CIU98_B V2 Host SDK[V1.2]

-Stm32l433

用户移植手册

V1.1

北京中电华大电子设计有限责任公司
CEC Huada Electronic Design Co.,Ltd

2021 年 12 月

声　明

变更记录

版本	修改描述	日期
V1.0	初稿	2021-7-3
V1.1	修改参考资料中的文档名，添加“V2”、“Stm32l433”字样；	2021-12-13

目 录

1 引言

本文档主要介绍 CIU98_B V2 Host SDK[1.2]-Stm32l433（以下简称 SDK）的代码移植以及相关注意事项，帮助客户快速使用 SDK。

2 参考资料

《CIU98_B V2 Host SDK[V1.2]-Stm32l433 API 接口说明.chm》V1.1
《CIU98_B V2 Host SDK[V1.2]-Stm32l433 用户使用手册.doc》V1.1

3 SDK 移植

基于 ST MCU（例如 STM32L433）的硬件平台，本 SDK 提供了符合《HED_SPI 通讯协议规范 2.0》和《HED_I2C 通讯协议规范 2.0》的通信协议层及硬件适配层的具体实现。SDK 的一级目录结构如下，用户需要将目录下需要的相关代码移植到自己实际的 MCU 平台上，并与其上层应用及底层驱动一并编译为板级程序。

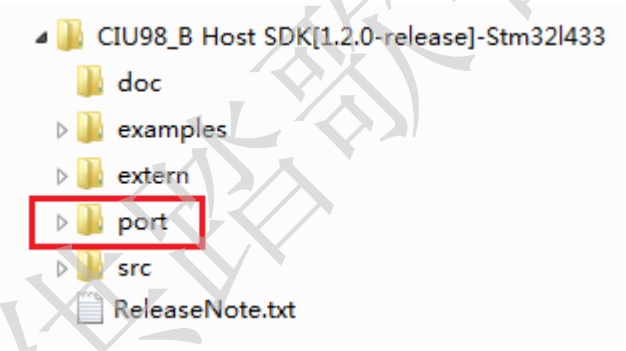


图 1 SDK 移植目录

用户移植时，建议用户首先将目录下的port和src文件夹整体拷贝到自己MCU平台的工程代码中，待通信接口调试通过后，再移植examples中需要的应用代码。

src主要提供了应用编程接口层、命令层、接口抽象层和链路通信协议层的相关代码实现，几乎不用修改，而用户只需重点关注port文件下硬件适配层的驱动代码，可以根据所使用的MCU和通信接口，参照代码源程序进行修改。

3.1 源文件介绍

Port目录下的源文件主要有port_spi、port_i2c、port_util和port_config，用户根据使用的具体MCU回退通信接口修改相应的源文件。

3.1.1 Port_spi

源文件	函数	说明	备注

port_spi.c port_spi.h	HAL_SPI_MspInit	用于SPI接口的4线IO驱动模式配置的初始化。	用户可根据当前使用MCU的SPI接口特点进行相应配置。
	HAL_SPI_DeInit	用于SPI接口的4线IO驱动模式配置的终止化。	用户可根据当前使用MCU的SPI接口特点进行相应配置。
	port_spi_init	SPI接口物理层参数配置（如通信模式、通信速率、IO驱动模式等），间接调用了函数HAL_SPI_MspInit	默认设置为：Mode0模式，通信速率10MHz，IO硬输出模式。 用户可根据实际需要修改参数.BaudRatePrescaler值来设置不同的通信速率。
	port_spi_deinit	直接调用了函数HAL_SPI_DeInit	
	port_spi_gpio_init	与SE RST复位引脚相连接的控制IO初始化。	
	以下为硬件适配层实现的函数，		
	port_spi_periph_init	SPI通信接口初始化	初始化SPI接口，并将RST控制IO默认设置为高电平。 调用的函数port_spi_init参数为SPI2，表示使用第2个SPI接口。
	port_spi_periph_deinit	SPI通信接口终止化	终止化SPI接口，并将RST控制IO设置为低电平，设置SPI通信接口的4个IO为低电平。
	port_spi_periph_chip_select	SPI通信使能或去使能	CS若设置为硬输出模式，直接调用__HAL_SPI_ENABLE或__HAL_SPI_DISABLE函数使能或去使能，若设置软模式，需以IO模式控制CS下拉或上拉。 若使用其它MCU，需根据所使用MCU特征进行相应的

			CS控制操作。 SDK默认为硬输出模式，用户可以修改port_spi.h的PORT_SPI_NSS_MODE宏参数来设置其它模式。
	port_spi_periph_transmit	发送多字节数据	调用HAL_SPI_Transmit函数发送数据。 用户移植时需修改为使用当前MCU所实现的SPI发送多字节数据的函数。
	port_spi_periph_receive	接收多字节数据	调用HAL_SPI_Receive函数接收数据。 用户移植时需修改为使用当前MCU所实现的SPI接收多字节数据的函数。
	port_spi_periph_control	根据输入的控制码，实现SE的控制，如RST引脚的复位控制。 由通信协议层调用。	用于控制RST引脚进行复位操作。 默认控制模式：RST先拉低，低电平持续时间为1ms，然后RST拉高后，高电平持续时间为10ms。 用户可以不用做任何修改
	port_spi_periph_delay	微秒级的延时。 由通信协议层调用。	用户需要根据当前使用的MCU，重新实现此函数或修改当前函数的for循环终止值，尽量保证微秒延时的准确性。
	port_spi_periph_timer_start	接收SE响应数据时，启动接收超时的计时。 由通信协议层调用。	调用了ST提供的获取计数值HAL_GetTick函数，计数频率值为1ms。用户需要根据当前使用的MCU，使用相应的ms计数函数。
	port_spi_periph_timer_differ	接收SE响应数据时，检测接收是否超时。 由通信协议层调用。	

3.1.2 Port_i2c

源文件	函数	说明	备注
port_i2c.c port_i2c.h	HAL_I2C_MspInit	用于I2C接口的SCL、SDA IO驱动模式配置的初始化。	用户可根据当前使用MCU的I2C接口特点进行相应配置。
	HAL_I2C_MspDeInit	用于I2C接口的SCL、SDA IO驱动模式配置的终止化。	用户可根据当前使用MCU的I2C接口特点进行相应配置。
	port_i2c_init	I2C接口物理层参数配置（如设备从地址、通信速率、IO驱动模式等），间接调用了函数HAL_SPI_MspInit	默认设置为：从地址为0x2A，通信速率400KHz，占空比16/9. 用户可根据实际需要修改参数.Timing值来设置不同的通信速率。
	port_i2c_deinit	直接调用了函数HAL_I2C_DeInit	
	port_i2c_gpio_init	与SE RST复位引脚相连接的控制IO初始化。	
	以下为硬件适配层实现的函数，		
	port_i2c_periph_init	I2C通信接口初始化	RST控制IO默认初始化为高电平。 调用函数port_i2c_init函数初始化I2C接口。
	port_i2c_periph_deinit	I2C通信接口终止化	RST控制IO设置为低电平
	port_i2c_periph_transmit	发送多字节数据	调用HAL_I2C_Master_Transmit函数发送数据。 用户移植时需修改为使用当前MCU所实现的I2C发送多字节数据的函数。
	port_i2c_periph_receive	接收多字节数据	调用HAL_I2C_Master_Receive函数接收数据。 用户移植时需修改为使用当

			前MCU所实现的I2C接收多字节数据的函数。
	port_i2c_periph_control	根据输入的控制码，实现SE的控制，如RST引脚的复位控制。 由通信协议层调用。	用于控制RST引脚进行复位操作。 默认控制模式：RST先拉低，低电平持续时间为1ms，然后RST拉高后，高电平持续时间为10ms。 用户可以不用做任何修改
	port_i2c_periph_delay	微秒级的延时。 由通信协议层调用。	用户需要根据当前使用的MCU，重新实现此函数或修改当前函数的for循环终止值，尽量保证微秒延时的准确性。
	port_i2c_periph_timer_start	接收SE响应数据时，启动接收超时的计时。 由通信协议层调用。	调用了ST提供的获取计数值HAL_GetTick函数，计数频率值为1ms。用户需要根据当前使用的MCU，使用相应的ms计数函数。
	port_i2c_periph_timer_differ	接收SE响应数据时，检测接收是否超时。 由通信协议层调用。	

3.1.3 Port_util

源文件	函数	说明	备注
port_util.c port_util.h	port_printf	log调试输出接口，支持不同打印输出格式	用户可以根据使用的MCU特征或打印需求，修改此函数
	port_power_on	控制SE供电电源上电	如果用户使用IO来控制SE供电电源的导通与断开（例如控制CMOS管），可使用此函数来设置控制IO的电平值。 默认设置低电平将导通供电电源，设置高电平将断
	port_power_off	控制SE供电电源下电	

			开供电电源。

3.1.4 Port_config

源文件	函数	说明	备注
port_config.c port_config.h	port_system_clock_config	MCU系统时钟初始化	用户可以根据使用的MCU初始化需要的系统时钟。默认系统时钟设置为80MHz。
	port_gpio_init	电源控制IO，LED控制IO初始化。	如果有SE供电电源控制、LED指示灯控制操作需求，可以在此函数进行IO功能初始化。
	port_printf_init	串口打印输出初始化	用户若使用UART打印输出Log信息，需要使用此函数初始化UART通信接口。默认设置UART1接口、波特率为115200bps、1个停止为、无奇偶校验位。
	port_mcu_init	MCU上电初始化	调用上述初始化函数。

3.2 SPI 移植注意事项

当进行SPI通信接口移植时，用户可以关注上述源文件介绍的相关备注信息，另外，用户在往自己实际平台移植时，可以重点关注如下几个事项。

3.2.1 IO 定义

SPI通信接口的4线IO和RST控制IO的定义可查看port_spi.h文件中的IO宏定义，例如，与RST引脚连接的控制IO由GPIOA7切换为GPIOA8时，可以将文件中的宏定义修改为如下值即可。

```
//SE0 RST  控制IO
#define PORT_SPI_SE0_RST_IO_PORT        GPIOA
#define PORT_SPI_SE0_RST_IO_PIN         GPIO_PIN_8
#define PORT_SPI_SE0_RST_IO_CLK_ENABLE()      __HAL_RCC_GPIOA_CLK_ENABLE()
```

3.2.2 通信速率设置

由于Port_config.c文件中函数port_system_clock_config将系统时钟设置为80MHz，且

Pclk时钟设置为系统时钟1分频，为了设置10MHz的SPI通信速率，在port_spi.c文件的port_spi_init中，将.BaudRatePrescaler参数值设置为“SPI_BAUDRATEPRESCALER_8”，即Pclk时钟的8分频。例如，如果需要设置5MHz的SPI通信速率，可以将.BaudRatePrescaler参数值设置为“SPI_BAUDRATEPRESCALER_16”，即Pclk时钟的16分频。

由于.BaudRatePrescaler参数值只能为2的倍数，所以通过此方式，SPI通信速率只能设置为10MHz/5MHz/2.5MHz….值，如果需要设置如11MHz或9MHz通信速率时，除了修改.BaudRatePrescaler参数外，还需相应地修改系统时钟。

3.2.3 片选 CS 控制模式设置

SPI通信的片选CS控制模式通常分为两种：硬输出模式和软输出模式，硬输出模式时，CS由SPI接口模块独自控制，例如对于STM32 MCU，在调用函数__HAL_SPI_ENABLE和函数__HAL_SPI_DISABLE时，片选CS控制引脚自动使能（CS低电平）及去使能（CS高电平）。而软输出模式时，片选CS控制引脚可以为任意的普通IO，可直接控制CS引脚低/高电平来使能或去使能。

SDK默认为硬输出模式，如果用户在同一个SPI接口上，除了SE外，还挂载了其它SPI通信的从设备，并使用IO来控制不同CS的引脚，用户可以将port_spi.h头文件中的宏定义PORT_SPI_NSS_MODE值修改为软输出模式即可

```
#define     PORT_SPI_NSS_SOFT           0x00 /*!< 软输出模式*/
#define     PORT_SPI_NSS_HARD_OUTPUT    0x01 /*!< 硬输出模式*/

#define     PORT_SPI_NSS_MODE     PORT_SPI_NSS_SOFT
```

3.2.4 通信时间参数

SPI通信相关的时间参数及其推荐值如下，SDK在协议设计时将这些参数进行了相应的宏定义，并将推荐值作为这些宏的默认值，原则上用户可以不做修改。

间隔时间名称	T7(ms)	T6(ms)	WPT (us)	T3(us)	T4(us)	T5(us)	T_BGT（us）
最小间隔时间	1	10	210	200	20	30	200

WPT、T3~T5、T_BGT参数的宏定义可查看头文件：proto_spi.h，例如宏定义如下：

```
#define SPI_SEND_CS_WAKEUP_TIME                 210   //us   WPT
#define SPI_SEND_DATA_OVER_WAIT_TIME            200   //us   T3
#define SPI_RESET_POLL_SLAVE_INTERVAL_TIME      20    //us   T4
#define SPI_BEFORE_RECEIVE_DATA_WAIT_TIME       30    //us   T5
#define SPI_SEND_BGT_TIME                       200   //us   T_BGT
```

而T7和T6参数宏定义可查看头文件：port_spi.h，例如宏定义如下

```
#define     PORT_SPI_SE_RST_LOW_DELAY        1000     //us  T7
#define     PORT_SPI_SE_RST_HIGH_DELAY       10000    //us T6
```

3.3 I2C 移植注意事项

当进行I2C通信接口移植时，用户可以关注上述源文件介绍的相关备注信息，另外，用户在往自己实际平台移植时，可以重点关注如下几个事项。

3.3.1 IO 定义

I2C通信接口的SDA、SCL和RST控制IO的定义可查看port_i2c.h文件中的IO宏定义，例如，与RST引脚连接的控制IO由GPIOA7切换为GPIOA8时，可以将文件中的宏定义修改为如下值即可。

```
//SE0 RST  控制IO
#define PORT_I2C_SE0_RST_IO_PORT        GPIOA
#define PORT_I2C_SE0_RST_IO_PIN         GPIO_PIN_8
#define PORT_I2C_SE0_RST_IO_CLK_ENABLE()    __HAL_RCC_GPIOA_CLK_ENABLE()
```

3.3.2 通信速率设置

原则上，用户需要根据自己所使用的MCU和通信速率要求来设置相应的通信速率和占空比，对于STM32硬件平台的MCU，可以通过修改port_i2c.c文件的port_i2c_init函数的参数.Timing来设置，在保持80MHz Pclk时钟不变的前提下，SDK提供了分别提供400KHz、100KHz通信速率以及16/9、2/1占空比参数宏定义，在port_i2c_init函数中，.Timing参数默认值为PORT_I2C_TIMING_400K_16_9，表示I2C接口的通信速率为400KHz，占空比为16/9。

如果需要设置I2C接口的通信速率为100KHz，占空比为2/1，可以通过修改port_i2c_init函数的参数.Timing值为PORT_I2C_TIMING_100K_2_1即可。

如果MCU系统时钟发生改变，那就必须相应地修改赋值给.Timing参数的宏定义的具体数值，数值的具体意义需参考MCU的使用手册。

3.3.3 寻址位数及从地址设置

如果SE的I2C寻址位数及从机地址未改变，保持SDK I2C通信的寻址位数及从地址为默认值即可，默认值为7位的0x2A。

如果I2C寻址位数由7为变为10位时，可以通过修改port_i2c.c文件的port_i2c_init函数的参数.AddressingMode来设置，例如将参数值修改为I2C_ADDRESSINGMODE_10BIT。

如果修改I2C从地址，实际上可以直接修改port_i2c.c文件的port_i2c_init函数的参数

.OwnAddress1值即可，也可以修改port_i2c.h文件中的PORT_I2C_ADDRESS_2A宏定义值。

3.3.4 通信时间参数

I2C通信相关的时间参数主要关注T7和T6，其参数宏定义可查看头文件：port_i2c.h，例如宏定义如下

```
#define     PORT_I2C_SE_RST_LOW_DELAY        1000     //us  T7
#define     PORT_I2C_SE_RST_HIGH_DELAY       10000    //us T6
```

3.4 通用功能注意事项

3.4.1 log 打印输出

在config.h文件中，宏“_DEBUG”为log信息打印输出总开关，屏蔽了此宏，所有log信息关闭，反之，log信息开启。

对于不同输出格式的log信息输出功能的开启与关闭，可以通过在log.h文件打开或屏蔽相应的宏，例如屏蔽如下宏定义，SDK中所有与LOGI相应的log输出功能将被关闭。

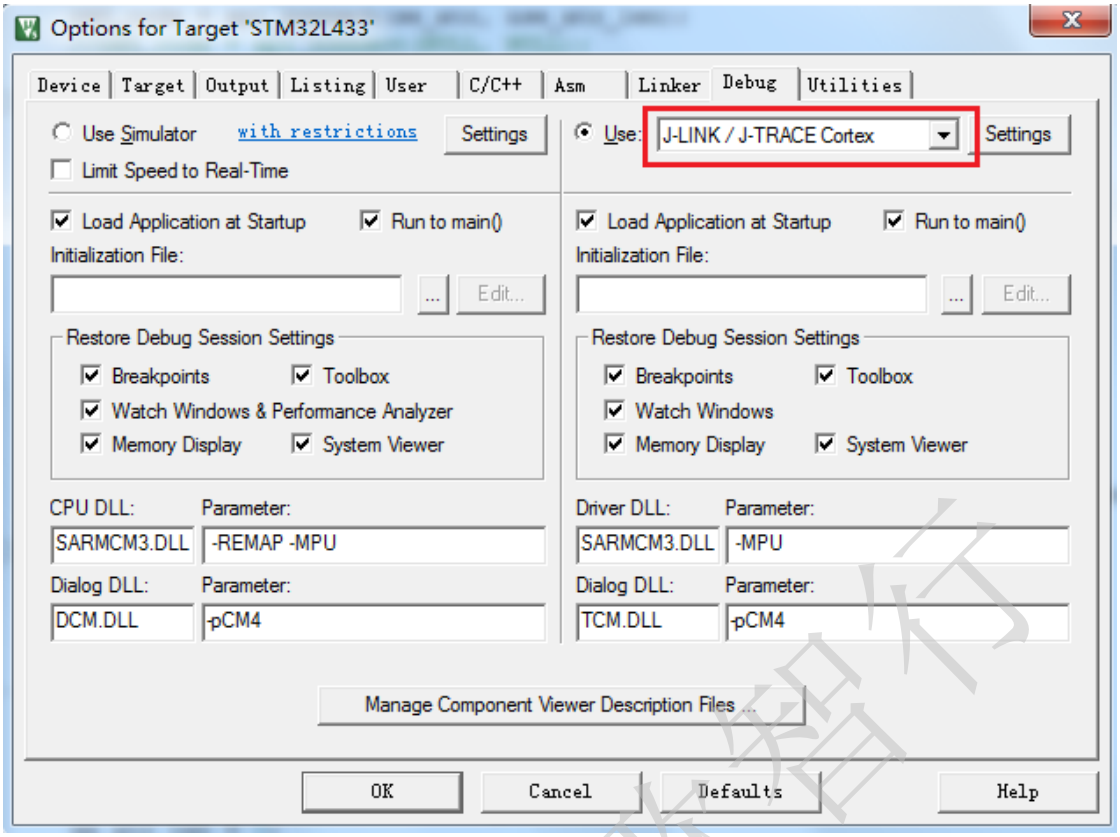
```
//#define LOGI(format, ...) LOG_TRACE(format, "I", __FILE__, __LINE__, __func__, __VA_ARGS__)
```

Log信息打印支持UART硬打印输出和Jlink软打印输出两种模式。

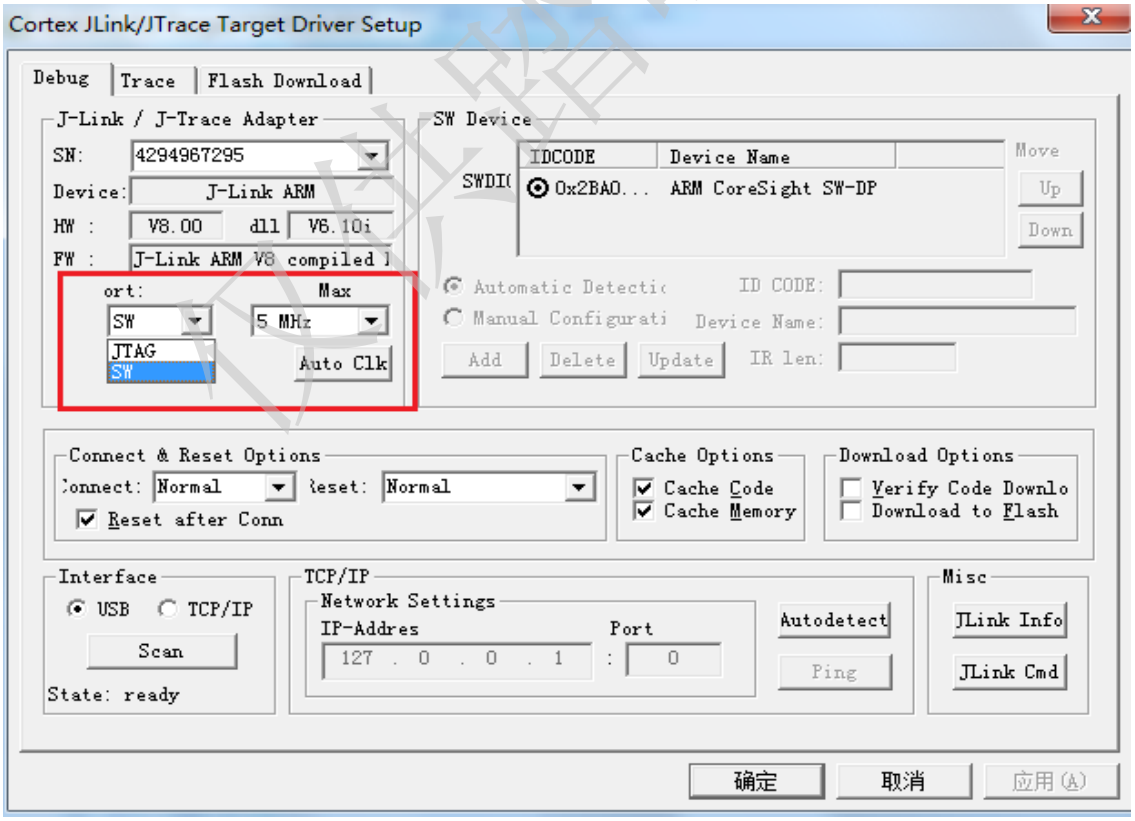
将config.h头文件中的PORT_UART_PRINTF_ENABLE宏设置为1，将使能UART硬打印输出模式，波特率默认设置为115200bsp，用户可以借助USB转UART的辅助设备向PC端输出打印信息。

将config.h头文件中的PORT_UART_PRINTF_ENABLE宏设置为0，UART硬打印输出模式被禁用，即切换为Jlink软打印模式，即可以在Keil工程调试运行下，直接借助Jlink输出打印的log信息。使用Jlink软打印模式，Keil工程目标选项设置如下：

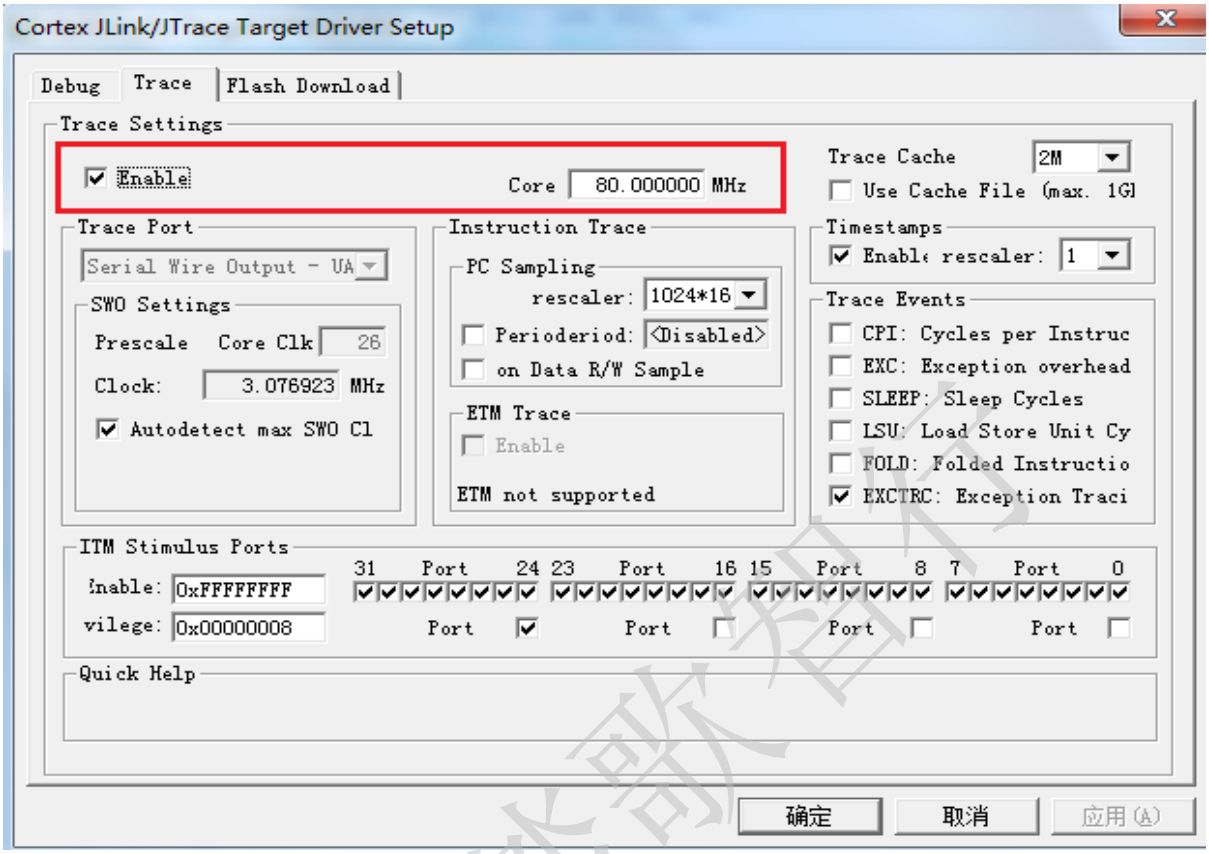
1.Debug页面下选择J-LINK



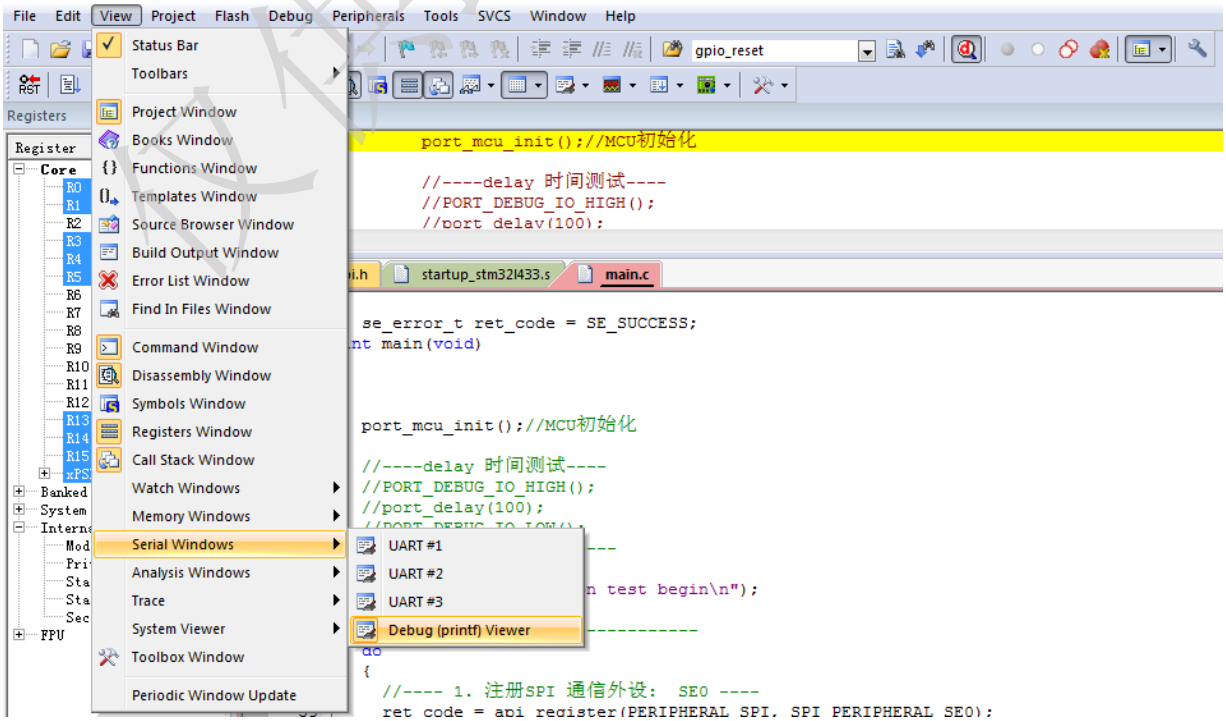
2.点击“Settings”进入设置界面，选择SW调试模式，时钟点击“Auto Clk”



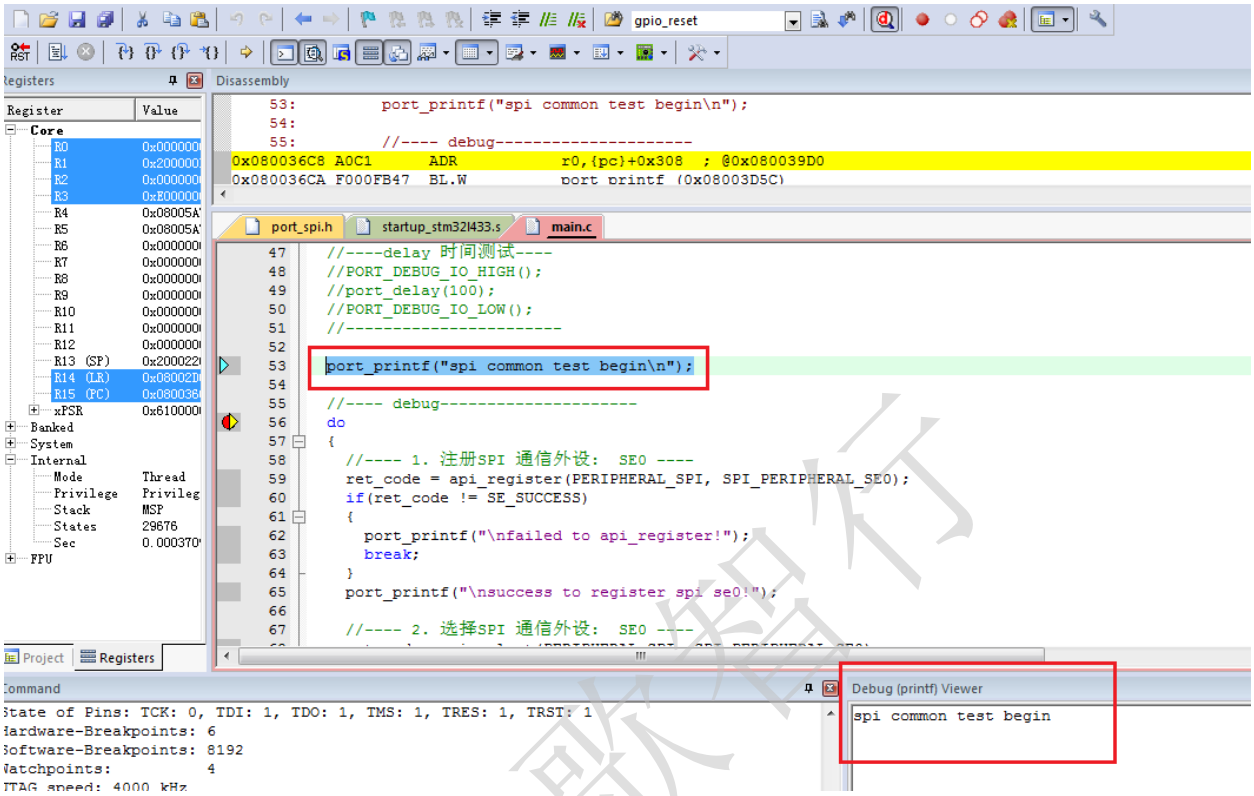
3. 点击“Trace”进入Trace界面，由于MCU系统时钟设置为80MHz，所以将Core值设置为80MHz。



4.启动“Debug”，程序准备运行，通过“View”—>“Serial Windows”—>“Debug (printf) Viewer”调出log输出框。



5.当程序运行时，若有 Log 输出信息，相应的信息输出如下图所示。



SDK 的 Keil 工程默认已设置了软打印输出格式，用户运行程序时，直接调出 Debug (printf) Viewer 显示框查看打印信息。

3.4.2 微秒延时精度

为减少用户在通信移植时引入的时序问题，建议用户移植时，应优先调试 port_spi.c 或 port_i2c.c 中微秒延时函数，并尽可能地提供延时函数的精度。若是硬件平台内在因素或程序实现不便性导致延时不准，提供的通信时间参数默认值可适当放大些，但不能减小。

3.4.3 RESET 请求帧

在调用 api_connect、api_reset 函数对 SE 进行连接或复位操作时，通过如下全局变量参数值来设置是否向 SE 发送 RESET 请求帧及接收响应。

源文件名称	全志变量	参数值	说明
proto_spi.c	g_bSPIHedOpenSeMode	HED20_SPI_OPEN_SE_RESET_REQ	在 api_connect 时，SPI 通信有 RESET 请求帧操作
		HED20_SPI_OPEN_SE_RESET_NONE	在 api_connect 时，SPI 通信无 RESET 请求帧操作

	g_bSPIHedRstSeMode	HED20_SPI_RESET_SE_RESET_REQ	在 api_reset 时，SPI 通信有 RESET 请求帧操作
		HED20_SPI_RESET_SE_RESET_NONE	在 api_reset 时，SPI 通信有 RESET 请求帧操作
proto_i2c.c	g_bI2cHedOpenSeMode	HED20_I2C_OPEN_SE_RESET_REQ	在 api_connect 时，I2C 通信有 RESET 请求帧操作
		HED20_I2C_OPEN_SE_RESET_NONE	在 api_connect 时，I2C 通信无 RESET 请求帧操作
	g_bI2cHedRstSeMode	HED20_I2C_RESET_SE_RESET_REQ	在 api_reset 时，I2C 通信有 RESET 请求帧操作
		HED20_I2C_RESET_SE_RESET_NONE	在 api_reset 时，I2C 通信有 RESET 请求帧操作

SDK 默认为均有 RESET 请求帧操作，即全局变量默认值如下。

```
uint8_t    g_bSPIHedOpenSeMode = HED20_SPI_OPEN_SE_RESET_REQ;
uint8_t    g_bSPIHedRstSeMode = HED20_SPI_RESET_SE_RESET_REQ;
uint8_t    g_bI2cHedOpenSeMode = HED20_I2C_OPEN_SE_RESET_REQ;
uint8_t    g_bI2cHedRstSeMode = HED20_I2C_RESET_SE_RESET_REQ;
```

3.4.4 ATR 请求帧

在调用 api_connect、api_reset 函数对 SE 进行连接或复位操作时，通过函数参数值可以设置是否向 SE 发送 ATR 请求帧及接收 ATR 数据。

函数调用方式	说明
api_connect(atr, &atrlen)	连接时获取 ATR 值
api_connect(NULL, NULL)	连接时不获取 ATR 值
api_reset(atr, &atrlen)	复位时获取 ATR 值
api_reset(NULL, NULL)	复位时不获取 ATR 值

SDK 默认为均获取 ATR 值。