I2C 教程

——疯壳·开发板系列

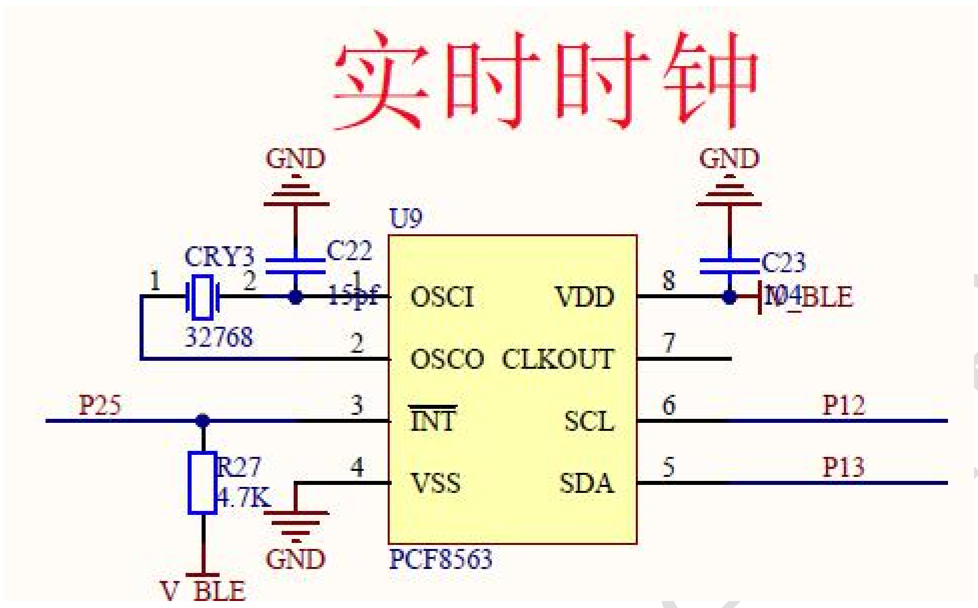
Wolverine-Team

2015/7/24

目录

第一节 RTC 硬件电路

实时时钟芯片为 I2C 接口，分别接在 MCU 的 P12 与 P13 引脚，中断引脚接在 P25 引脚，如下图所示：



第二节 I2C

2.1 I2C 介绍

I2C 总线是一个为系统中电路通信提供支持的可编程控制总线，它是一个软件定义的两线通信协议。

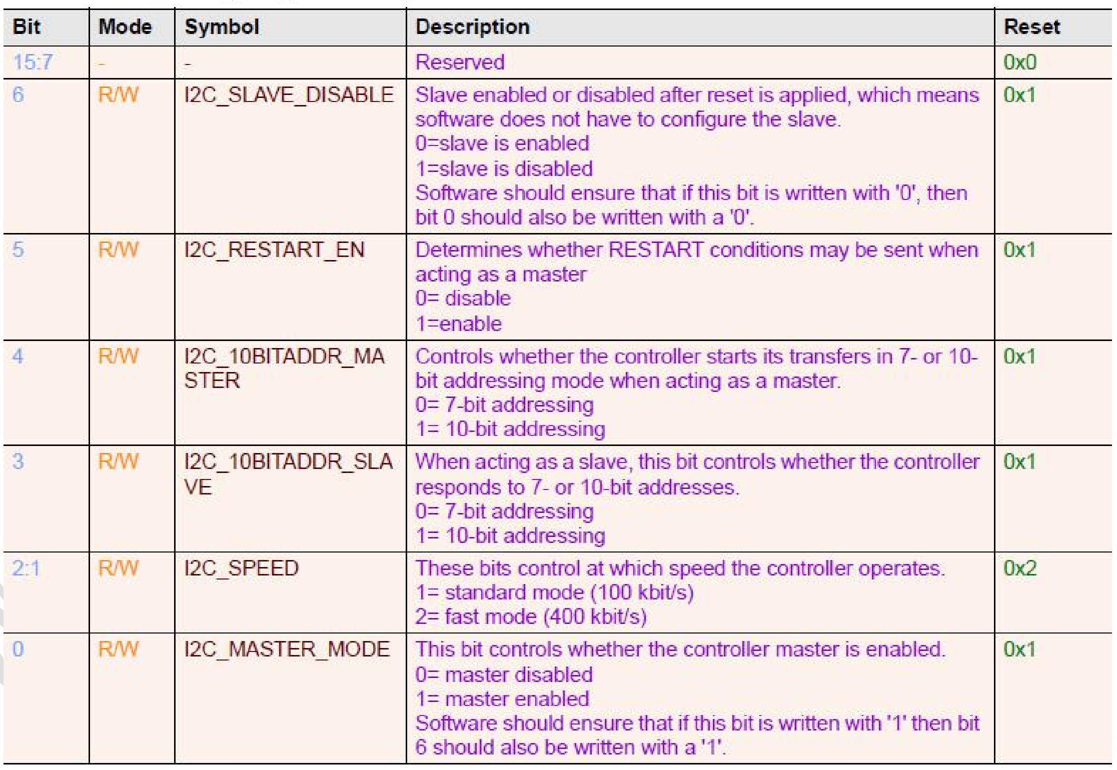
两线 I2C 串行接口包括一个串行数据线（SDA）和一个串行时钟线（SCL）；
支持两种通信速率，标准模式（0~100Kb/s）和快速模式（小于等于 400Kb/s）；
时钟同步；
32 字节的发送接收 FIFO；
主机发送与接收操作；
7 或 10 位地址，7 或 10 为混合格式发送；
块发送模式；
默认从地址为 0x055；
中断或者轮询操作模式；
可编程的数据线保持时间；

2.2 寄存器介绍

I2C 相关的寄存器比较多，所以我们只介绍常用的寄存器，其它的可以参考官方数据手册 AD14580_DS_v3.1.pdf，位于目录：..\WT 开发板\硬件资料。

2.2.1 I2C 控制寄存器

Table 166: I2C_CON_REG (0x50001300)



Bit	Mode	Symbol	Description	Reset
15:7	-	-	Reserved	0x0
6	R/W	I2C_SLAVE_DISABLE	Slave enabled or disabled after reset is applied, which means software does not have to configure the slave. 0=slave is enabled 1=slave is disabled Software should ensure that if this bit is written with '0', then bit 0 should also be written with a '0'.	0x1
5	R/W	I2C_RESTART_EN	Determines whether RESTART conditions may be sent when acting as a master 0= disable 1=enable	0x1
4	R/W	I2C_10BITADDR_MASTER	Controls whether the controller starts its transfers in 7- or 10-bit addressing mode when acting as a master. 0= 7-bit addressing 1= 10-bit addressing	0x1
3	R/W	I2C_10BITADDR_SLAVE	When acting as a slave, this bit controls whether the controller responds to 7- or 10-bit addresses. 0= 7-bit addressing 1= 10-bit addressing	0x1
2:1	R/W	I2C_SPEED	These bits control at which speed the controller operates. 1= standard mode (100 kbit/s) 2= fast mode (400 kbit/s)	0x2
0	R/W	I2C_MASTER_MODE	This bit controls whether the controller master is enabled. 0= master disabled 1= master enabled Software should ensure that if this bit is written with '1' then bit 6 should also be written with a '1'.	0x1

15:7 位：保留不使用；

6 位：I2C 从设备使能位，’0’表示从设备使能，’1’表示从设备不可用，该位不一定要软件设置，但是要保证如果该位为’0’则该寄存器的第 0 位也为’0’；

5 位：当作为主设备时，是否发送重启条件，’0’表示不可以，’1’表示可以；

4 位：作为主设备时，决定以 7 位地址还是 10 位地址开始发送，’0’表示 7 位地址，’1’表示 10 位地址；

3 位：作为从设备时，决定以 7 位地址还是 10 位地址开始发送，’0’表示 7 位地址，’1’表示 10 位地址；

2:1 位：I2C 通信速度选择，1 表示标准速度（100Kbit/s），2 表示快速（400Kbit/s）；

0 位：I2C 主设备使能，’0’表示主设备不可用，’1’表示主设备使能，要保证如果该位为’1’则该寄存器的第 6 位也为’1’；

2.2.2 I2C 目标地址寄存器

Table 167: I2C_TAR_REG (0x50001304)

Bit	Mode	Symbol	Description	Reset
15:12	-	-	Reserved	0x0
11	R/W	SPECIAL	This bit indicates whether software performs a General Call or START BYTE command. 0: ignore bit 10 GC_OR_START and use IC_TAR normally 1: perform special I2C command as specified in GC_OR_START bit	0x0
10	R/W	GC_OR_START	If bit 11 (SPECIAL) is set to 1, then this bit indicates whether a General Call or START byte command is to be performed by the controller. 0: General Call Address - after issuing a General Call, only writes may be performed. Attempting to issue a read command results in setting bit 6 (TX_ABRT) of the IC_RAW_INTR_STAT register. The controller remains in General Call mode until the SPECIAL bit value (bit 11) is cleared. 1: START BYTE	0x0
9:0	R/W	IC_TAR	This is the target address for any master transaction. When transmitting a General Call, these bits are ignored. To generate a START BYTE, the CPU needs to write only once into these bits. Note: If the IC_TAR and IC_SAR are the same, loopback exists but the FIFOs are shared between master and slave, so full loopback is not feasible. Only one direction loopback mode is supported (simplex), not duplex. A master cannot transmit to itself; it can transmit to only a slave	0x55

15:12 位：保留不使用；

11 位：该位决定软件是否进行广播或者开始字节命令，’0’表示忽略第 10 位 GC_OR_START 并且正常使用 IC_TAR；

10 位：如果第 11 位设置为’1’，则该位表示控制器是否进行广播或开始字节命令，’0’表示发送广播地址，之后只能进行写操作，如果进行读操作则导致 TX_ABRT 置位，控制器一直停留在广播模式，直到第 11 位被清除，’1’表示发送开始字节；

9:0 位：这是主设备发送的目标地址，如果发送广播则该位被忽略，CPU 只需要写一次这些位；注意如果目标地址与从设备地址相同则存在回路，但是 FIFO 为主从共用，所以完全回路是可行的，只支持单方向的回路，一个主设备不能给自己发送数据只能发送给从设备。

2.2.3 I2C 接收发送数据缓存与命令寄存器

Table 169: I2C_DATA_CMD_REG (0x50001310)

Bit	Mode	Symbol	Description	Reset
15:9	-	-	Reserved	0x0

8	R/W	CMD	This bit controls whether a read or a write is performed. This bit does not control the direction when the I2C Ctrl acts as a slave. It controls only the direction when it acts as a master. 1 = Read 0 = Write When a command is entered in the TX FIFO, this bit distinguishes the write and read commands. In slave-receiver mode, this bit is a "don't care" because writes to this register are not required. In slave-transmitter mode, a "0" indicates that CPU data is to be transmitted and as DAT or IC_DATA_CMD[7:0]. When programming this bit, you should remember the following: attempting to perform a read operation after a General Call command has been sent results in a TX_ABRT interrupt (bit 6 of the I2C_RAW_INTR_STAT_REG), unless bit 11 (SPECIAL) in the I2C_TAR register has been cleared. If a "1" is written to this bit after receiving a RD_REQ interrupt, then a TX_ABRT interrupt occurs. NOTE: It is possible that while attempting a master I2C read transfer on the controller, a RD_REQ interrupt may have occurred simultaneously due to a remote I2C master addressing the controller. In this type of scenario, it ignores the I2C_DATA_CMD write, generates a TX_ABRT interrupt, and waits to service the RD_REQ interrupt	0x0
7:0	R/W	DAT	This register contains the data to be transmitted or received on the I2C bus. If you are writing to this register and want to perform a read, bits 7:0 (DAT) are ignored by the controller. However, when you read this register, these bits return the value of data received on the controller's interface.	0x0

15:9 位：保留不使用；

8 位：读写控制位，作为从设备时不能控制方向，只能作为主设备时使用，'0'表示写，'1'表示读；

7:0 位：存储 I2C 总线上发送或接收的数据，如果你正在操作该寄存器并且要进行读操作则该位被忽略，如果你读该寄存器则该位存储的是接收到的数据。

2.2.4 I2C 清除 TX_ABRT 中断

Table 184: I2C_CLR_TX_ABRT_REG (0x50001354)

Bit	Mode	Symbol	Description	Reset
15:1	-	-	Reserved	0x0
0	R	CLR_TX_ABRT	Read this register to clear the TX_ABRT interrupt (bit 6) of the IC_RAW_INTR_STAT register, and the I2C_TX_ABRT_SOURCE register. This also releases the TX FIFO from the flushed/reset state, allowing more writes to the TX FIFO. Refer to Bit 9 of the I2C_TX_ABRT_SOURCE register for an exception to clearing IC_TX_ABRT_SOURCE.	0x0

15:1 位：保留不使用；

0 位：清除发送异常停止位，读该位则清除发送异常停止中断位，和发送异常停止源寄存器位。同时发送 FIFO 从刷新/复位状态中释放出来，可以允许更多写入。

2.2.5 I2C 使能寄存器

Table 190: I2C_ENABLE_REG (0x5000136C)

Bit	Mode	Symbol	Description	Reset
15:1	-	-	Reserved	0x0

0	R/W	CTRL_ENABLE	Controls whether the controller is enabled. 0: Disables the controller (TX and RX FIFOs are held in an erased state) 1: Enables the controller Software can disable the controller while it is active. However, it is important that care be taken to ensure that the controller is disabled properly. When the controller is disabled, the following occurs: * The TX FIFO and RX FIFO get flushed. * Status bits in the IC_INTR_STAT register are still active until the controller goes into IDLE state. If the module is transmitting, it stops as well as deletes the contents of the transmit buffer after the current transfer is complete. If the module is receiving, the controller stops the current transfer at the end of the current byte and does not acknowledge the transfer. There is a two ic_clk delay when enabling or disabling the controller	0x0

15:1 位：保留不使用；

0 位：控制器使能位；

2.2.6 I2C 状态寄存器

Table 191: I2C_STATUS_REG (0x50001370)

Bit	Mode	Symbol	Description	Reset
15:7	-	-	Reserved	0x0
6	R	SLV_ACTIVITY	Slave FSM Activity Status. When the Slave Finite State Machine (FSM) is not in the IDLE state, this bit is set. 0: Slave FSM is in IDLE state so the Slave part of the controller is not Active 1: Slave FSM is not in IDLE state so the Slave part of the controller is Active	0x0
5	R	MST_ACTIVITY	Master FSM Activity Status. When the Master Finite State Machine (FSM) is not in the IDLE state, this bit is set. 0: Master FSM is in IDLE state so the Master part of the controller is not Active 1: Master FSM is not in IDLE state so the Master part of the controller is Active	0x0
4	R	RFF	Receive FIFO Completely Full. When the receive FIFO is completely full, this bit is set. When the receive FIFO contains one or more empty location, this bit is cleared. 0: Receive FIFO is not full 1: Receive FIFO is full	0x0
3	R	RFNE	Receive FIFO Not Empty. This bit is set when the receive FIFO contains one or more entries; it is cleared when the receive FIFO is empty. 0: Receive FIFO is empty 1: Receive FIFO is not empty	0x0
2	R	TFE	Transmit FIFO Completely Empty. When the transmit FIFO is completely empty, this bit is set. When it contains one or more valid entries, this bit is cleared. This bit field does not request an interrupt. 0: Transmit FIFO is not empty 1: Transmit FIFO is empty	0x1
1	R	TFNF	Transmit FIFO Not Full. Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full. 0: Transmit FIFO is full 1: Transmit FIFO is not full	0x1
0	R	I2C_ACTIVITY	I2C Activity Status.	0x0

15:7 位：保留不使用；

6 位：判断从设备是否活动；

5 位：判断主设备是否活动；

4 位：判断接收 FIFO 是否全满；

3 位：判断接收 FIFO 是否为空；

2 位：判断发送 FIFO 是否全满；
1 位：判断发送 FIFO 是否为空；
0 位：判断 I2C 模块是否活动。

2.2.7 I2C 接收 FIFO 数目寄存器

Table 193: I2C_RXFLR_REG (0x50001378)

Bit	Mode	Symbol	Description	Reset
15:6	-	-	Reserved	0x0
5:0	R	RXFLR	Receive FIFO Level. Contains the number of valid data entries in the receive FIFO. Size is constrained by the RXFLR value	0x0

15:6 位：保留不使用；
5:0 位：接收 FIFO 可以接收多少字节。

2.2.8 I2C 发送异常终止源寄存器

Table 195: I2C_TX_ABRT_SOURCE_REG (0x50001380)

Bit	Mode	Symbol	Description	Reset
15	R	ABRT_SLVRD_INTX	1: When the processor side responds to a slave mode request for data to be transmitted to a remote master and user writes a 1 in CMD (bit 8) of 2IC_DATA_CMD register	0x0
14	R	ABRT_SLV_ARBLOST	1: Slave lost the bus while transmitting data to a remote master. I2C_TX_ABRT_SOURCE[12] is set at the same time. Note: Even though the slave never "owns" the bus, something could go wrong on the bus. This is a fail safe check. For instance, during a data transmission at the low-to-high transition of SCL, if what is on the data bus is not what is supposed to be transmitted, then the controller no longer own the bus.	0x0
13	R	ABRT_SLVFLUSH_TXFIFO	1: Slave has received a read command and some data exists in the TX FIFO so the slave issues a TX_ABRT interrupt to flush old data in TX FIFO.	0x0
12	R	ARB_LOST	1: Master has lost arbitration, or if I2C_TX_ABRT_SOURCE[14] is also set, then the slave transmitter has lost arbitration. Note: I2C can be both master and slave at the same time.	0x0
11	R	ABRT_MASTER_DIS	1: User tries to initiate a Master operation with the Master mode disabled.	0x0
10	R	ABRT_10B_RD_NORSTRT	1: The restart is disabled (IC_RESTART_EN bit (I2C_CON[5]) = 0) and the master sends a read command in 10-bit addressing mode.	0x0

9	R	ABRT_SBYTE_NORSTRT	To clear Bit 9, the source of the ABRT_SBYTE_NORSTRT must be fixed first; restart must be enabled (I2C_CON[5]=1), the SPECIAL bit must be cleared (I2C_TAR[11]), or the GC_OR_START bit must be cleared (I2C_TAR[10]). Once the source of the ABRT_SBYTE_NORSTRT is fixed, then this bit can be cleared in the same manner as other bits in this register. If the source of the ABRT_SBYTE_NORSTRT is not fixed before attempting to clear this bit, bit 9 clears for one cycle and then gets re-asserted. 1: The restart is disabled (IC_RESTART_EN bit (I2C_CON[5]) = 0) and the user is trying to send a START Byte.	0x0
8	R	ABRT_HS_NORSTRT	1: The restart is disabled (IC_RESTART_EN bit (I2C_CON[5]) = 0) and the user is trying to use the master to transfer data in High Speed mode	0x0
7	R	ABRT_SBYTE_ACKDET	1: Master has sent a START Byte and the START Byte was acknowledged (wrong behavior).	0x0
6	R	ABRT_HS_ACKDET	1: Master is in High Speed mode and the High Speed Master code was acknowledged (wrong behavior).	0x0
5	R	ABRT_GCALL_READ	1: the controller in master mode sent a General Call but the user programmed the byte following the General Call to be a read from the bus (IC_DATA_CMD[9] is set to 1).	0x0
4	R	ABRT_GCALL_NOACK	1: the controller in master mode sent a General Call and no slave on the bus acknowledged the General Call.	0x0
3	R	ABRT_TXDATA_NOACK	1: This is a master-mode only bit. Master has received an acknowledgement for the address, but when it sent data byte(s) following the address, it did not receive an acknowledge from the remote slave(s).	0x0
2	R	ABRT_10ADDR2_NOACK	1: Master is in 10-bit address mode and the second address byte of the 10-bit address was not acknowledged by any slave.	0x0
1	R	ABRT_10ADDR1_NOACK	1: Master is in 10-bit address mode and the first 10-bit address byte was not acknowledged by any slave.	0x0
0	R	ABRT_7B_ADDR_NOACK	1: Master is in 7-bit addressing mode and the address sent was not acknowledged by any slave.	0x0

15 位：当主设备需要发送数据时，却进入读数据状态；
14 位：当发送数据时，从设备丢失总线；
13 位：当从设备要接收数据时，FIFO 中已经有一些数据；
12 位：失去仲裁；
11 位：当主设备不可用时，用户进行主设备的操作；
10 位：重启不可用，并且主设备在 10 位地址模式下发送读命令；
9 位：重启不可用，但是用户发送一个开始字节；
8 位：重启不可用，但是用户试图在高速模式下发送数据；
7 位：主设备已经发送了一个开始字节，并且开始字节被确认；
6 位：主设备在高速模式下，并且被确认；
5 位：主设备控制器广播之后进行读操作；
4 位：主设备发送广播，但是没有从设备确认；
3 位：只有主设备有效，主设备已经发送地址，并确认，但是发送数据得不到确认信号；
2 位：主设备使用 10 位地址模式，10 位地址的第二个字节没有被任何从设备确认；
1 位：主设备使用 10 位地址模式，10 位地址的第一个字节没有被任何从设备确认；
0 位：主设备使用 7 位地址模式，但是没有被任何从设备确认。

2.3 寄存器配置讲解

```
#define CLK_PER_REG                    (* ( volatile uint16*)0x50000004)
```

```
#define I2C_CON_REG                    (* ( volatile uint16*)0x50001300)
#define I2C_TAR_REG                    (* ( volatile uint16*)0x50001304)
#define I2C_DATA_CMD_REG               (* ( volatile uint16*)0x50001310)
#define I2C_CLR_TX_ABRT_REG            (* ( volatile uint16*)0x50001354)
#define I2C_ENABLE_REG                 (* ( volatile uint16*)0x5000136C)
#define I2C_STATUS_REG                 (* ( volatile uint16*)0x50001370)
#define I2C_RXFLR_REG                  (* ( volatile uint16*)0x50001378)
#define I2C_TX_ABRT_SOURCE_REG         (* ( volatile uint16*)0x50001380)
```

启动 I2C 模块的时钟：CLK_PER_REG |= 0x0020;

I2C 的初始化寄存器配置：

先关闭 I2C 控制器， I2C_ENABLE_REG=0x00；

设置为主模式，关闭从模式，可以重复开始，速度设置为快速，地址为 7 位模式(0x0000000001100101)， I2C_ CON _REG =0x0065；

设置目标设备地址为 0x51， I2C_TAR_REG =0x51；

打开 I2C 控制器， I2C_ENABLE_REG=0x01；

等待控制器准备好，while((I2C_STATUS_REG & 0x20) != 0);

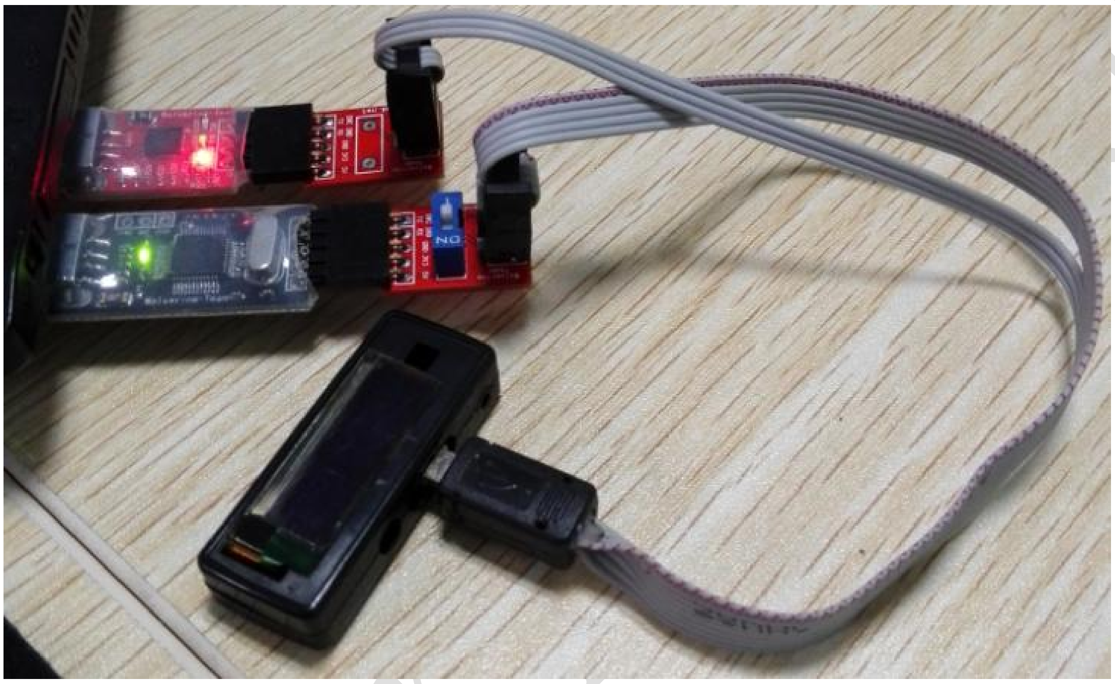
读取地址为 0x98 处的一个字节，先发送地址 I2C_DATA_CMD_REG = 0x98;等待发送完毕 while((I2C_STATUS_REG&0x0002)==0);发送读指令 I2C_DATA_CMD_REG = 0x0100; 等待发送完毕 while((I2C_STATUS_REG&0x0004)==0);之后等待数据接收完毕 while(I2C_RXFLR_REG == 0);读取接收缓冲区的数据即为接收数据 rx_data = I2C_DATA_CMD_REG;

向地址为 0x98 处写入一个字节 0xaa,先发送地址 I2C_DATA_CMD_REG = 0x98;等待发送完毕 while((I2C_STATUS_REG&0x0002)==0);发送数据 I2C_DATA_CMD_REG = 0xaa; 等待发送完毕 while((I2C_STATUS_REG&0x0004)==0);

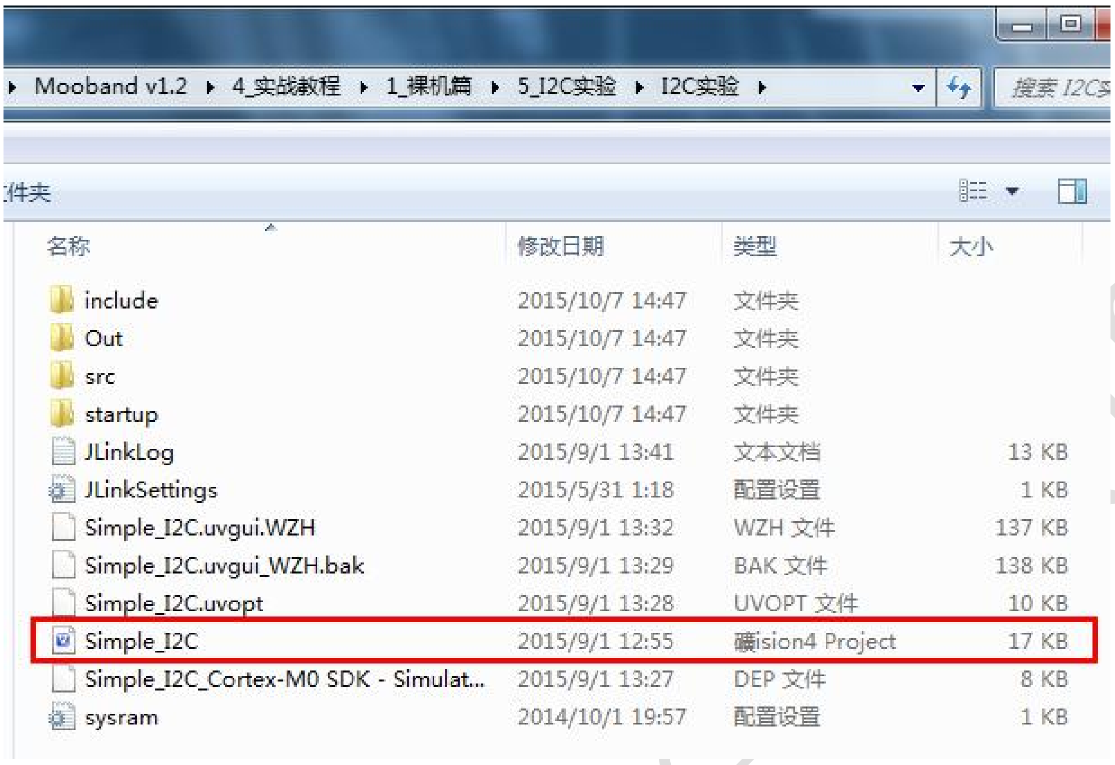
第三节 I2C 实验

实验需要使用的模块有：带屏手环，Jlink 调试工具，USB 转串模块，一根手环下载调试线。

将 JLINK 通过下载调试线连接到手环的 USB 调试接口，JLINK 插在有拨码开关的一端，注意丝印标注一一对应，将 JLINK 插上电脑的 USB 口。将 USB 转串模块插在手环现在调试线的另一端，注意丝印标注一一对应，然后将 USB 转串模块插在电脑的 USB 接口。如下图所示：



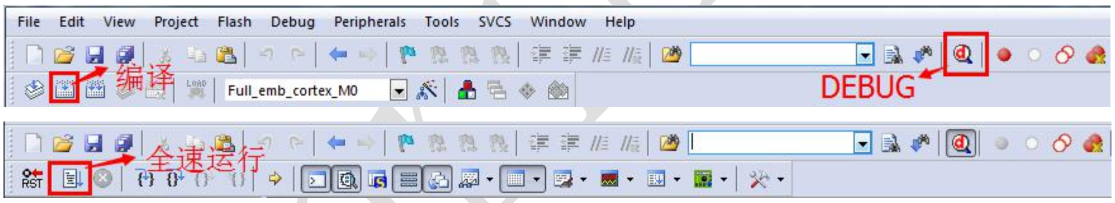
打开 I2C 实验的 Keil 工程 Simple_I2C.uvproj，位于目录：..\ 4_实战教程\1_裸机篇\5_I2C 实验\I2C 实验，如下图所示：

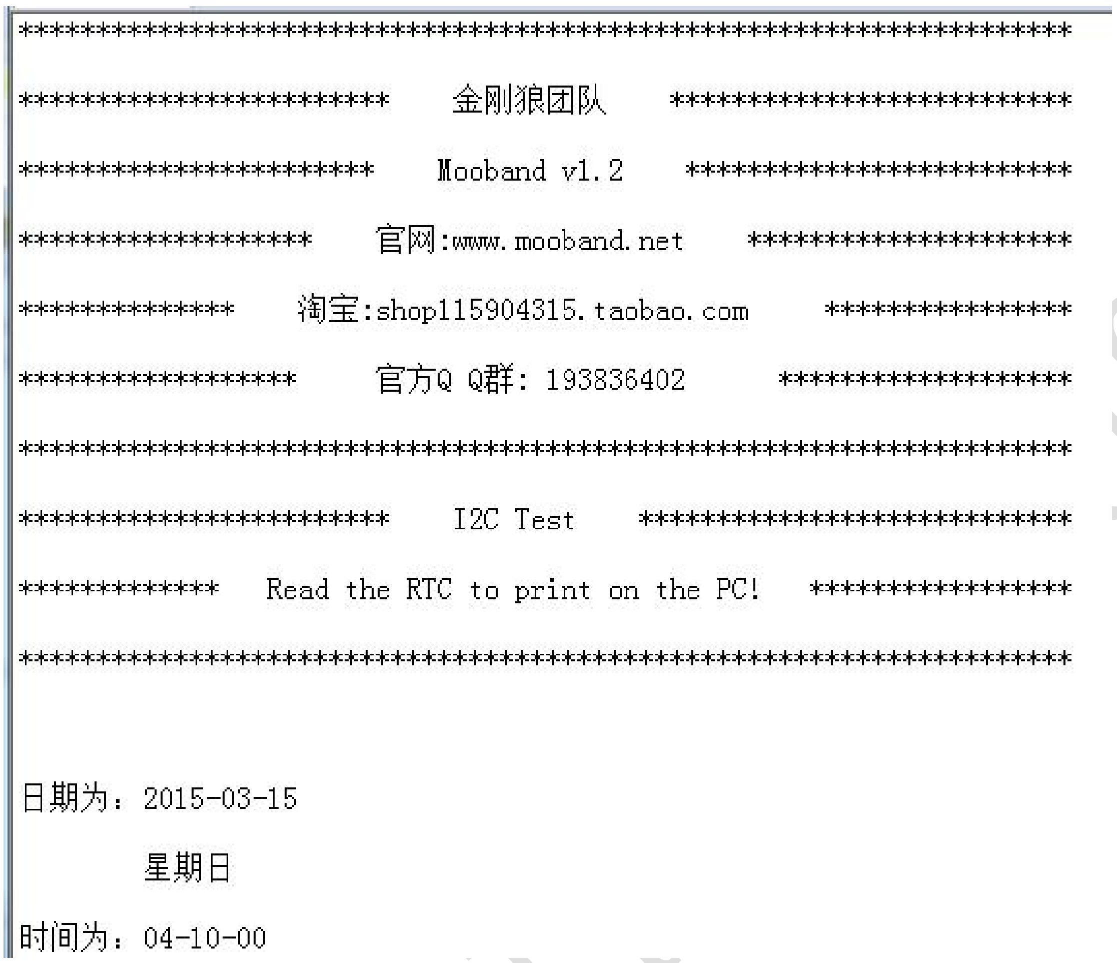


▸ Mooband v1.2 ▸ 4_实战教程 ▸ 1_裸机篇 ▸ 5_I2C实验 ▸ I2C实验 ▸

名称	修改日期	类型	大小
include	2015/10/7 14:47	文件夹	
Out	2015/10/7 14:47	文件夹	
src	2015/10/7 14:47	文件夹	
startup	2015/10/7 14:47	文件夹	
JLinkLog	2015/9/1 13:41	文本文档	13 KB
JLinkSettings	2015/5/31 1:18	配置设置	1 KB
Simple_I2C.uvgui.WZH	2015/9/1 13:32	WZH 文件	137 KB
Simple_I2C.uvgui_WZH.bak	2015/9/1 13:29	BAK 文件	138 KB
Simple_I2C.uvopt	2015/9/1 13:28	UVOPT 文件	10 KB
Simple_I2C	2015/9/1 12:55	礦ision4 Project	17 KB
Simple_I2C_Cortex-M0 SDK - Simulat...	2015/9/1 13:27	DEP 文件	8 KB
sysram	2014/10/1 19:57	配置设置	1 KB

打开串口调试助手连接串口，波特率为 115200。打开 KEIL 工程之后，编译代码，点击 DEBUG，然后点击全速运行，就可以看到串口调试助手打印出的信息，如下图所示：


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Full_emb_cortex_M0
编译
DEBUG
全速运行



```
**********************************************************************
************************     金刚狼团队     ***************************
***********************     Mooband v1.2     *************************
*******************     官网:www.mooband.net     *********************
**************     淘宝:shop115904315.taobao.com     ****************
******************     官方Q Q群: 193836402     *********************
**********************************************************************
************************     I2C Test     ****************************
*************     Read the RTC to print on the PC!     ****************
**********************************************************************


日期为：2015-03-15
        星期日
时间为：04-10-00
```