嵌入式系统设计

实验指导

北方工业大学信息工程学院

通信工程系、通信实验中心

2011-09-10

实验 1　设计入门

一、实验目的

1、理解嵌入式系统的组成和设计方法；
2、理解嵌入式系统软件的设计步骤和调试方法；
3、理解定时器和并行接口的原理和使用；
4、熟悉嵌入式系统的开发环境。

二、实验内容

设计一个简单的嵌入式系统，包括 CPU、存储器、定时器、2 个按键接口和 4 个 LED 接口，实现按键控制 LED 的流水显示。

1、4 个 LED 流水显示，每 1s 移位 1 次；
2、2 个按键控制 4 个 LED 的流水显示方向；
3、用中断方式实现定时处理。（选做）

系统硬件原理框图和软件流程图如图 1.1 和 1.2 所示。

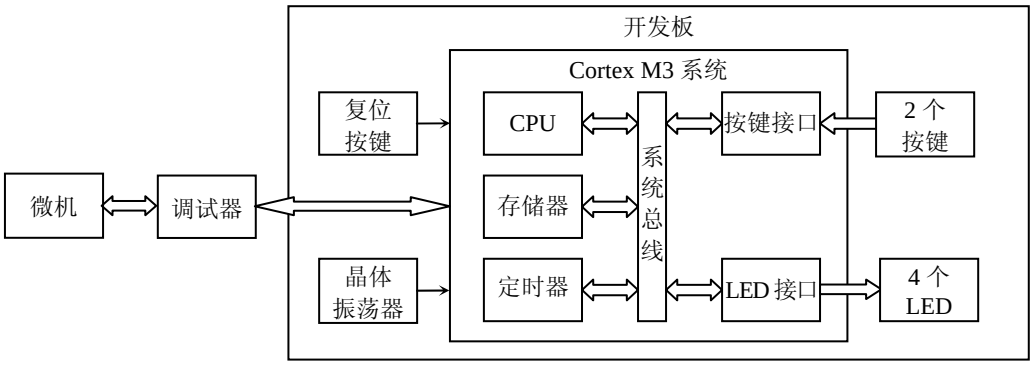


图 1.1　系统硬件原理框图

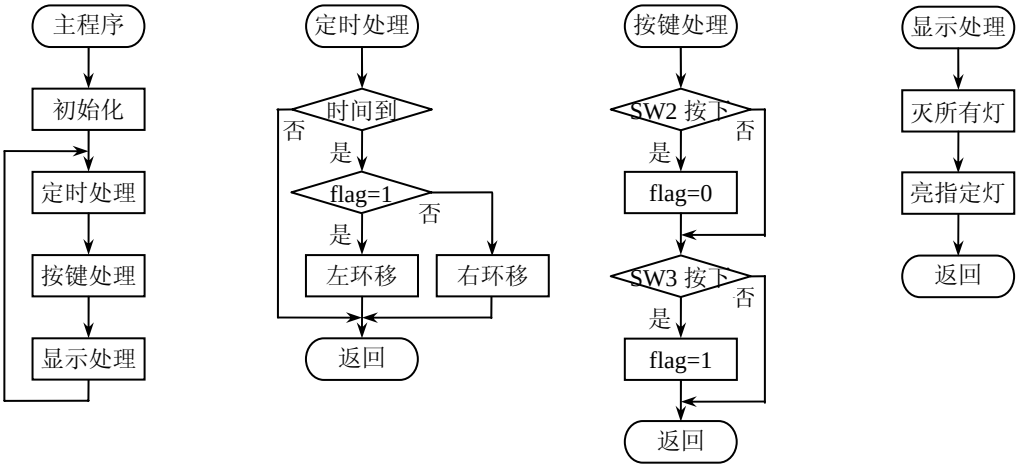


图 1.2　系统软件流程图

三、实验基础

1、系统滴答定时器（SysTick）

SysTick 的核心是 1 个 24 位递减计数器，使用时根据需要设置初值（STRVR），启动（ENABLE=1）后在系统时钟（HCLK 或 HCLK/8）的作用下递减，减到 0 时置计数标志位（COUNTFLAG）并重装初值。系统可以查询计数标志位，也可以在中断允许（TICKINT=1）时产生系统滴答中断。SysTick 的结构图如图 1.3 所示。

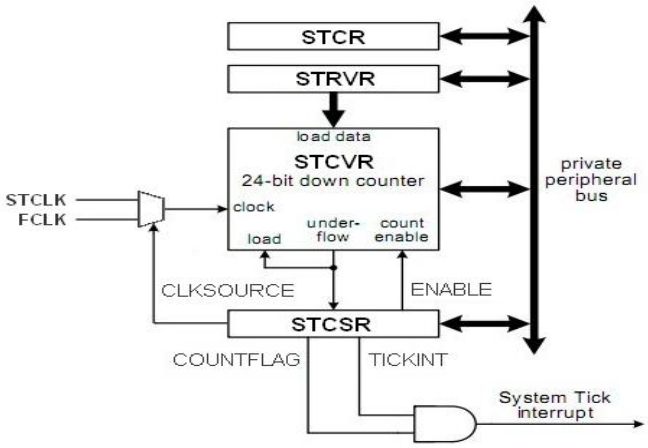


图 1.3　SysTick 结构图

SysTick 通过 4 个 32 位寄存器进行操作，如表 1.1 所示。

表 1.1　SysTick 寄存器

地址	名称	类型	复位值	说　　明
0xE000E010	STCSR	读/写	0	控制状态寄存器
0xE000E014	STRVR	读/写	-	24 位重装值寄存器，计数到 0 时重装到 STCVR
0xE000E018	STCVR	读/写清除	-	24 位当前值寄存器，写清除，同时清除 COUNTFLAG
0xE000E01C	STCR	读	-	校准寄存器

控制状态寄存器（STCSR）有 3 个控制位和 1 个状态位，如表 1.2 所示。

表 1.2　控制状态寄存器

位	名称	类型	复位值	说　　明
0	ENABLE	读/写	0	启动定时器
1	TICKINT	读/写	0	中断允许：0-计数到 0 时不中断，1-计数到 0 时中断
2	CLKSOURCE	读/写	0	时钟源选择：0-时钟 HCLK/8，1-时钟 HCLK
16	COUNTFLAG	读	0	计数标志：SysTick 计数到 0 时置 1，读取后自动清零

SysTick 的编程操作方法有 2 种：直接操作寄存器和使用库函数，实验使用直接操作寄存器，直接操作寄存器分为 3 步：寄存器定义、初始化和读写操作。

寄存器定义的方法为：

```
#define NVIC_STCSR (*(volatile unsigned long *) (0xE000E010))
```

寄存器初始化和读写操作的方法为：

```
NVIC_STCSR |= 1;                // 启动定时器
if(NVIC_STCSR & 0x10000)        // 定时时间到
```

2、外设时钟控制

为了降低功耗，外设的时钟是可控的，初始化外设时必须允许外设时钟，否则外设不能工作。外设时钟通过外设时钟允许寄存器（RCC_APB2ENR）进行控制，RCC_APB2ENR 寄存器的内容如表 1.3 所示（地址为 0x40021018）。

表 1.3　RCC_APB2ENR 寄存器

位	名称	类型	复位值	说　　明
0	AFIOEN	读/写	0	AFIO 时钟允许：0-禁止，1-允许
2	GPIOAEN	读/写	0	GPIOA 时钟允许：0-禁止，1-允许
3	GPIOBEN	读/写	0	GPIOB 时钟允许：0-禁止，1-允许
4	GPIOCEN	读/写	0	GPIOC 时钟允许：0-禁止，1-允许
5	GPIODEN	读/写	0	GPIOD 时钟允许：0-禁止，1-允许
6	GPIOEEN	读/写	0	GPIOE 时钟允许：0-禁止，1-允许
7	GPIOFEN	读/写	0	GPIOF 时钟允许：0-禁止，1-允许
8	GPIOGEN	读/写	0	GPIOG 时钟允许：0-禁止，1-允许
9	ADC1EN	读/写	0	ADC1 时钟允许：0-禁止，1-允许
10	ADC2EN	读/写	0	ADC2 时钟允许：0-禁止，1-允许
11	TIM1EN	读/写	0	TIM1 定时器时钟允许：0-禁止，1-允许
12	SPI1EN	读/写	0	SPI1 时钟允许：0-禁止，1-允许
13	TIM8EN	读/写	0	TIM8 定时器时钟允许：0-禁止，1-允许
14	USART1EN	读/写	0	USART1 时钟允许：0-禁止，1-允许
15	ADC3EN	读/写	0	ADC3 时钟允许：0-禁止，1-允许

3、通用并行接口（GPIO）

GPIO 包括多个 16 位 I/O 端口，每个端口可以独立设置 4 种输入方式和 4 种输出方式，并可独立地置位或复位。标准 I/O 端口的基本结构如图 1.4 所示。

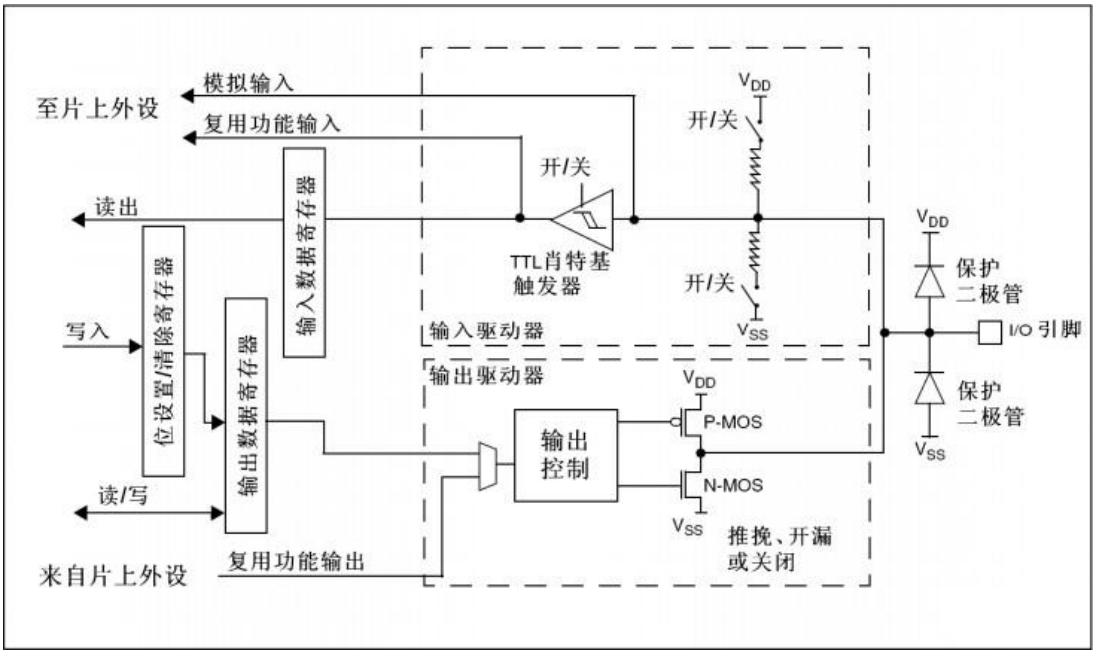


图 1.4　标准 I/O 端口基本结构图

系统通过 7 个 32 位寄存器控制 GPIO，如表 1.4 所示（GPIOA-GPIOD 的基地址依次为 0x40010800、0x40010C00、0x40011000 和 0x40011400）。

表 1.4　GPIO 寄存器

偏移地址	名称	类型	复位值	说　　明
0x00	CRL	读/写	0x44444444	配置寄存器低位，P7-0，每个端口对应 4 位
0x04	CRH	读/写	0x44444444	配置寄存器高位，P15-8，每个端口对应 4 位
0x08	IDR	读	-	16 位输入数据寄存器
0x0C	ODR	读/写	0x0000	16 位输出数据寄存器
0x10	BSRR	写	0x00000000	位置位/复位寄存器：高 16 位复位，低 16 位置位 0-不影响，1-ODR 对应位置位/复位
0x14	BRR	写	0x0000	位复位寄存器，和 BSRR 的高 16 位功能相同 0-不影响，1-ODR 对应位复位
0x18	LCKR	读/写	0x00000	配置锁定寄存器

配置寄存器（CRL 和 CRH）中每个端口对应的 4 个配置位是 CNF[1:0]和 MODE[1:0]，端口配置如表 1.5 所示。

表 1.5　端口配置

CNF[1:0]	输入配置	输出配置	MODE[1:0]	配置模式
00	模拟输入	通用推挽输出	00	输入(复位状态)
01	悬浮输入(复位状态)	通用开漏输出	01	输出 10MHz
10	上拉/下拉输入	复用推挽输出	10	输出 2MHz
11	保留	复用开漏输出	11	输出 50MHz

4、实验参考程序

直接操作寄存器的实验参考程序如下：

```
// 定时器寄存器地址定义
#define NVIC_STCSR  (*(volatile unsigned long *) (0xE000E010))
#define NVIC_STRVR  (*(volatile unsigned long *) (0xE000E014))
#define NVIC_STCVR  (*(volatile unsigned long *) (0xE000E018))
// 外设时钟允许寄存器地址定义
#define RCC_APB2ENA (*(volatile unsigned long *) (0x40021018))
// GPIOB寄存器地址定义
#define GPIOB_IDR   (*(volatile unsigned long *) (0x40010C08))
// GPIOC寄存器地址定义
#define GPIOC_CRL   (*(volatile unsigned long *) (0x40011000))
#define GPIOC_CRH   (*(volatile unsigned long *) (0x40011004))
#define GPIOC_ODR   (*(volatile unsigned long *) (0x4001100C))
#define GPIOC_BSRR  (*(volatile unsigned long *) (0x40011010))
#define GPIOC_BRR   (*(volatile unsigned long *) (0x40011014))
// 函数声明
void SysTick_Init(void);
void SysTick_Handler(void);
```

```
void GpioB_Init(void);
void Key_Proc(void);
void GpioC_Init(void);
void Led_Proc(void);
// 全局变量声明
int flag = 1, led = 0x200;
// 主函数
int main(void)
{
  SysTick_Init();              // 系统定时器初始化
  GpioB_Init();                // 按键接口初始化
  GpioC_Init();                // LED 接口初始化
  while(1)
  {
    SysTick_Handler();         // 定时处理
    Key_Proc();                // 按键处理
    Led_Proc();                // LED 显示处理
  }
}
// 系统定时器初始化
void SysTick_Init(void)
{
  NVIC_STRVR = 0x7a1200;       // 1s 定时值
  NVIC_STCSR = 5;              // 时钟 HCLK，禁止中断，启动定时器
}
// 定时处理
void SysTick_Handler(void)
{
  if(NVIC_STCSR & 0x10000)     // 定时时间到
  {
    if(flag)
    {
      led <<= 1;               // 左环移
      if(led == 0x400)  led = 0x40;
    }
    else
    {
      led >>= 1;               // 右环移
      if(led == 0x20)  led = 0x200;
    }
  }
}
```

```
// 按键接口初始化
void GpioB_Init(void)
{
  RCC_APB2ENA |= 8;              // 允许 GPIOB 时钟
}
// 按键处理
void Key_Proc(void)
{
  int key;
  key = ~GPIOB_IDR;              // 读键值
  if(key & 0x8000)               // SW2 按下
    flag = 0;
  if(key & 0x4000)               // SW3 按下
    flag = 1;
}
// LED 接口初始化
void GpioC_Init(void)
{
  RCC_APB2ENA |= 0x0010;         // 允许 GPIOC 时钟
  GPIOC_CRL |= 0x55000000;       // PC7-6 通用开漏输出
  GPIOC_CRH |= 0x00000055;       // PC9-8 通用开漏输出
  GPIOC_ODR |= 0x03C0;           // 灭所有灯
}
// LED 显示处理
void Led_Proc(void)
{
//GPIOC_ODR |= 0x03C0;           // 灭所有灯
//GPIOC_ODR &= ~led;             // 亮指定灯
  GPIOC_BSRR = 0x03C0;           // 灭所有灯
  GPIOC_BRR = led;               // 亮指定灯
}
```

四、实验报告

1、实验目的；（5 分）
2、实验内容；（5 分）
3、硬件方框图和电路图；（30 分）
4、软件流程图和核心语句；（30 分）
5、设计过程中遇到的问题和解决方法；（20 分）
6、收获和建议等。（10 分）

实验 2　液晶屏实验

一、实验目的

1、掌握并行接口实现串行数据输出的方法；
2、掌握液晶屏的工作原理和使用方法。

二、实验内容

在实验 1 硬件设计的基础上增加液晶屏接口，在软件设计中增加相关的程序：
1、用定时器实现分秒计时；
2、用液晶屏实现分秒显示。

三、实验基础

1、液晶显示模块简介

开发板上使用的 LCM046 液晶显示模块是 4 位多功能通用型 8 段(8.8.8.8)液晶显示模块，内含看门狗(WDT)定时器，2 种频率的蜂鸣驱动电路，内置显示 RAM，可显示任意字段笔划，3-4 线串行接口，工作电压 2.4～5.2V，如图 2.1 所示。

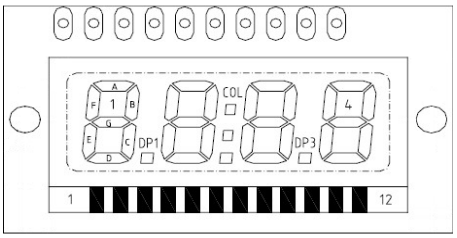


图 2.1　LCM046 液晶显示模块

显示模块的引脚说明如表 2.1 所示。

表 2.1　显示模块引脚说明

引脚	功能	方向	说　明	引脚	功能	方向	说　明
1	/CS	输入	片选，内部上拉	2	/RD	输入	读，内部上拉
3	/WR	输入	写，内部上拉	4	DA	双向	数据，内部上拉
5	GND	-	地	6	VLCD	输入	对比度调整
7	VDD	输入	电源	8	IRQ	输出	定时器
9	BZ	输出	蜂鸣器正极	10	/BZ	输入	蜂鸣器负极

显示模块的写操作时序如图 2.2 所示。

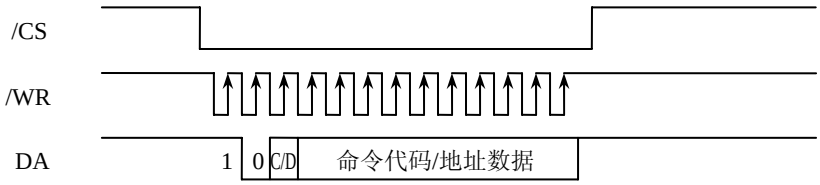


图 2.2　显示模块写操作时序

写命令格式为：100 命令代码（C7-C0：8 位）0，共 12 位

写数据格式为：101 地址（A5-A0：6 位）数据（D0-D3：4 位），共 13 位

101 地址（A5-A0：6 位）数据（n*4 位）（n=1-8），共 13-41 位

常用命令代码如表 2.2 所示，地址数据如表 2.3 所示。

表 2.2　显示模块常用命令代码

命令代码	功　能	命令代码	功　能
0000 0000	关振荡器	0000 0001	开振荡器
0000 0010	关显示器	0000 0011	开显示器
0001 0100	定义外部晶体振荡器工作	0001 1000	定义内部 RC 振荡器工作
0010 1001	模块初始化		

表 2.3　显示模块地址数据

地　址	数　　据				地　址	数　　据			
A5-A0	D3	D2	D1	D0	A5-A0	D3	D2	D1	D0
000000	COL	1E	1F	1A	000001	1D	1C	1G	1B
000010	DP1	2E	2F	2A	000011	2D	2C	2G	2B
000100	DP2	3E	3F	3A	000101	3D	3C	3G	3B
000110	DP3	4E	4F	4A	000111	4D	4C	4G	4B

模块上电后软件应延时200ms以上再初始化模块，初始化模块的步骤是：

（1）写模块初始化命令　　100 0010 1001

（2）定义内部RC振荡器工作 100 0001 1000

（3）开振荡器　　100 0000 0001

（4）开显示器　　100 0000 0011

为了实现低功耗工作，每次读写操作后应将/CS、/RD、/WR 和 DA 置高电平。

2、液晶显示模块程序设计

液晶显示模块程序设计的基础是写子程序：

```
// LCD写子程序
// 入口参数：data-写数据，bit-数据位数
void Lcd_Write(int data, int bit)
{
  int m, n;
  GPIOC_BRR = 0x1000;          // LCD_CS = 0
  for(m=bit-1; m>=0; m--)
  {
    for(n=0; n<10; n++);       // 延时
    GPIOC_BRR = 0x0800;        // LCD_WR = 0
    for(n=0; n<10; n++);
    if(data & (1<<m))
      GPIOC_BSRR = 0x0400;     // LCD_DA = 1
    else
      GPIOC_BRR = 0x0400;      // LCD_DA = 0
```

```
    for(n=0; n<10; n++);
    GPIOC_BSRR = 0x0800;        // LCD_WR = 1
  }
  GPIOC_BSRR = 0x1400;          // LCD_CS = 1, LCD_DA = 1
}
```

LCD初始化子程序如下：

```
// LCD初始化
void Lcd_Init(void)
{
  RCC_APB2ENA |= 0x10;          // 允许GPIOC时钟
  GPIOC_CRH |= 0x00055500;      // PC12-10通用开漏输出
  GPIOC_ODR |= 0x1C00;          // PC12-10输出高电平

  Lcd_Write(0x42a<<1, 12);      // 模块初始化
  Lcd_Write(0x418<<1, 12);      // 内部振荡器
  Lcd_Write(0x401<<1, 12);      // 开振荡器
  Lcd_Write(0x403<<1, 12);      // 开显示器
}
```

LCD显示处理子程序如下：

```
// LCD显示处理
// 入口参数：*data-数据指针
void Lcd_Proc(char *data)
{
  char lcd_code[16]=            // 显示编码
  {                             // AFE.BGCD
    0xeb,                       // 11101011 0
    0x0a,                       // 00001010 1
    0xad,                       // 10101101 2
    0x8f,                       // 10001111 3
    0x4e,                       // 01001110 4
    0xc7,                       // 11000111 5
    0xe7,                       // 11100111 6
    0x8a,                       // 10001010 7
    0xef,                       // 11101111 8
    0xcf,                       // 11001111 9
    0xee,                       // 11101110 A
    0x67,                       // 01100111 B
    0xe1,                       // 11100001 C
    0x2f,                       // 00101111 D
    0xe5,                       // 11100101 E
    0xe4                        // 11100100 F
  };
```

```
  Lcd_Write((5<<10)+(0<<4)+(lcd_code[data[0]]>>4)+data[1], 13);
  Lcd_Write((5<<10)+(1<<4)+(lcd_code[data[0]]&0xf), 13);
  Lcd_Write((5<<10)+(2<<4)+(lcd_code[data[2]]>>4)+data[3], 13);
  Lcd_Write((5<<10)+(3<<4)+(lcd_code[data[2]]&0xf), 13);
  Lcd_Write((5<<10)+(4<<4)+(lcd_code[data[4]]>>4)+data[5], 13);
  Lcd_Write((5<<10)+(5<<4)+(lcd_code[data[4]]&0xf), 13);
  Lcd_Write((5<<10)+(6<<4)+(lcd_code[data[6]]>>4)+data[7], 13);
  Lcd_Write((5<<10)+(7<<4)+(lcd_code[data[6]]&0xf), 13);
}
```

LCD显示处理子程序的调用方法如下：

```
// 定义显示数据数组      COL   DP1   DP2   DP3
char lcd_data[8] = {1, 0, 2, 0, 3, 0, 4, 0};
Lcd_Proc(lcd_data);            // LCD显示处理
```

计时程序段如下：

```
if((++sec & 0xf) == 0xa)       // 2-10进制调整
  sec += 6;
if(sec == 0x60)                // 1分钟时间到
{
  sec = 0;
  if((++min & 0xf) == 0xa)     // 2-10进制调整
  min += 6;
  if(min == 0x60)              // 1小时时间到
    min = 0;
}
```

将时分转换成显示数据的程序段如下：

```
lcd_data[0] = min >> 4;        // 分十位
lcd_data[2] = min & 0xf;       // 分个位
lcd_data[4] = sec >> 4;        // 秒十位
lcd_data[6] = sec & 0xf;       // 秒个位
lcd_data[1] = sec & 1;         // 分闪烁
```

四、实验报告

1、实验目的；（5 分）
2、实验内容；（5 分）
3、硬件方框图和电路图；（30 分）
4、软件流程图和核心语句；（30 分）
5、设计过程中遇到的问题和解决方法；（20 分）
6、收获和建议等。（10 分）

实验 3　SPI 实验

一、实验目的

1、掌握 SPI 接口的基本原理和编程方法；

2、掌握用 SPI 接口控制液晶屏的方法。

二、实验内容

在实验 1 硬件设计的基础上增加 SPI 接口，将 SPI 接口与液晶屏相连，如表 3.1 所示。

表 3.1　SPI 接口与液晶屏连接表

SPI 引脚	功　能	方向	电路板引脚	液晶屏引脚	功　能	方向
PA4-NSS	从设备选择	输出	JP11-6	1-/CS	片选	输入
PA5-SCK	串口时钟	输出	JP11-7	3-/WR	写	输入
PA6-MISO	主入从出	输入	JP11-8			
PA7-MOSI	主出从入	输出	JP11-9	4-DA	数据	双向

特别注意：由于液晶屏在电路板上已经和 PC10-12 相连，为了保证电路的安全，PC10-12 必须使用悬浮输入方式（复位状态）或开漏输出方式（输出 1）！

1、在软件设计中增加相关的初始化程序和分秒显示程序；

2、断开 NSS 和 SCK 与液晶屏的连接，将 MOSI 和 MISO 相连，编程将 MOSI 输出的数据通过 MISO 输入并送 LCD 显示。

三、实验基础

1、SPI 接口简介

SPI（Serial peripheral interface：串行设备接口）是工业标准串行协议，通常用于嵌入式系统，将微处理器连接到各种片外传感器、转换器、存储器和控制设备。

SPI 可以实现主设备或从设备协议，当配置为主设备时，SPI 可以连接多达 16 个独立的从设备，发送数据和接收数据寄存器的宽度可配置为 8 位或 16 位。

SPI 使用 2 根数据线、1 根时钟线和 1 根控制线实现串行通信。

① 主入从出（MISO）：主设备输入数据，从设备输出数据；

② 主出从入（MOSI）：主设备输出数据，从设备输入数据；

③ 串行时钟（SCK）：主设备输出，从设备输入，用于同步数据位；

④ 从设备选择（NSS）：主设备到从设备的选择信号（低电平有效）。

2、SPI 接口设计

SPI 的方框图如图 3.1 所示，其中 NSS 是一个可选的引脚，用来选择从设备。NSS 的功能是用作“片选引脚”，让主设备可以单独地与特定从设备通信，避免数据线上的冲突。

当 SPI 配置为主设备时，如果 NSS 被允许(SSOE=1)，NSS 引脚作为输出引脚被拉低，此时所有 NSS 引脚连接到主设备 NSS 引脚的 SPI 设备会检测到低电平，如果它们配置为 NSS 硬件模式，就会自动进入从设备状态。

当 SPI 配置为主设备、NSS 配置为输入引脚（MSTR=1，SSOE=0）时，如果 NSS 被拉低，则这个 SPI 设备进入主设备失败状态：MSTR 位被自动清除，SPI 变为从设备。

从设备的 NSS 引脚也可以由主设备的一个标准 I/O 引脚来驱动。

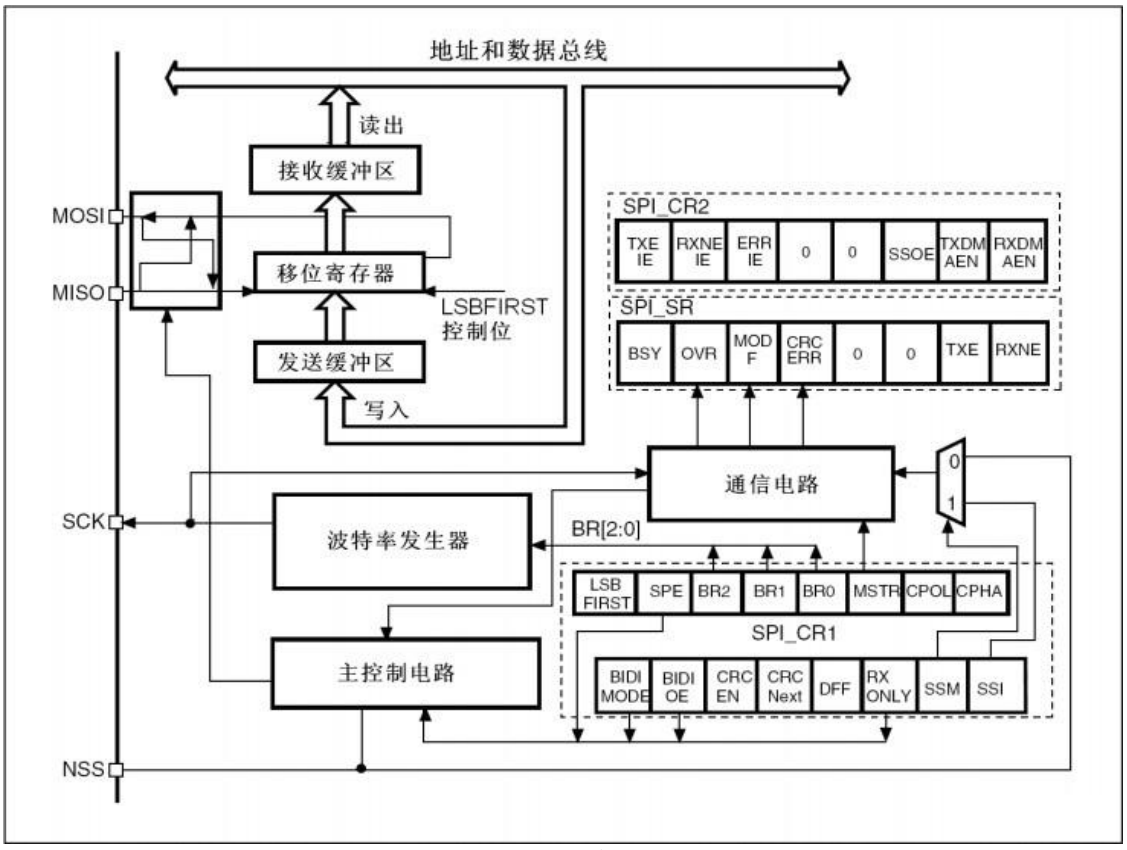


图 3.1　SPI 方框图

SPI 通过 7 个寄存器进行操作，如表 3.2 所示（SPI1 的基地址为 0x40013000）。

表 3.2　SPI 寄存器

偏移地址	名称	类型	复位值	说　明
0x00	CR1	读/写	0x0000	控制寄存器 1
0x04	CR2	读/写	0x0000	控制寄存器 2
0x08	SR	读	0x0002	状态寄存器
0x0C	DR	读/写	0x0000	数据寄存器
0x10	CRCPR	读/写	0x0007	CRC 多项式寄存器
0x14	RXCRCR	读	0x0000	接收 CRC 寄存器
0x18	TXCRCR	读	0x0000	发送 CRC 寄存器

常用的状态寄存器（SR）状态位如表 3.3 所示。

表 3.3　常用状态寄存器状态位

位	名称	类型	复位值	说　明
7	BSY	读	0	忙标志（由硬件置位或复位）
1	TXE	读	1	发送数据寄存器空（写 DR 清除）
0	RXNE	读	0	接收数据寄存器不空（读 DR 清除）

常用的控制寄存器 1（CR1）控制位如表 3.4 所示。

表 3.4　常用控制寄存器 1 控制位

位	名称	类型	复位值	说　　明
11	DFF	读/写	0	数据帧格式：0-8 位，1-16 位
7	LSBFIRST	读/写	0	帧格式：0-先发送 MSB，1-先发送 LSB
6	SPE	读/写	0	SPI 允许
5:3	BR[2:0] 波特率控制	读/写	0	000-f_{PCLK}/2　001-f_{PCLK}/4　010-f_{PCLK}/8　011-f_{PCLK}/16 100-f_{PCLK}/32　101-f_{PCLK}/64　110-f_{PCLK}/128　111-f_{PCLK}/256
2	MSTR	读/写	0	主设备选择：0-从设备，1-主设备
1	CPOL	读/写	0	时钟极性（Clock polarity）
0	CPHA	读/写	0	时钟相位（Clock phase）

时钟极性和时钟相位所有组合的信号波形如图 3.2 所示。

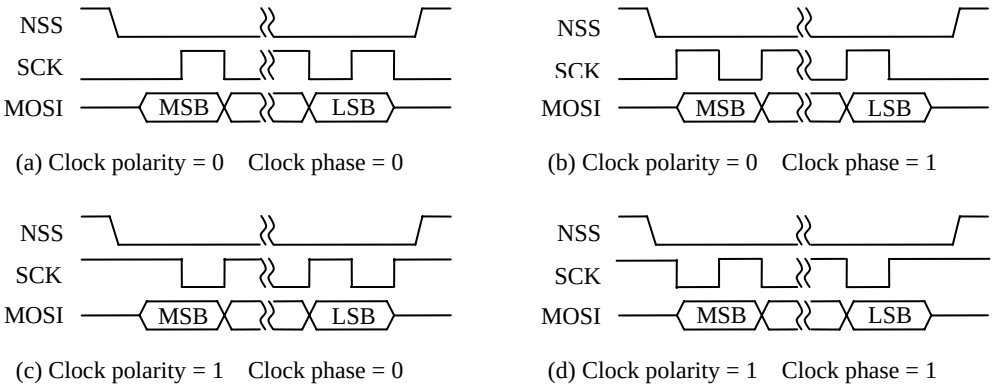


图 3.2　时钟极性和时钟相位所有组合的信号波形

常用的控制寄存器 2（CR2）控制位如表 3.5 所示。

表 3.5　常用控制寄存器 2 控制位

位	名称	类型	复位值	说　　明
7	TXEIE	读/写	0	发送数据寄存器空中断允许
6	RXNEIE	读/写	0	接收数据寄存器不空中断允许
2	SSOE	读/写	0	SS 输出允许

3、SPI 接口软件设计

直接操作寄存器的 SPI 初始化子程序如下：

```
void Spi1_Init(void)
{
  RCC_APB2ENA |= 0x1005;        // 允许SPI1、GPIOA、AFIO时钟
  GPIOA_CRL |= 0xd0d50000;      // PA7(MOSI)PA5(SCK)复用开漏输出
                                // PA4(NSS)通用开漏输出
  GPIOA_ODR |= 0x00b0;          // PA754输出高电平

  SPI1_CR1 |= 0x0b6f;           // 16位、SPI允许、PCLK/32
                                // 主模式、CPOL=1、CPHA=1
}
```

SPI 发送子程序如下：

```
// SPI发送
// 入口参数：data-发送数据
void Spi1_Txd(int data)
{
  GPIOA_BRR = 0x0010;          // PA4 = 0
  while(!(SPI1_SR & 0x02));    // 发送数据寄存器不空等待
  SPI1_DR = data;              // 发送数据
  while(SPI1_SR & 0x80);       // 忙等待
  GPIOA_BSRR = 0x0010;         // PA4 = 1
}
```

SPI 接收子程序如下：

```
// SPI接收子程序
// 出口参数：接收数据（接收成功）/0（接收不成功）
char Spi1_Rxd()
{
  if(Spi1_SR & 0x01)           // 接收数据寄存器不空
    return Spi1_DR;            // 接收数据
  else
    return 0;
}
```

利用SPI进行LCD初始化的子程序如下：

```
// LCD初始化
void Lcd_Init(void)
{
  Spi1_Init();                 // SPI1初始化

  Spi1_Txd(0x42a<<5);          // 模块初始化
  Spi1_Txd(0x418<<5);          // 内部振荡器
  Spi1_Txd(0x401<<5);          // 开振荡器
  Spi1_Txd(0x403<<5);          // 开显示器
}
```

四、实验报告

1、实验目的；（5 分）
2、实验内容；（5 分）
3、硬件方框图和电路图；（30 分）
4、软件流程图和核心语句；（30 分）
5、设计过程中遇到的问题和解决方法；（20 分）
6、收获和建议等。（10 分）

实验 4　UART 实验

一、实验目的

1、掌握 UART 接口的基本原理和主要指标；

2、掌握 UART 接口的编程原理和使用方法。

二、实验内容

在实验 2 硬件设计的基础上增加 UART 接口，在软件设计中增加相关的程序。

1、编程实现通过 UART 接口将分秒显示在 PC 屏幕上（每秒显示 1 次）；

2、编程实现通过 PC 键盘完成分秒设置；

3、用 printf()实现分秒显示，用中断方式实现分秒设置。（选做）

三、实验基础

1、UART 接口简介

UART（Universal Asynchronous Receiver/Transmitter：通用异步收发器）接口是异步串行通信接口的总称，包括 RS232、RS422、RS423、RS449 和 RS485 等，其中 RS232 是 PC 上常用的串行通信接口，相关标准规定了接口的机械特性、电气特性和功能特性等。

RS232 机械特性规定 RS232 使用 25 针 D 型连接器，后来简化为 9 针 D 型连接器。

RS232 电气特性规定逻辑“1”的电平低于-3V，逻辑“0”的电平高于+3V，和 TTL 电平不同，因此通过 RS232 和 TTL 电路通信时必须进行电平转换。RS232 的电平可高达±12V，所以通信距离可达 100 英尺（约 30m），数据速率可达 20Kbps，改进后可达 115.2Kbps。

RS232 功能特性规定各引脚的功能，9 针 D 型连接器的引脚功能如表 4.1 所示，其中最常用的引脚只有 3 个：RXD（接收数据）、TXD（发送数据）和 GND（地）。

表 4.1　RS232 引脚功能

引脚	功能	方向	说　明	引脚	功能	方向	说　明
1	DCD	输入	载波检测	6	DSR	输入	数据设备准备好
2	RXD	输入	接收数据	7	RTS	输出	请求发送
3	TXD	输出	发送数据	8	CTS	输入	清除发送
4	DTR	输出	数据终端准备好	9	RI	输入	振铃指示
5	GND		地				

RS232 的主要指标有 2 个：数据速率和数据格式。数据速率用波特率表示，数据格式包括 1 个起始位、8-9 个数据位、1 个校验位和 1-2 个停止位。通信双方的数据速率和数据格式必须一致，否则无法实现通信。

2、UART 接口设计

UART 通过接收数据输入 RX 和发送数据输出 TX 收发数据，数据的收发使用双重数据缓冲：收发移位寄存器和收发数据寄存器，收发移位寄存器在收发脉冲的作用下完成接收数据的串并转换和发送数据的并串转换，收发脉冲由波特率发生器产生。

UART 的方框图如图 4.1 所示。

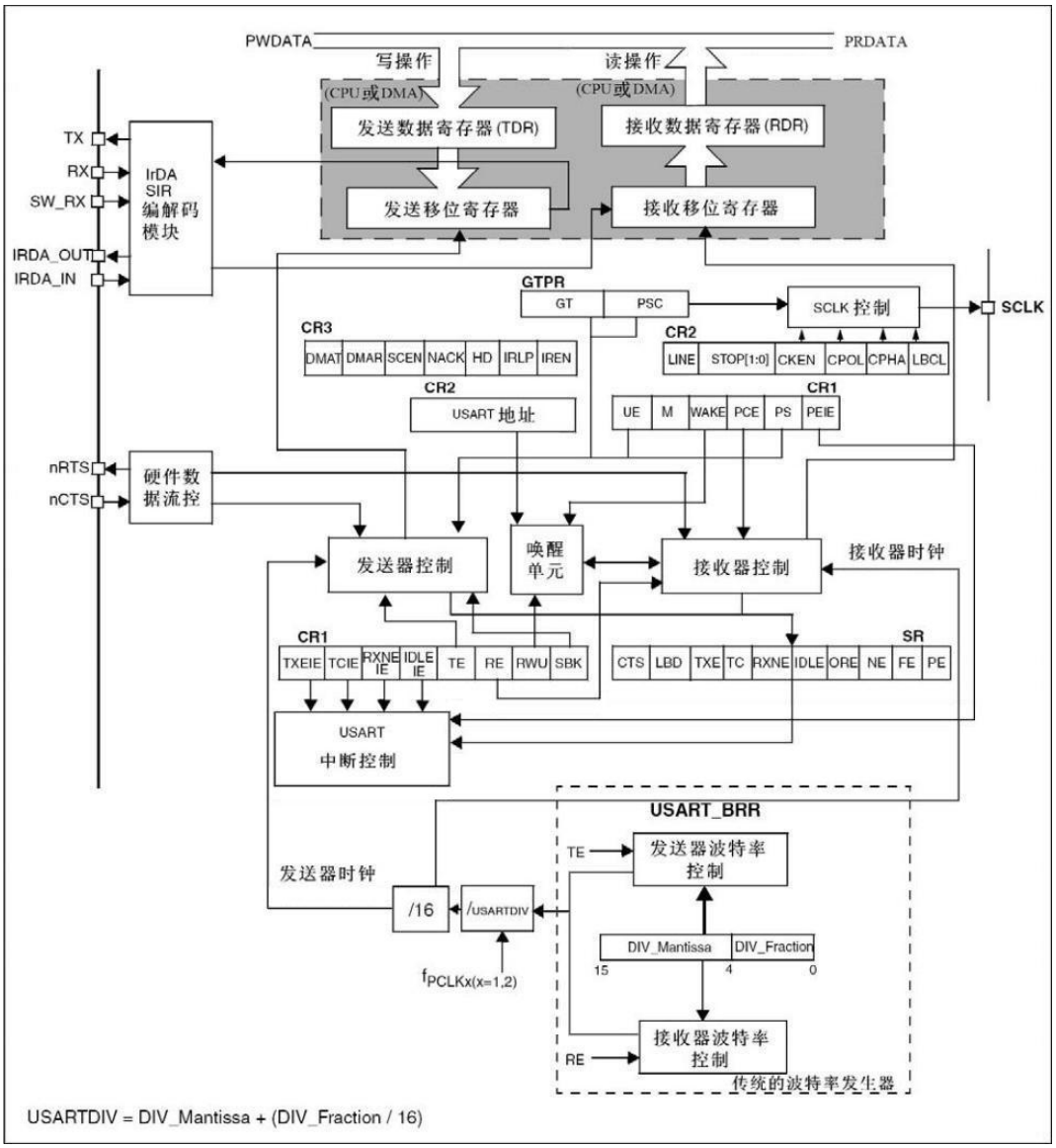


图 4.1　UART 方框图

UART 通过 7 个寄存器进行操作，如表 4.2 所示（UART1 的基地址为 0x40013800）。

表 4.2　UART 寄存器

偏移地址	名称	类型	复位值	说　　明
0x00	SR	读/写清除	0x00C0	状态寄存器
0x04	DR	读/写	-	数据寄存器
0x08	BRR	读/写	0x0000	波特率寄存器
0x0C	CR1	读/写	0x0000	控制寄存器 1
0x10	CR2	读/写	0x0000	控制寄存器 2
0x14	CR3	读/写	0x0000	控制寄存器 3
0x18	GTPR	读/写	0x0000	保护时间和预定标寄存器（智能卡使用）

常用的状态寄存器（SR）状态位如表 4.3 所示。

表 4.3　常用状态寄存器状态位

位	名称	类型	复位值	说　　明
7	TXE	读	1	发送数据寄存器空（写 DR 清除）
5	RXNE	读/写 0 清除	0	接收数据寄存器不空（读 DR 清除）

波特率寄存器（BRR）如表 4.4 所示。

表 4.4　波特率寄存器

位	名称	类型	复位值	说　　明
15:4	DIV_Mantissa	读/写	0	分频值整数部分，分频值=f_{CK}/(16*波特率)
3:0	DIV_Fraction	读/写	0	分频值小数部分

常用的控制寄存器（CR1 和 CR2）控制位如表 4.5 和 4.6 所示。

表 4.5　常用控制寄存器 1 控制位

位	名称	类型	复位值	说　　明
13	UE	读/写	0	UART 允许
12	M	读/写	0	字长：0-8 位，1-9 位
10	PCE	读/写	0	校验控制允许：0-禁止，1-允许
9	PS	读/写	0	校验选择：0-偶校验，1-奇校验
7	TXEIE	读/写	0	发送数据寄存器空中断允许
5	RXNEIE	读/写	0	接收数据寄存器不空中断允许
3	TE	读/写	0	发送允许
2	RE	读/写	0	接收允许

表 4.6　常用控制寄存器 2 控制位

位	名称	类型	复位值	说　　明
13:12	STOP[1:0]	读/写	0	停止位数：00-1 位，10-2 位

3、UART 接口软件设计

直接操作寄存器的 UART 初始化子程序如下：

```
// UART1初始化
void Uart1_Init(void)
{
  RCC_APB2ENA |= 0x4005;        // 允许UART1、GPIOA、AFIO时钟
  GPIOA_CRH ^= 0x000000f0;      // PA9(TX1)复用推挽输出
                                // PA10(RX1)悬浮输入(复位状态)
  UART1_BRR = 0x0045;           // 8M/(16*115200)=4.34(*16)
  UART1_CR1 = 0x200C;           // UART允许、发送允许、接收允许
                                // 8位数据、无校验、1位停止
}
```

UART 发送和接收子程序如下：

```
// UART发送子程序
// 入口参数：data-发送数据
void Uart1_Txd(char data)
```

```
{
  while(!(UART1_SR & 0x80));   // 发送数据寄存器不空等待
  UART1_DR = data;             // 发送数据
}
// UART接收子程序
// 出口参数：接收数据（接收成功）/0（接收不成功）
char Uart1_Rxd()
{
  if(UART1_SR & 0x20)          // 接收数据寄存器不空
    return UART1_DR;           // 接收数据
  else
    return 0;
}
```

发送秒个位的语句为：

```
Uart1_Txd((sec & 0x0f) + 0x30);
```

按实验内容要求编写程序，编译并运行程序。

用交叉串口线将实验板和 PC 连接起来，在 PC 上进行下列操作：

① 单击“开始”，选择“所有程序”>“附件”>“通讯”>“超级终端”打开“新建连接 – 超级终端”；

② 在“连接描述”中输入任意名称，单击“确定”；

③ 在“连接到”中的“连接时使用”下拉框中选择“COM1”，单击“确定”；

④ 在“COM1 属性”中选择“每秒位数”为 115200，“数据流控制”为“无”。

正常情况下在 PC 的超级终端中可以看到分秒的显示，在超级终端中输入数字可以设置实验板的分秒。

四、实验报告

1、实验目的；（5 分）

2、实验内容；（5 分）

3、硬件方框图和电路图；（30 分）

4、软件流程图和核心语句；（30 分）

5、设计过程中遇到的问题和解决方法；（20 分）

6、收获和建议等。（10 分）

实验 5　实时钟设计

一、实验目的

1、掌握嵌入式系统的组成和设计方法；
2、掌握嵌入式系统硬件的设计步骤和要点；
3、掌握嵌入式系统软件的设计步骤和调试方法；
4、掌握定时器、并行接口和 UART 的原理和使用；

二、实验内容

综合实验 1-4 实现下列功能：
1、定时器用于实时钟的月日、时分和秒计时（20 分）；
2、2 个按键用于实时钟的显示切换（20 分）和**设置**（20 分）；
3、LCD 用于实时钟的月日、时分和秒显示（20 分）；
4、通过 UART 接口在微机上显示（10 分）和设置（10 分）实时钟。

三、实验基础

1、系统状态图
按键操作和 LCD 显示的状态图如图 5.1 所示。

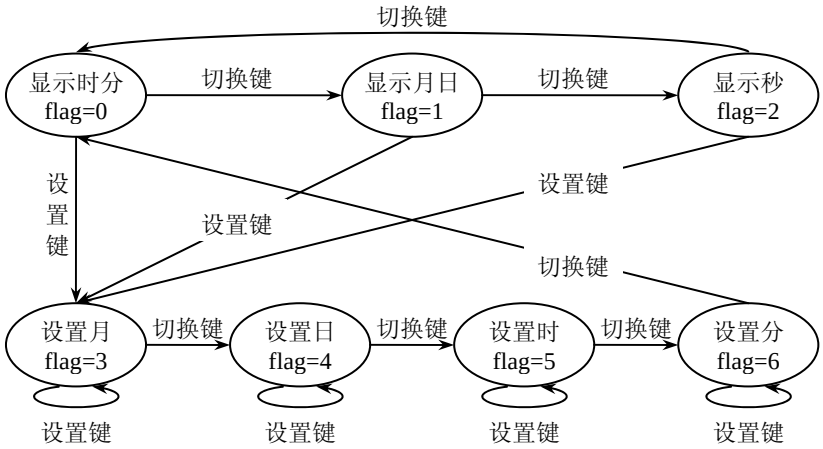


图 5.1　按键操作和 LCD 显示状态图

2、外部中断/事件控制器（EXTI）

每个配置为输入方式的 GPIO 引脚都可以配置成外部中断/事件方式（EXTI），每个中断/事件都有独立的触发和屏蔽，触发请求可以是上升沿、下降沿或者双边沿触发。

每个外部中断都有对应的挂起标志，系统可以查询挂起标志响应触发请求，也可以在中断允许时以中断方式响应触发请求。

外部中断/事件控制器方框图如图 5.2 所示（系统默认的外部中断输入线是 PA0-15，可以通过 AFIO 的 EXTI 控制寄存器 AFIO_EXTICR1-4 配置成其他 GPIO 引脚）。

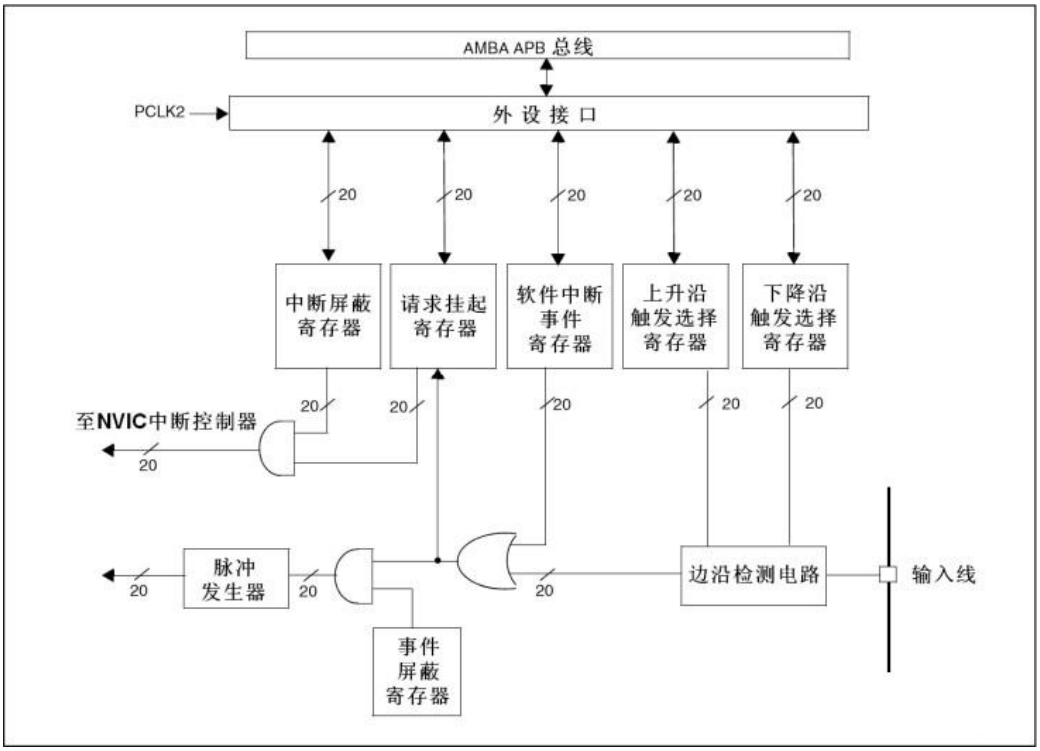


图 5.2　外部中断/事件控制器方框图

EXTI 通过 6 个寄存器进行操作，如表 5.1 所示（EXTI 的基地址为 0x40010400）。

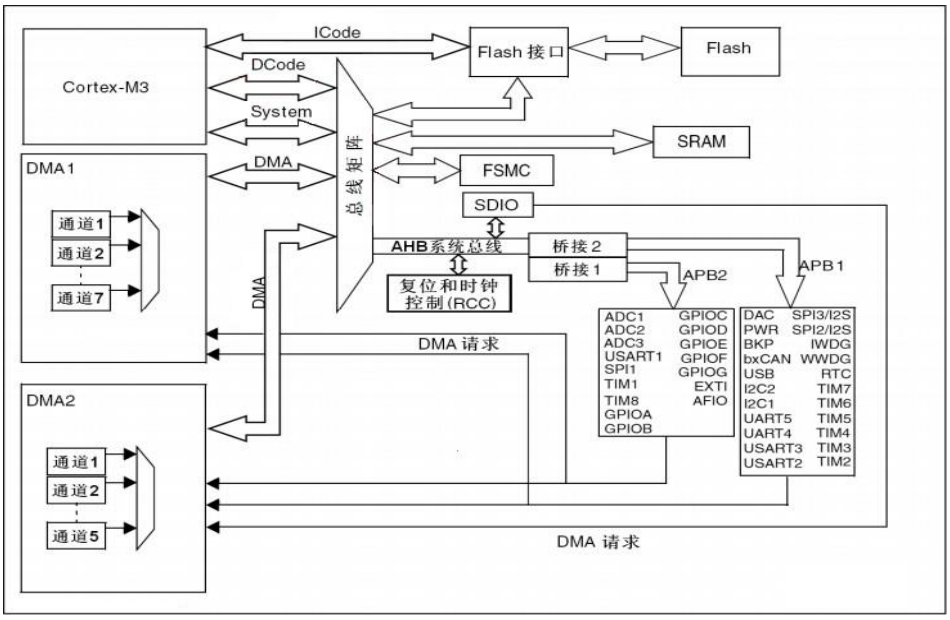
表 5.1　EXIT 寄存器

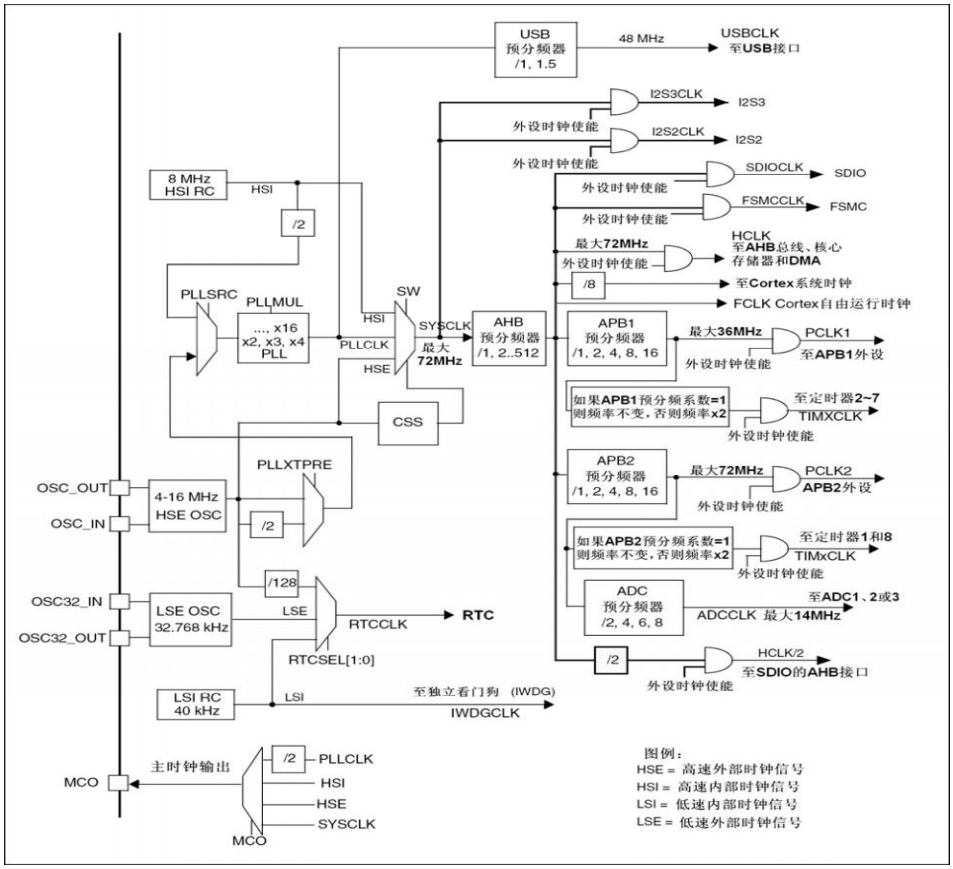
偏移地址	名称	类型	复位值	说　　明
0x00	IMR	读/写	0x00000	中断屏蔽寄存器：0-屏蔽，1-允许
0x04	EMR	读/写	0x00000	事件屏蔽寄存器：0-屏蔽，1-允许
0x08	RTSR	读/写	0x00000	上升沿触发选择寄存器：0-禁止，1-允许
0x0C	FTSR	读/写	0x00000	下降沿触发选择寄存器：0-禁止，1-允许
0x10	SWIER	读/写	0x00000	软件中断事件寄存器
0x14	PR	读/写 1 清除	0xXXXXX	请求挂起寄存器：0-无触发请求，1-有触发请求

四、实验报告

1、实验目的；　　（5 分）
2、实验内容；　　（5 分）
3、硬件方框图和电路图；　　（30 分）
4、软件流程图和核心语句；　　（30 分）
5、设计过程中遇到的问题和解决方法；　　（20 分）
6、收获和建议等。　　（10 分）

附录 1　芯片系统结构图和时钟树





附录 2　存储器映像表

地址范围		设备名称
0xE000 0000 - 0xE00F FFFF		内部设备
0x4000 0000 - 0x5003 FFFF		外部设备
AHB	0x5000 0000 - 0x5003 FFFF	USB OTG 全速
	0x4003 0000 - 0x4FFF FFFF	保留
	0x4002 8000 - 0x4002 9FFF	以太网
	0x4002 3400 - 0x4002 7FFF	保留
	0x4002 3000 - 0x4002 33FF	CRC
	0x4002 2000 - 0x4002 23FF	FLASH 接口
	0x4002 1400 - 0x4002 1FFF	保留
	0x4002 1000 - 0x4002 13FF	RCC 制(复位和时钟控)
	0x4002 0800 - 0x4002 0FFF	保留
	0x4002 0400 - 0x4002 07FF	DMA2
	0x4002 0000 - 0x4002 03FF	DMA1
	0x4001 8400 - 0x4001 7FFF	保留
	0x4001 8000 - 0x4001 83FF	SDIO
APB2	0x4001 4000 - 0x4001 7FFF	保留
	0x4001 3C00 - 0x4001 3FFF	ADC3
	0x4001 3800 - 0x4001 3BFF	USART1
	0x4001 3400 - 0x4001 37FF	TIM8 定时器
	0x4001 3000 - 0x4001 33FF	SPI1
	0x4001 2C00 - 0x4001 2FFF	TIM1 定时器
	0x4001 2800 - 0x4001 2BFF	ADC2
	0x4001 2400 - 0x4001 27FF	ADC1
	0x4001 2000 - 0x4001 23FF	GPIO 端口 G
	0x4001 1C00 - 0x4001 1FFF	GPIO 端口 F
	0x4001 1800 - 0x4001 1BFF	GPIO 端口 E
	0x4001 1400 - 0x4001 17FF	GPIO 端口 D
	0x4001 1000 - 0x4001 13FF	GPIO 端口 C
	0x4001 0C00 - 0x4001 0FFF	GPIO 端口 B
	0x4001 0800 - 0x4001 0BFF	GPIO 端口 A
	0x4001 0400 - 0x4001 07FF	EXTI
	0x4001 0000 - 0x4001 03FF	AFIO

APB1	0x4000 7800 - 0x4000 FFFF	保留
	0x4000 7400 - 0x4000 77FF	DAC
	0x4000 7000 - 0x4000 73FF	电源控制(PWR)
	0x4000 6C00 - 0x4000 6FFF	后备寄存器(BKP)
	0x4000 6800 - 0x4000 6BFF	bxCAN2
	0x4000 6400 - 0x4000 67FF	bxCAN1
	0x4000 6000 - 0x4000 63FF	USB/CAN 共享的 512 字节 SRAM
	0x4000 5C00 - 0x4000 5FFF	USB 全速设备寄存器
	0x4000 5800 - 0x4000 5BFF	I2C2
	0x4000 5400 - 0x4000 57FF	I2C1
	0x4000 5000 - 0x4000 53FF	UART5
	0x4000 4C00 - 0x4000 4FFF	UART4
	0x4000 4800 - 0x4000 4BFF	USART3
	0x4000 4400 - 0x4000 47FF	USART2
	0x4000 4000 - 0x4000 43FF	保留
	0x4000 3C00 - 0x4000 3FFF	SPI3/I2S3
	0x4000 3800 - 0x4000 3BFF	SPI2/I2S2
	0x4000 3400 - 0x4000 37FF	保留
	0x4000 3000 - 0x4000 33FF	独立看门狗(IWDG)
	0x4000 2C00 - 0x4000 2FFF	窗口看门狗(WWDG)
	0x4000 2800 - 0x4000 2BFF	RTC
	0x4000 1800 - 0x4000 27FF	保留
	0x4000 1400 - 0x4000 17FF	TIM7 定时器
	0x4000 1000 - 0x4000 13FF	TIM6 定时器
	0x4000 0C00 - 0x4000 0FFF	TIM5 定时器
	0x4000 0800 - 0x4000 0BFF	TIM4 定时器
	0x4000 0400 - 0x4000 07FF	TIM3 定时器
	0x4000 0000 - 0x4000 03FF	TIM2 定时器
0x2000 0000		SRAM
0x0000 0000 - 0x1FFF FFFF		FLASH
	0x1FFF F800 - 0x1FFF F80F	选择字节
	0x1FFF F000 - 0x1FFF F7FF	系统存储器
	0x0800 0000 - 0x0801 FFFF	主存储器

附录 3　开发板相关部件连接关系表

开发板	原理图	CPU	功能描述
SW1	RESET	NRST	复位按键
SW2	KEY0	PB15	按键0
SW3	KEY1	PB14	按键1
	LCD_CS	PC12	LCD选择
	LCD_WR	PC11	LCD写
	LCD_DA	PC10	LCD数据
	LCD_RD	PD2	LCD读
	OSC_OUT	PD1	8MHz
	OSC_IN	PD0	8MHz
	OSC32_OUT	PC15	32KHz
	OSC32_IN	PC14	32KHz
DS1	POWER		+3.3V电源指示
JP1	+5V		+5V（外接）
JP2	RST_1302	PA1	DS1302复位
	DATA_1302	PA2	DS1302数据
	SCLK_1302	PA3	DS1302时钟
JP3	SCL_EEPROM	PB6	I^2C SCL（E^2PROM）
	SDA_EEPROM	PB7	I^2C SDA（E^2PROM）
JP4	T1_IN	PA9	UART1发送
	R1_OUT	PA10	UART1接收
	USB_D-	PA11	USB_DM
	USB_D+	PA12	USB_DP
JP5	TEMP_DS18B20	PB13	DS18B20单总线
JP6	BOOT0	BOOT0	BOOT0选择
JP7	LED0	PC9	DS2
	LED1	PC8	DS3
	LED2	PC7	DS4
	LED3	PC6	DS5
JP8	BOOT1	PB2	BOOT1选择
JP9			USB_DP上拉选择
JP10			PIN1~16脚引出
JP11			PIN17~32脚引出
JP12			PIN33~48脚引出
JP13			PIN49~64脚引出
JP14			VBAT输入选择