR_TX));     \
    while( ( _GetENDPOINT(bEpNum) & EPTX_STAT ) != wState ) \
    { \
        _SetENDPOINT(bEpNum, (_wRegVal | EP_CTR_RX|EP_CTR_TX));    \
    }; \
  } /* _SetEPTxStatus */
```

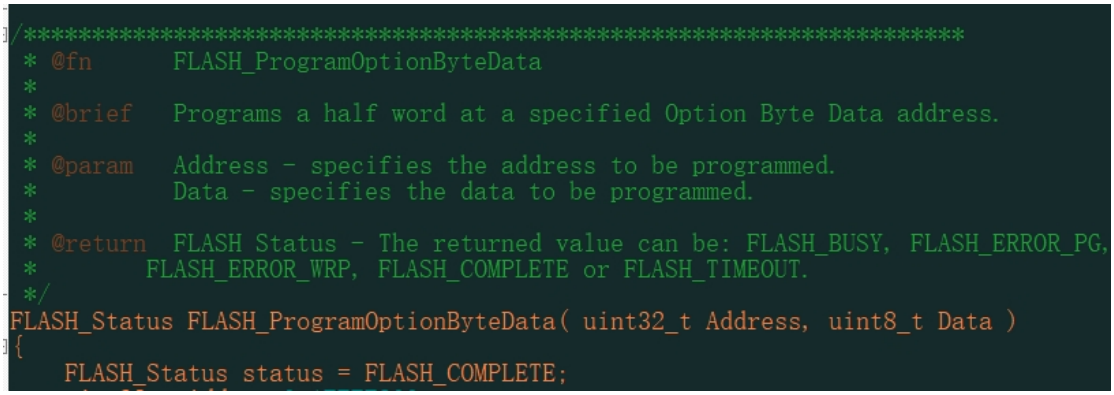
图 18 CH32 初始化

4.5 FLASH

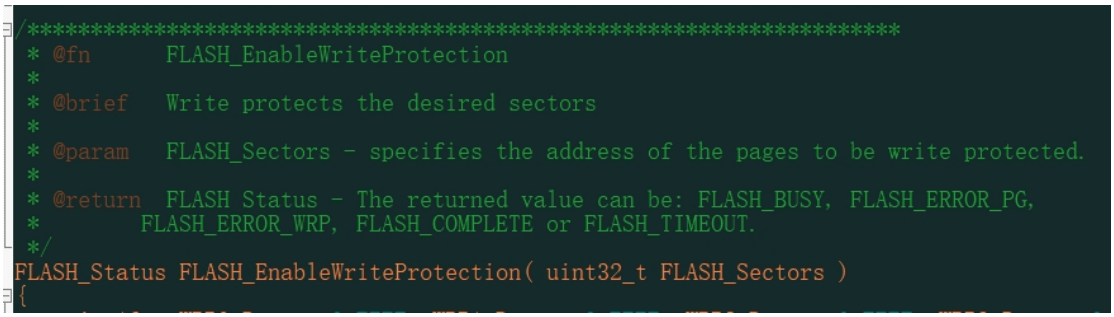
CH32 FLASH 编程支持半字编程和 STM32 做兼容，但速度较慢，推荐使用快速编程（256B-1Page）。FLASH 擦除后，STM32 字读为 0xFFFFFFFF，CH32 字读为 0xe339e339，半字读 0xe339，偶地址字节读 0x39，奇地址读 0xe3。

4.5.1 用户字操作

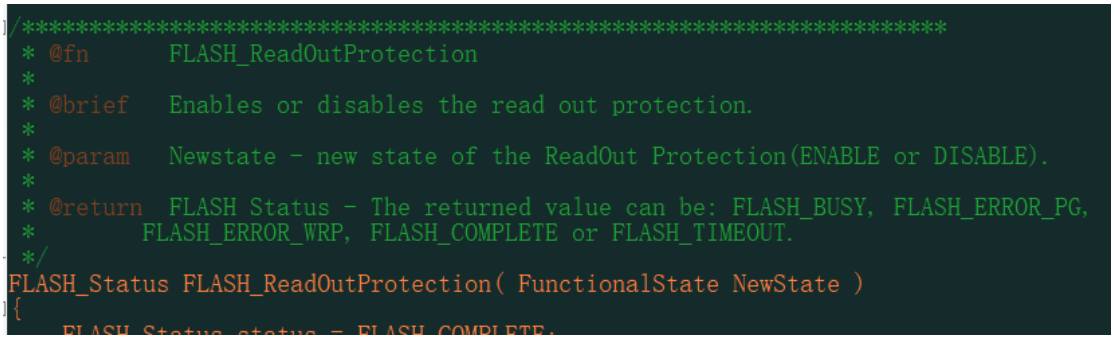
用户字 FLASH 擦除后不为 0xFF，故若需要修改用户字，需先读出保存，擦用户字，将需要修改的配置字和之前读出的一起写入。具体操作参考可参考 CH32F20xEVT 中底层库 ch32f20x_flash.c 中相关库函数处理。



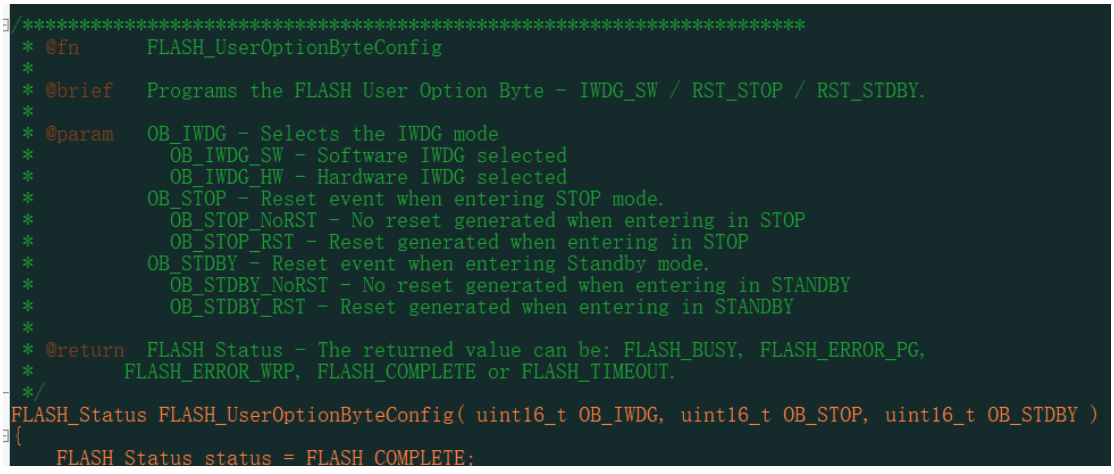
```
/*******************************************************************************
 * @fn      FLASH_ProgramOptionByteData
 *
 * @brief   Programs a half word at a specified Option Byte Data address.
 *
 * @param   Address - specifies the address to be programmed.
 *          Data - specifies the data to be programmed.
 *
 * @return  FLASH Status - The returned value can be: FLASH_BUSY, FLASH_ERROR_PG,
 *        FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.
 */
FLASH_Status FLASH_ProgramOptionByteData( uint32_t Address, uint8_t Data )
{
    FLASH_Status status = FLASH_COMPLETE;
```



```
/*******************************************************************************
 * @fn      FLASH_EnableWriteProtection
 *
 * @brief   Write protects the desired sectors
 *
 * @param   FLASH_Sectors - specifies the address of the pages to be write protected.
 *
 * @return  FLASH Status - The returned value can be: FLASH_BUSY, FLASH_ERROR_PG,
 *        FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.
 */
FLASH_Status FLASH_EnableWriteProtection( uint32_t FLASH_Sectors )
{
```



```
/*******************************************************************************
 * @fn      FLASH_ReadOutProtection
 *
 * @brief   Enables or disables the read out protection.
 *
 * @param   Newstate - new state of the ReadOut Protection(ENABLE or DISABLE).
 *
 * @return  FLASH Status - The returned value can be: FLASH_BUSY, FLASH_ERROR_PG,
 *        FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.
 */
FLASH_Status FLASH_ReadOutProtection( FunctionalState NewState )
{
    FLASH_Status status = FLASH_COMPLETE;
```



```
/*******************************************************************************
 * @fn      FLASH_UserOptionByteConfig
 *
 * @brief   Programs the FLASH User Option Byte - IWDG_SW / RST_STOP / RST_STDBY.
 *
 * @param   OB_IWDG - Selects the IWDG mode
 *            OB_IWDG_SW - Software IWDG selected
 *            OB_IWDG_HW - Hardware IWDG selected
 *          OB_STOP - Reset event when entering STOP mode.
 *            OB_STOP_NoRST - No reset generated when entering in STOP
 *            OB_STOP_RST - Reset generated when entering in STOP
 *          OB_STDBY - Reset event when entering Standby mode.
 *            OB_STDBY_NoRST - No reset generated when entering in STANDBY
 *            OB_STDBY_RST - Reset generated when entering in STANDBY
 *
 * @return  FLASH Status - The returned value can be: FLASH_BUSY, FLASH_ERROR_PG,
 *        FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.
 */
FLASH_Status FLASH_UserOptionByteConfig( uint16_t OB_IWDG, uint16_t OB_STOP, uint16_t OB_STDBY )
{
    FLASH_Status status = FLASH_COMPLETE;
```

图 19 CH32 用户字 FLASH 操作

4.5.2 用户 FLASH 操作

STM32 支持半字编程，支持 4KB 擦和全擦。CH32 在此基础上增加快速页编程（256B）、4K 擦、32K 擦和 64K 擦。CH32 半字编程较慢，故推荐使用快速页编程。具体操作参考可参考 CH32F20xEVT 中底层库 ch32f20x_flash.c 中相关库函数处理或

EVT\EXAM\FLASH\FLASH_Program FLASH 例程。

```
/*******************************************************************************
 * @fn      FLASH_Unlock_Fast
 *
 * @brief   Unlocks the Fast Program Erase Mode.
 *
 * @return  none
 */
void FLASH_Unlock_Fast( void )
```

```
/*******************************************************************************
 * @fn      FLASH_Lock_Fast
 *
 * @brief   Locks the Fast Program Erase Mode.
 *
 * @return  none
 */
```

```
/*******************************************************************************
 * @fn      FLASH_ErasePage_Fast
 *
 * @brief   Erases a specified FLASH page (1page = 256Byte).
 *
 * @param   Page_Address - The page address to be erased.
 *
 * @return  none
 */
```

```
/*******************************************************************************
 * @fn      FLASH_EraseBlock_32K_Fast
 *
 * @brief   Erases a specified FLASH Block (1Block = 32KByte).
 *
 * @param   Block_Address - The block address to be erased.
 *
 * @return  none
 */
void FLASH_EraseBlock_32K_Fast( uint32_t Block_Address )
```

```
/*******************************************************************************
 * @fn      FLASH_EraseBlock_64K_Fast
 *
 * @brief   Erases a specified FLASH Block (1Block = 64KByte).
 *
 * @param   Block_Address - The block address to be erased.
 *
 * @return  none
 */
void FLASH_EraseBlock_64K_Fast( uint32_t Block_Address )
```

```
/*********************************************************************
 * @fn      FLASH_EraseBlock_64K_Fast
 *
 * @brief   Erases a specified FLASH Block (1Block = 64KByte).
 *
 * @param   Block_Address - The block address to be erased.
 *
 * @return  none
 */
void FLASH_EraseBlock_64K_Fast( uint32_t Block_Address )
```

图 20 CH32 用户 FLASH 操作

4.5.3 FLASH 取指等待时延

STM32 在不同主频下 FLASH 取指需要时延，对应关系即 零等待（0-24MHz）、一等待（24MHz-48MHz）、两等待（48MHz-72MHz）。

CH32 在不同主频下都是零等待。具体操作参考可参考 CH32F20xEVT 中每个例程下的 system_ch32f20x.c 配置。

4.6 SPI

4.6.1 SPI 支持最高时钟频率

STM32 支持时钟最高 36MHz。CH32 支持时钟最高为 72MHz,若要使用高时钟频率需特殊配置。36Mhz-72MHz 为高速模式，该模式仅在时钟 2 分频（即 CTLR1 寄存器中 BR=000）时有效。使能高速模式，需将 HSCR 寄存器位 0 ，置 1。

4.6.2 一主多从，从机发送数据出错

SPI 从机模式 MISO 不被片选时，CH32 为复用推挽，ST 为高阻态。使用一主多从功能时，需从机在不被片选时，关闭 SPI，保持 MISO 引脚为高阻态。从机通过检测片选引脚为低电平使能 SPI，高电平关闭 SPI。此时主机控制片选引脚拉低后，发送数据需有一定延时，确保从机被片选后有足够时间使能 SPI。

4.7 DMA

4.7.1 相关寄存器操作不通点

在 DMA 通道使能后，CH32 PADDRx 和 MADDRx 不可写，STM32 CMAPx、CPARx 都可以写（官方手册中不建议这样操作）。

注：PADDRx 对应 CMAPx，MADDRx 对应 CPARx。

4.8 RTC

4.8.1 RTC 闹钟唤醒后，闹钟中断标志未被置位

RTC 闹钟中断中判断的标志由 RTC_IT_ALR，更改为 EXTI_Line17 中断标志。