CIU98_B V2 Host SDK[V1.2]

-Raspberrypi4B_Linux

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”、“Raspberrypi4B_Linux”字样；	2021-12-13

目 录

1 引言

本文档主要介绍 CIU98_B V2 Host SDK[1.2-release]-Raspberrypi4B_Linux（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]-Raspberrypi4B_Linux API 接口说明.chm》V1.1
《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux 用户使用手册.doc》V1.1

3 SDK 移植

基于树莓派（Raspberrypi4B）的 Linux 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的应用工程上，并与其上层应用及底层驱动一并编译为板级程序。

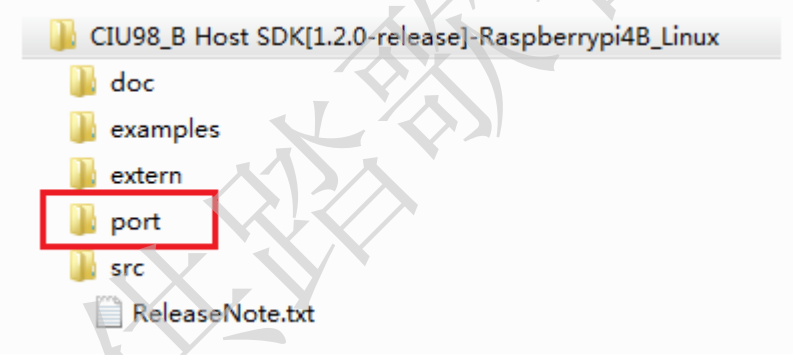

图 1 SDK 移植目录

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，可以根据所使用的通信接口情况，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	port_spi_set	用于设置SPI接口的通道端口、通信模式、通信速率和通信数据位数。	用户可不做修改。
	port_spi_tx	通过Linux ioctl方式实现发送多字节数据。	用户若已有其它方式实现的发送、接收多字节数据函数，可对应地修改此函数实现。
	port_spi_rx	通过Linux ioctl方式实现接收多字节数据。	
	port_spi_init	直接调用port_spi_set函数	用户可不做修改。
	port_spi_deinit	调用了close函数，关闭接口	用户可不做修改。
	port_spi_gpio_init	与SE RST复位引脚相连接的控制IO初始化。 若SPI CS设置为软输出模式时，需对CS控制IO进行初始化。	用户可以自实现RST复位控制IO的初始化，以及实现CS控制IO的初始化（SPI CS为软输出模式）。
	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。
	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。
	port_spi_periph_chip_select	SPI通信使能或去使能	SPI CS为硬输出模式时，此函数实际为空操作。 SPI CS为软输出模式时，此函数通过控制CS引脚来使能或去使能SPI通信的片选。
	port_spi_periph_transmit	发送多字节数据	调用port_spi_tx函数发送数据。
	port_spi_periph_receive	接收多字节数据	调用port_spi_rx函数接收数据。
	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。	函数实现通过调用计时用的系统函数gettimeofday，用户也可以使用其它方式来实现微秒级延时，但应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。	函数实现通过调用计时用的系统函数gettimeofday。用户也可以使用其它计时函数。
	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	port_i2c_set	用于设置I2C接口的通道端口、从机地址等。	用户可不做修改。
	port_i2c_tx	通过Linux ioctl方式实现发送多字节数据。	用户若已有其它方式实现的发送、接收多字节数据函数，可对应地修改此函数实现。
	port_i2c_rx	通过Linux ioctl方式实现接收多字节数据。	
	port_i2c_init	直接调用port_i2c_set函数	用户可不做修改。
	port_i2c_deinit	调用了close函数，关闭接口	用户可不做修改。
	port_i2c_gpio_init	与SE RST复位引脚相连接的控制IO初始化。	用户也可以自实现RST复位控制IO的初始化。
	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	初始化 I2C 接口，并将 RST 控制 IO 默认设置为高电平。
	port_i2c_periph_deinit	I2C通信接口终止化	终止化 I2C 接口，并将 RST

			控制 IO 设置为低电平。
	port_i2c_periph_transmit	发送多字节数据	调用port_i2c_tx函数发送数据。
	port_i2c_periph_receive	接收多字节数据	调用port_i2c_rx函数接收数据。
	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。	函数实现通过调用计时用的系统函数gettimeofday，用户也可以使用其它方式来实现微秒级延时，但应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。	函数实现通过调用计时用的系统函数gettimeofday。用户也可以使用其它计时函数。
	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。
	port_power_off	控制SE供电电源下电	

			默认设置低电平将导通供电电源，设置高电平将断开供电电源。
	port_lock	用于通信接口资源保护的上锁	
	port_unlock	用于通信接口资源保护的解锁	

3.1.4 Port_config

源文件	函数	说明	备注
port_config.c port_config.h	port_gpio_export	通知Linux系统导出需要控制的GPIO引脚	定义的IO操作函数主要用于RST控制引脚的初始化，以及高、低电平控制，用户可以使用其它IO操作函数来替代这些函数的调用。
	port_gpio_unexport	通知Linux系统隐藏需要控制的GPIO引脚	
	port_gpio_direction	配置GPIO为输入或输出模式	
	port_gpio_write	控制GPIO输出高、低电平	
	port_gpio_read	获得GPIO输入电平状态	
	port_gpio_init	电源控制IO初始化。	如果有SE供电电源控制操作需求，可以在此函数进行IO功能初始化。
	port_mcu_init	MCU上电初始化	只调用了port_gpio_init，用户可以添加其它初始化函数。

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，用户在往自己工程移植时，可以重点关注如下几个事项。

3.2.1 IO 定义

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GPIO.1（功能名）切换为GPIO.4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      18
#define PORT_SPI_SE0_RST_IO      23
```

3.2.2 通信速率设置

SPI通信接口的CLK时钟频率，可以通过修改config.h文件中的SPI_SPEED定义，默认值为15000000，表示CLK时钟频率设置为15MHz。。

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接口驱动独自控制，无需用户额外操作。而软输出模式时，片选CS控制引脚可以为任意的普通IO，即需额外控制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.c文件中的全局变量spi_comm_parm_se0的最后一个初始参数值设置为PORT_SPI_CS_CTRL_MODE_SOFT即可

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，用户在往自己工程移植时，可以重点关注如下几个事项。

3.3.1 IO 定义

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GPIO.1（功能名）切换为GPIO.4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      18
#define PORT_SPI_SE0_RST_IO      23
```

3.3.2 通信速率设置

I2C通信接口默认的总线速率为100Kb/s，SDK未提供修改通信速率API接口，用户可以通过修改Linux系统的/boot/config,txt文件设置其它总线速率。例如修改为400Kb/s通信速率，操作如下。

输入命令：sudo vim /boot/config.txt

查找包含“dtparam=i2c_arm=on”的行，添加“，i2c_arm_baudrate=400000”，其中400000是新设置的速率400Kb/s，主要i2c前面的逗号。

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。

如果修改I2C从地址，可以修改config.h文件中的I2C_SLAVE_ADRR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__)

3.4.2 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;
```

3.4.3 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。

3.4.4 通信接口选择

SDK 默认支持 SPI 和 I2C 两种通信接口，用户可以根据实际所使用的接口情况，通过修改 port_config.h 文件的宏定义选择对应的通信接口。例如只有 SPI 通信接口时，宏定义如下：

```
#define HED_SPI_SE0_ENABLE     1   //SPI 接口  使能

#define HED_I2C_SE0_ENABLE     0   //I2C 接口  失能
```