

## 第 12 章 龙芯 1B 的 I2C

### 12.1 I2C 接口

本章将介绍 I2C 的详细描述和配置使用。LS1B0200 芯片集成了 I2C 接口，主要用于实现两个器件之间数据的交换。I2C 总线是由数据线 SDA 和时钟 SCL 构成的串行总线，可发送和接收数据。器件与器件之间进行双向传送，最高传送速率 400kbps。LS1B0200 芯片共集成 3 路 I2C 接口，其中第二路和第三路分别通过 CAN0 和 CAN1 复用实现，如下表 12-1 所示；复用配置参加 23 章 MUX 寄存器小节。

表 12-1 复用关系

I2C_SDA1	CAN0_RX
I2C_SCL1	CAN0_TX
I2C_SDA2	CAN1_RX
I2C_SCL2	CAN1_TX

#### 12.1.2 I2C 控制器结构

I2C 主控制器的结构，主要模块有，时钟发生器（Clock Generator）、字节命令控制器（Byte Command Controller）、位命令控制器（Bit Command controller）、数据移位寄存器（Data Shift Register）。其余为 LPB 总线接口和一些寄存器。这些模块之间的关系，如图 12-1 所示。

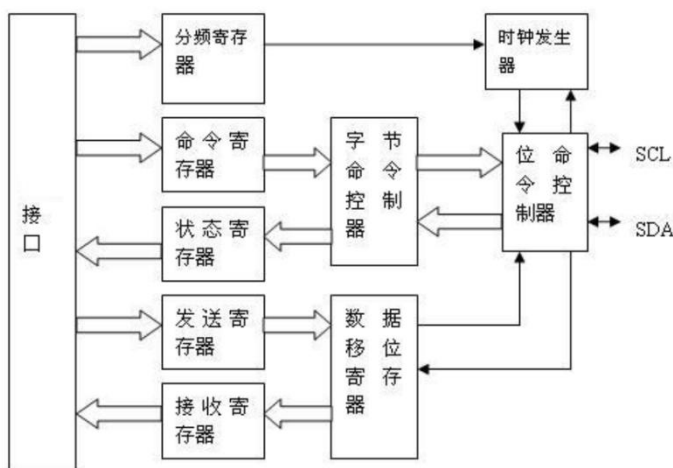


图 12-1 I2C 主控制器结构

1. **时钟发生器模块：** 产生分频时钟，同步位命令的工作。
2. **字节命令控制器模块：** 将一个命令解释为按字节操作的时序，即把字节操作分解为位操作。
3. **位命令控制器模块：** 进行实际数据的传输，以及位命令信号产生。
4. **数据移位寄存器模块：** 串行数据移位。

#### 12.1.3 I2C 控制寄存器

LS1B0200 共集成 3 路 I2C 接口，其功能寄存器完全一样，只是访问基址不一样，I2C 控制器寄存器基地址如表 12-2 所示。

表 12-2 UART 控制器寄存器基地址

名称	基地址（Base）	地址空间
----	-----------	------

I2C0	0xbfe58000	16KB
I2C1	0xbfe68000	16KB
I2C2	0xbfe70000	16KB

I2C 控制器寄存器如表 12-3 所示,3 个 I2C 控制器寄存器基本上相同。每个 I2C 有一个基地址（Base），描述如表 12-3 所示。

表 12-3 UART 控制器寄存器描述

分频锁存器低字节寄存器（PRERlo）（偏移地址：0x00；寄存器位宽：8 位；复位值：0xff）			
位域	位域名称	访问	说明
7:0	PRERlo	R/W	存放分频锁存器的低 8 位
分频锁存器高字节寄存器（PRERhi）（偏移地址：0x01；寄存器位宽：8 位；复位值：0xff）			
位域	位域名称	访问	说明
7:0	PRERhi	R/W	存放分频锁存器的高 8 位
控制寄存器（CTR）（偏移地址：0x02；寄存器位宽：8 位；复位值：0x00）			
位域	位域名称	访问	说明
7	EN	R/W	模块工作使能位为 1 正常工作模式, 0 对分频寄存器进行操作
6	IEN	R/W	中断使能位 为 1 则打开中断
5:0	Reserved	R/W	保留
发送数据寄存器（TXR）（偏移地址：0x03；寄存器位宽：8 位；复位值：0x00）			
位域	位域名称	访问	说明
7:1	DATA	W	存放下个将要发送的字节
0	DRW	W	当数据传送时，该位保存的是数据的最低位； 当地址传送时，该位指示读写状态
接受数据寄存器（RXR）（偏移地址：0x03；寄存器位宽：8 位；复位值：0x00）			
位域	位域名称	访问	说明
7:0	RXR	R	存放最后一个接收到的字节
命令控制寄存器（CR）（偏移地址：0x04；寄存器位宽：8 位；复位值：0x00）			
位域	位域名称	访问	说明
7	STA	W	产生 START 信号
6	STO	W	产生 STOP 信号
5	RD	W	产生读信号
4	WR	W	产生写信号
3	ACK	W	产生应答信号
2:1	Reserved	W	保留
0	IACK	W	产生中断应答信号
状态寄存器（SR）（偏移地址：0x04；寄存器位宽：8 位；复位值：0x00）			
位域	位域名称	访问	说明
7	RxACK	R	收到应答位 1 没收到应答位 0 收到应答位
6	Busy	R	I2C 总线忙标志位 1 总线在忙 0 总线空闲
5	AL	R	当 I2C 核失去 I2C 总线控制权时，该位置 1

4:2	Reserved	R	保留
1	TIP	R	指示传输的过程 1 表示正在传输数据 0 表示数据传输完毕
0	IF	R	中断标志位，一个数据传输完，或另外一个器件 发起数据传输，该位置 1

（1）通信速率的配置：模块中被分频时钟 `clock_a` 的频率是 `DDR_clk` 频率的一半（`DDR_clk` 配置见 22 章）；假设分频锁存器的值为 `prescale`，`SCL` 总线的输出频率为 `clock_s`（该时钟根据用户需要和外部 `I2C` 设备特性确定），则应满足如下关系：

$$\text{Prcescale} = \text{clock\_a} / (5 * \text{clock\_s}) - 1$$

$$\text{或者 } \text{Prcescale} = \text{DDR\_clk} / (10 * \text{clock\_s}) - 1$$

（2）控制寄存器：`bit7` 为模块工作使能位；改位为 1 时正常工作模式，为 0 时对分屏寄存器进行操作。

（3）发送数据寄存器：`[bit7:bit1]` 存放下个将要发送的字节；当数据传送时，`bit0` 位为保存的是数据的最低位，当地址传送时，`bit0` 位为指示读写状态

（4）命令控制寄存器：在 `I2C` 发送数据后硬件自动清零。对这些位读操作时候总是读回 '0'。

（5）在使用 `I2C` 总线通信时需要配置通信速率。

`I2C` 控制器寄存器配置的相关代码，如代码清单 12-1 所示。

代码清单 12-1 `I2C` 控制器寄存器配置的相关代码

---

```

1.  /***** 以下代码在 ls1x_i2c_bus.c 文件中 *****/
2.  /* 初始化与 BSP 总线频率 */
3.  pIIC->base_freq = LS1x_BUS_FREQUENCY(CPU_XTAL_FREQUENCY);
4.
5.  /* 设置分频器，默认为 100kHz */
6.  fdr_val = baudrate < pIIC->baudrate ? baudrate : pIIC->baudrate;
7.  fdr_val = fdr_val > 0 ? fdr_val : 100000;
8.  fdr_val = pIIC->base_freq / (5 * fdr_val) - 1;
9.
10. /* 设置控制寄存器 */
11. ctrl = pIIC->hwI2C->ctrl;
12. ctrl &= ~(i2c_ctrl_en | i2c_ctrl_ien);          //对分频寄存器进行操作，关闭中断
13. pIIC->hwI2C->ctrl = ctrl;
14. pIIC->hwI2C->prerlo = fdr_val & 0xFF;          //设置分频锁存器低字节
15. pIIC->hwI2C->prerhi = (fdr_val >> 8) & 0xFF;    //设置分频锁存器高字节
16.
17. /* 设置命令控制器，正常工作模式 */
18. pIIC->hwI2C->cmd_sr.cmd = 0x00;                //命令状态寄存器值写为 0，不产生任何信号
19. ctrl |= i2c_ctrl_en;                          //设置为正常工作模式
20. pIIC->hwI2C->ctrl = ctrl;

```

---

## 12.2 I2C API 函数分析

### 12.2.1 I2C 驱动函数

I2C 驱动源代码在 ls1x-drv/i2c/ls1x\_i2c\_bus.c 中，头文件在 ls1x - drv/include/ls1x\_i2c\_bus.h 中，配置代码在 include/bsp.h 中。

#### 1. 启用 I2C 设备

需要用到哪个 I2C 设备，只需要在 bsp.h 中反注释宏定义，如代码清单 12-2 所示。

代码清单 12-2 启用 I2C 宏义

---

```
1.  #define BSP_USE_I2C0
2.  // #define BSP_USE_I2C1
3.  // #define BSP_USE_I2C2
```

---

#### 2. I2C 设备参数定义

在 ls1x\_i2c\_bus.c 文件中，对 I2C 设备配置参数进行结构体封装，如代码清单 12-3 所示。

代码清单 12-3 I2C 设备配置参数结构体

---

```
1.  typedef struct
2.  {
3.      struct LS1x_I2C_regs *hwI2C;    /* 指针指向 I2C 硬件寄存器 */
4.      unsigned int base_freq;         /* 总线频率变量 */
5.      unsigned int baudrate;         /* 通信速率 */
6.      unsigned int dummy_char;       /* 空字符 */
7.      /* interrupt support */
8.      unsigned int irqNum;            /* 中断向量编号 */
9.      unsigned int int_ctrlr;        /* 中断寄存器基地址 (INT0 的地址) */
10.     unsigned int int_mask;          /* 中断屏蔽位 */
11.     /* mutex */                     /* 带有系统 */
12. #if defined(OS_RTTHREAD)
13.     rt_mutex_t i2c_mutex;
14. #elif defined(OS_UCOS)
15.     OS_EVENT *i2c_mutex;
16. #elif defined(OS_FREERTOS)
17.     SemaphoreHandle_t i2c_mutex;
18. #else // defined(OS_NONE)
19.     int i2c_mutex;
20. #endif
21.     int initialized;                /* 是否初始化 */
22.     char dev_name[16];              /* 设备名称 */
23. #if (PACK_DRV_OPS)
24.     libi2c_ops_t *ops;              /* 指针指向总线操作函数 */
25. #endif
26. } LS1x_I2C_bus_t;
```

---

以下 I2C0 为例，介绍如何填充 I2C 设备配置参数结构体，代码在 ls1x\_i2c\_bus.c 文件中，

如代码清单 12-4 所示。定义结构体变量的同时初始化成员变量，填充变量的不同点是 I2C 寄存器的基地址不同，其它基本相同。

代码清单 12-4 I2C0 设备配置参数结构体填充

---

```
1.  /* I2C0 */
2.  static LS1x_I2C_bus_t ls1x_I2C0 =
3.  {
4.  .hwI2C = (struct LS1x_I2C_regs *)LS1x_I2C0_BASE, /*设备地址*/
5.  .base_freq = 0,                                /*总线频率*/
6.  .baudrate = 100000,                             /*通信速率*/
7.  .dummy_char = 0,                                /*空字符*/
8.  .i2c_mutex = 0,                                 /*设备锁*/
9.  .initialized = 0,                                /*是否初始化*/
10. .dev_name = "i2c0",                             /*设备名称*/
11. };
12. LS1x_I2C_bus_t *busI2C0 = &ls1x_I2C0;
13. #endif
```

---

### 3. 驱动函数与数据结构

在 ls1x\_io.h、ls1x\_i2c\_bus.c 和 ls1x\_i2c\_bus.h 文件中，定义了设备驱动的函数原型、驱动函数的数据结构等，如代码清单 12-5 所示。

代码清单 12-5 驱动函数与数据结构代码

---

```
1.  //用于设备驱动的函数原型（适用于所有外设）。ls1x_io.h 文件中定义
1.  /* 用于 SPI/I2C 总线驱动的函数原型 */
2.  typedef int (*I2C_init_t)(void *bus);
3.  typedef int (*I2C_send_start_t)(void *bus, unsigned Addr);
4.  typedef int (*I2C_send_stop_t)(void *bus, unsigned Addr);
5.  typedef int (*I2C_send_addr_t)(void *bus, unsigned Addr, int rw);
6.  typedef int (*I2C_read_bytes_t)(void *bus, unsigned char *bytes, int nbytes);
7.  typedef int (*I2C_write_bytes_t)(void *bus, unsigned char *bytes, int nbytes);
8.  typedef int (*I2C_ioctl_t)(void *bus, int cmd, void *arg);
9.
10. #if (PACK_DRV_OPS)
11. typedef struct libi2c_ops//定义 7 个函数原型接口，适用于所有外设
12. {
13.     I2C_init_t      init;
14.     I2C_send_start_t send_start;
15.     I2C_send_stop_t send_stop;
16.     I2C_send_addr_t send_addr;
17.     I2C_read_bytes_t read_bytes;
18.     I2C_write_bytes_t write_bytes;
19.     I2C_ioctl_t     ioctl;
20. } libi2c_ops_t;
```

---

```
21.
22. typedef libi2c_ops_t    libspi_ops_t;
23. #endif
24. //*****
25. //填充 driver_ops_t 结构体成员，在 ls1x_i2c_bus.c 文件中定义
26. #if (PACK_DRV_OPS)
27. static libi2c_ops_t LS1x_I2C_ops =
28. {
29.     .init          = LS1x_I2C_initialize,
30.     .send_start    = LS1x_I2C_send_start,
31.     .send_stop     = LS1x_I2C_send_stop,
32.     .send_addr     = LS1x_I2C_send_addr,
33.     .read_bytes    = LS1x_I2C_read_bytes,
34.     .write_bytes   = LS1x_I2C_write_bytes,
35.     .ioctl         = LS1x_I2C_ioctl,
36. };
37. LS1x_I2C_bus_t *busI2C0 = &ls1x_I2C0;
38. #endif
```

12.2.2 I2C 接口函数

有 7 个用户操作的 I2C 接口函数，与驱动函数一一对应，如表 12-4 所示。

表 12-4 用户 I2C 接口函数与驱动函数一一对应

用户接口函数	对应的驱动函数	功能描述
ls1x_i2c_initialize(i2c)	LS1x_I2C_initialize(void *bus);	初始化
ls1x_i2c_send_start(i2c, addr)	LS1x_I2C_send_start(void *bus, unsigned int Addr);	获取总线控制权
ls1x_i2c_send_stop(i2c, addr)	LS1x_I2C_send_stop(void *bus, unsigned int Addr);	释放总线控制权
ls1x_i2c_send_addr(i2c, addr,rw)	LS1x_I2C_send_addr(void *bus, unsigned int Addr, int rw);	发送片选信号
ls1x_i2c_read_bytes(i2c, buf, len)	LS1x_I2C_read_bytes(void *bus, unsigned char *buf, int len);	读数据
ls1x_i2c_write_bytes(i2c, buf, len)	LS1x_I2C_write_bytes(void *bus, unsigned char *buf, int len);	写数据
ls1x_i2c_ioctl(i2c, cmd, arg)	LS1x_I2C_ioctl(void *bus, int cmd, void *arg);	发送控制命令

用户接口函数最终是调用驱动函数，用户接口函数代码如代码清单 12-6 所示。

代码清单 12-6 用户接口函数代码

```
1. //用于设备驱动的函数类型。ls1x_i2c_bus.h 文件中定义
2. #if (PACK_DRV_OPS)
3.     #define ls1x_i2c_initialize(i2c)          i2c->ops->init(i2c)
```

```

4.     #define lslx_i2c_send_start(i2c, addr)      i2c->ops->send_start(i2c, addr)
5.     #define lslx_i2c_send_stop(i2c, addr)       i2c->ops->send_stop(i2c, addr)
6.     #define lslx_i2c_send_addr(i2c, addr, rw)   i2c->ops->send_addr(i2c, addr, rw)
7.     #define lslx_i2c_read_bytes(i2c, buf, len)  i2c->ops->read_bytes(i2c, buf, len)
8.     #define lslx_i2c_write_bytes(i2c, buf, len) i2c->ops->write_bytes(i2c, buf, len)
9.     #define lslx_i2c_ioctl(i2c, cmd, arg)       i2c->ops->ioctl(i2c, cmd, arg)
10.    #else
11.    #define lslx_i2c_initialize(i2c)              LS1x_I2C_initialize(i2c)
12.    #define lslx_i2c_send_start(i2c, addr)       LS1x_I2C_send_start(i2c, addr)
13.    #define lslx_i2c_send_stop(i2c, addr)        LS1x_I2C_send_stop(i2c, addr)
14.    #define lslx_i2c_send_addr(i2c, addr, rw)    LS1x_I2C_send_addr(i2c, addr, rw)
15.    #define lslx_i2c_read_bytes(i2c, buf, len)   LS1x_I2C_read_bytes(i2c, buf, len)
16.    #define lslx_i2c_write_bytes(i2c, buf, len)  LS1x_I2C_write_bytes(i2c, buf, len)
17.    #define lslx_i2c_ioctl(i2c, cmd, arg)        LS1x_I2C_ioctl(i2c, cmd, arg)
18.    #endif

```

---

### 1. lslx\_i2c\_initialize(i2c)函数分析

(1) 参数与返回值

①i2c: i2c 设备, 3 个 i2c 结构体变量指针供选择:

busI2C0、busI2C1、busI2C2

②返回值: 错误: -1; 正常: 0。

(2) 函数功能: 初始化 I2C, 根据总线频率设置分频系数, 从而设置通信速率

### 2. lslx\_i2c\_send\_start(i2c, addr) 函数分析

(1) 参数与返回值

①i2c: i2c 设备, 3 个 i2c 结构体变量指针供选择。

②addr: 未使用, 写入 0 即可

③返回值: 错误: -1; 正常: 0。

(2) 函数功能: 产生开始信号, 获取总线控制权

### 3. lslx\_i2c\_send\_stop(i2c, addr)函数分析

(1) 参数与返回值

①i2c: i2c 设备, 3 个 i2c 结构体变量指针供选择。

②addr: 未使用, 写入 0 即可。

③返回值: 错误: -1; 正常: 0。

(2) 函数功能: 产生停止信号, 释放总线控制权

### 4. lslx\_i2c\_send\_addr(i2c, addr, rw)函数分析

(1) 参数与返回值

①i2c: i2c 设备, 3 个 i2c 结构体变量指针供选择。

②addr: 七位从设备地址

③rw: 读写方向位, 1:读, 0:写。

③返回值: 错误: -1; 正常: 0。

(2) 函数功能: 发送从设备地址和读写方向位

### 5. lslx\_i2c\_read\_bytes(i2c, buf, len)函数分析

(1) 参数与返回值

①i2c: i2c 设备, 3 个 i2c 结构体变量指针供选择。

- ②buf: 存放读取到的数据。
- ③len: 读取的数据的长度（单位：字节）。
- ④返回值: 成功: 返回读取到的字节数。失败: -1
- (2) 函数功能: 从接受数据寄存器中读取数据

6. ls1x\_i2c\_write\_bytes(i2c, buf, len)函数分析

- (1) 参数与返回值
- ①i2c: i2c 设备，3 个 i2c 结构体变量指针供选择。
- ②buf: 写入的数据。
- ③len: 写入的数据的长度（单位：字节）。
- ④返回值: 成功: 返回写入成功的字节数。失败: -1
- (2) 函数功能: 将数据写入发送数据寄存器中

7. ls1x\_i2c\_iocctl(i2c, cmd, arg)函数分析

- (1) 参数与返回值
- ①i2c: i2c 设备，3 个 i2c 结构体变量指针供选择。
- ②cmd: 命令数据。
- ③arg: 需要传入的参数。
- ④返回值: 成功: 0。失败: -1
- ⑤说明: 此处只做了设置速率的命令，arg 参数则为要设置的 IIC 总线速率
- (2) 函数功能: 发送控制命令

12.3 I2C 开发步骤

- 第一步: 添加 ls1x\_i2c\_bus.h 头文件，并在 bsp.h 中打开使用到的 I2C 设备
- 第二步: 调用 ls1x\_i2c\_initialize()函数初始化函数; 注: 如果使用 I2C1 和 I2C2 是需要通过 CAN0 和 CAN1 复用实现。
- 第三步: 编写读写数据函数，如下 12-5 表所示

表 12-5 配置 I2C 读写步骤

	写数据	读数据
第三步: 发送起始信号	ls1x_i2c_send_start()	ls1x_i2c_send_start()
第四步: 发送从机地址和写命令	ls1x_i2c_send_addr(i2c, addr, w)	ls1x_i2c_send_addr(i2c, addr, w)
第五步: 发送从机寄存器的地址	ls1x_i2c_write_bytes(i2c, buf, len)	ls1x_i2c_write_bytes(i2c, buf, len)
第六步		发送停止信号: ls1x_i2c_send_stop()
第七步	发送数据: ls1x_i2c_write_bytes(i2c, buf, len)	发送起始信号: ls1x_i2c_send_start()
第八步	发送停止信号: ls1x_i2c_send_stop();	发 送 从 机 地 址 和 读 写 命 令:ls1x_i2c_send_addr(i2c, addr, r)
第九步		读 取 数 据 : ls1x_i2c_read_bytes((i2c, buf, len);
第十步		发送停止信号: ls1x_i2c_send_stop()

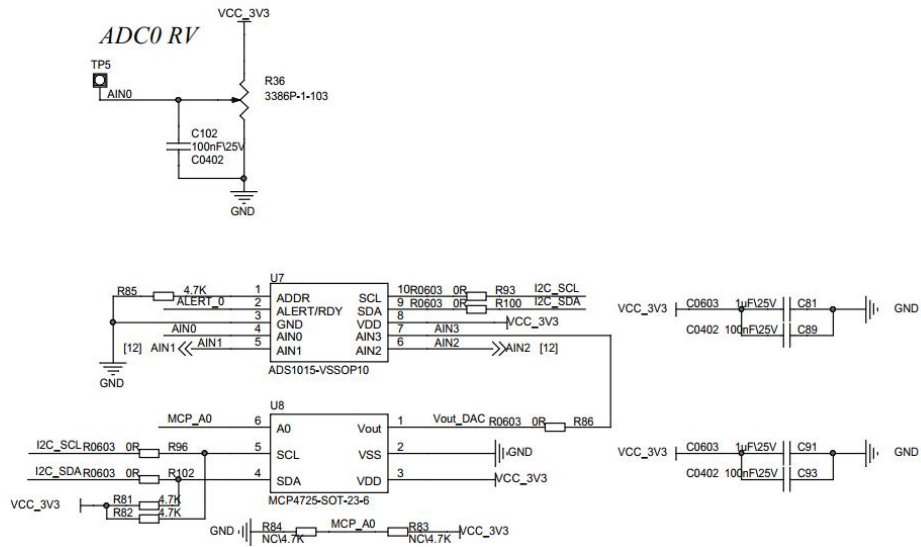
任务 5 读写 I2C 设备

(一) 任务描述

下面的从机器件全部搭载的是 I2C0 总线上



# ADC/DAC



## (二) 任务分析

### 1. 硬件电路分析

数模-模数转换芯片 (MCP4725, ADS1015) 共 6 片从机设备; 使用的总线全部是 I2C0 总线。

### 2. 软件设计-->以数模-模数转换 (MCP4725, ADS1015) 为例

首先, 按照新建项目导向, 建立工程, 在 bsp.h 头文件中打开 I2C0 的宏定义。

其次, 添加 ls1x\_i2c\_bus.h 的头文件, 调用 ls1x\_i2c\_initialize() 函数初始化 I2C0  
随后, 编写设备读写函数; 加载 ads1015.h 和 mcp4725.h 的头文件 (注: 这两个头文件在 ls1x - drv/include/i2c)

最后: 调用编写好的读写函数, 进行主从机数据传输。

注: 由于 IDE 中已经写好了数模-模数转换芯片的用户读写接口函数, 所以我们只需加载 ads1015.h 和 mcp4725.h 头文件, 直接调用这里的读写接口函数。

## (三) 任务实施

### 第 1 步: 硬件连接。

先用 USB 转串口线连接电脑 USB 和龙芯 1+X 口袋机的串口 (UART5), 再给龙芯 1+X 口袋机上电。

### 第 2 步: 新建工程。

首先, 打开“龙芯 1x 嵌入式集成开发环境”, 按照“文件->新建->新建项目向导...”方式点击, 新建工程。

### 第 3 步: 编写程序。

在自动生成代码的基础上, 编写代码, 并在 bsp.h 文件中打开 I2C0 宏, 如图代码清单 12-7 所示。

代码清单 12-7 I2C0(数模-模数转换) 程序

```
1. #include <stdio.h>
2.
3. #include "ls1b.h"
```



```

4. #include "mips.h"
5.
6. #include "bsp.h"
7. #include "ls1x_i2c_bus.h"
8. #include "i2c/ads1015.h"
9. #include "i2c/mcp4725.h"
10.
11. int main(void)
12. {
13.     printk("\r\nmain() function.\r\n");
14.
15.     //内存堆初始化
16.     lwmem_initialize(0);
17.     //初始化 I2C0 控制器
18.     ls1x_i2c_initialize(busI2C0);
19.     /* 获取设备的配置状态 */
20.     ls1x_mcp4725_ioc1(busI2C0, IOCTL_MCP4725_DISP_CONFIG_REG, NULL);
21.     ls1x_ads1015_ioc1(busI2C0, IOCTL_ADS1015_DISP_CONFIG_REG, NULL);
22.     printk("\n");
23.     char tbuf[40]={0}, sbuf[40]={0}, rt;
24.     unsigned short dac=0, adc=0;
25.
26.     /*
27.      * 裸机主循环
28.      */
29.     while(1) {
30.         sprintf((char *)tbuf, "MCP4725_write_adc: dac = %d", dac);
31.         /* 将数字量转换为模拟量 */
32.         set_mcp4725_dac(busI2C0, dac);
33.         dac += 5;
34.         if(dac > 4096)
35.             dac = 0;
36.         /* 将模拟量转换为数字量, 采用 ADS1015 的第三通道*/
37.         adc = get_ads1015_adc(busI2C0, ADS1015_REG_CONFIG_MUX_SINGLE_3);
38.         sprintf((char *)sbuf, "ADS1015_get_adc: adc = %d", adc);
39.
40.         printk("%s\n%s\n\n", tbuf, sbuf);
41.         delay_ms(500);
42.     }
43.     return 0;
44. }

```

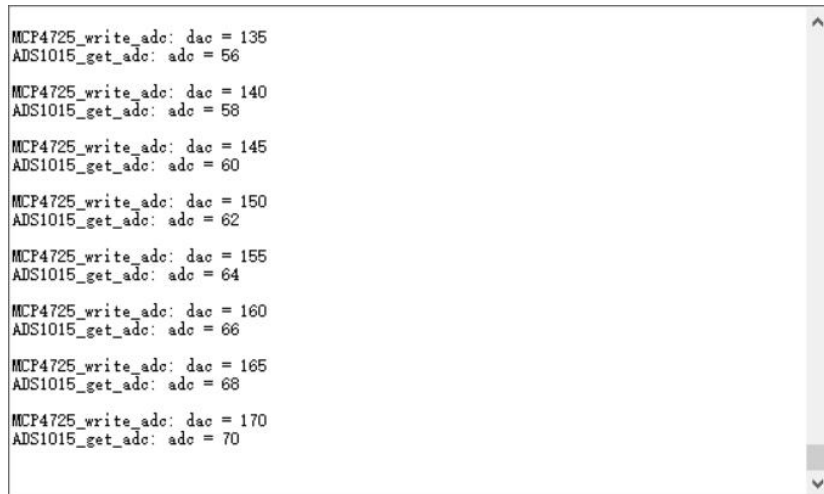
---

#### 第 4 步：程序编译及调试。

(1) 点击【】图标进行编译，编译无误后，点击【】图标，将程序下载到内存之

中。注意：此时代码没有下载到 **nand flash** 之中，按下复位键后，程序会消失。

（2）打开串口调试软件，如图 12-3 所示配置串口参数后，则在串口调试软件上打印相关信息。

A screenshot of a serial terminal window with a white background and a vertical scrollbar on the right. The text is black and shows a sequence of data points. Each point consists of two lines: the first line starts with 'MCP4725\_write\_adc:' followed by 'dac = ' and a value; the second line starts with 'ADS1015\_get\_adc:' followed by 'adc = ' and a value. The values for 'dac' range from 135 to 170 in increments of 5. The corresponding 'adc' values range from 56 to 70 in increments of 2. There are 8 such pairs of lines in the image.

```
MCP4725_write_adc: dac = 135
ADS1015_get_adc: adc = 56

MCP4725_write_adc: dac = 140
ADS1015_get_adc: adc = 58

MCP4725_write_adc: dac = 145
ADS1015_get_adc: adc = 60

MCP4725_write_adc: dac = 150
ADS1015_get_adc: adc = 62

MCP4725_write_adc: dac = 155
ADS1015_get_adc: adc = 64

MCP4725_write_adc: dac = 160
ADS1015_get_adc: adc = 66

MCP4725_write_adc: dac = 165
ADS1015_get_adc: adc = 68

MCP4725_write_adc: dac = 170
ADS1015_get_adc: adc = 70
```

图 12-3 串口调试软件上打印相关信息

（3）现象：可以看到 ADS1015 输出的值随 MCP4725 输入的值增大而增大，也就是说输入的数字量越大，经过 ADS1015 转换后输出的数字量也越大。

#### （四）任务拓展

请查看本书配套资源，了解拓展任务要求和程序代码。