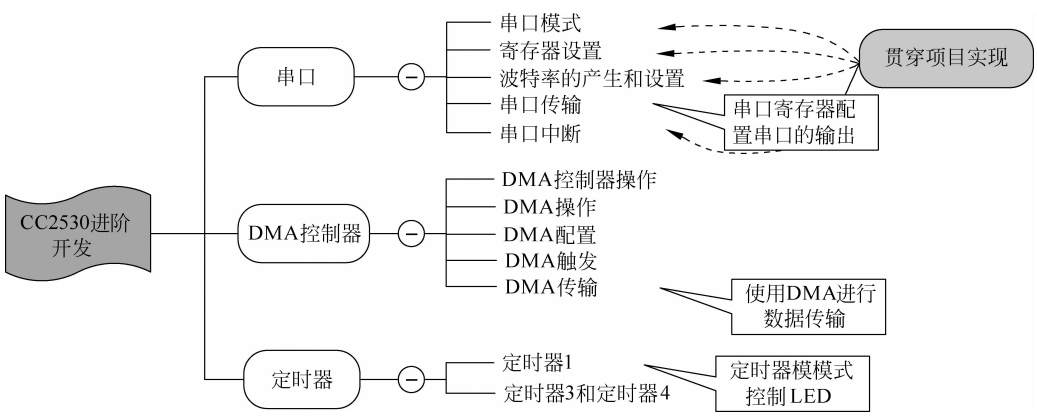
第5章 CC2530进阶开发

任务驱动

基于 ZigBee 的智能家居环境信息采集系统必须要有传感信息的采集，本章将完成任务——传感信息采集传输。具体任务如下：

CC2530 控制 DS18B20 采集传感信息并通过串口传输。

学习导航 / 课程定位



本章目标

知识点	Listen（听）	Know（懂）	Do（做）	Revise（复习）	Master（精通）
串口模式	★				
串口寄存器	★	★			
波特率的设置	★	★	★		
串口中断	★	★	★	★	
DMA 寄存器	★	★	★	★	
DMA 操作和配置	★	★	★	★	★
DMA 触发和传输	★	★	★	★	★
定时器寄存器	★	★	★	★	★
定时器使用	★	★	★	★	★
串口传输	★	★	★	★	★

5.1 串口

串口即串行通信接口，CC2530 有两个串行通信接口：USART0 和 USART1。两个 USART 具有相同的功能，可以通过设置相应的寄存器来决定选用哪一个串口。本节将讲述串口相关的操作和应用，例如串口模式、寄存器设置、串口波特率、串口中断以及串口的应用实例。

5.1.1 串口模式

CC2530 的串口有两种通信模式：UART 模式和 SPI 模式。其中 UART 模式为异步串行通信接口；SPI 为同步串行通信接口。

1. UART 模式

在 UART 模式中，接口使用两线或四线连接，使用全双工传送，接收器中的位同步不影响发送功能。异步串行通信接口 UART 模式操作具有以下特点：

- 8 位或 9 位负载数据；
- 奇校验、偶校验或者无奇偶校验；
- 配置起始位和停止位电平；
- 配置 LSB 或者 MSB 首先传送；
- 独立收发中断；
- 独立收发 DMA 触发；
- 奇偶校验和帧校验出错状态。

串口在 UART 模式下有发送和接收两种模式，数据的发送和接收由相应的寄存器设置，串口寄存器的设置详见 5.1.2 节介绍。

2. SPI 模式

在 SPI 模式中，串口通过 3 线接口或者 4 线接口与外部通信，SPI 接口包含引脚 MOSI、MISO、SCK 和 SSN。同步串行通信接口 SPI 模式具有以下特点：

- 3 线或者 4 线 SPI 接口；
- 具备主模式和从模式；
- 可配置的 SCK 极性和相位；
- 可配置的 LSB 或 MSB 传送。

5.1.2 寄存器设置

串口有 5 个寄存器，分别是串口控制和状态寄存器 UxCSR、串口 USART 控制寄存器 UxUCR、串口接收/传送数据缓存寄存器 UxDBUF 寄存器、串口波特率控制寄存器 UxBAUD 寄存器和串口通用控制 UxGCR 寄存器，其中 x 的取值为 0 或 1。

以下以串口 0 为例来讲解串口寄存器的设置和使用。

串口0控制和状态寄存器U0CSR的主要功能为：选择串口模式为SPI模式或者UART模式、负责UART接收器的打开和关闭、SPI模式的选择、UART帧状态检测、UART奇偶校验错误状态、串口接收发送字节状态和串口发送和接受的主动状态。其具体设置如表5-1所示。

表5-1 USART 0控制和状态寄存器U0CSR

位	名称	复位	R/W	描 述
7	MODE	0	R/W	USART模式选择 0：SPI模式 1：UART模式
6	RE	0	R/W	UART接收器使能，但是在UART完全配置之前不能接收。 0：禁止接收器 1：使能接收器
5	SLAVE	0	R/W	SPI主或者从模式选择 0：SPI主模式 1：SPI从模式
4	FE	0	R/W0	UART帧错误状态 0：无帧错误检测 1：字节收到不正确停止位级别
3	FRR	0	R/W0	UART奇偶校验错误状态 0：无奇偶校验检测 1：字节收到奇偶错误
2	RX_BYTE	0	R/W0	接收字节状态，UART模式和SPI模式。当读U0DBUF该位自动清零，通过写0清除它，这样有效丢弃U0BUF中的数据 0：没有收到字节 1：接收字节就绪
1	TX_BYTE	0	R/W0	传送字节状态，UART和SPI从模式 0：字节没有传送 1：写到数据缓存寄存器的最后字节已经传送
0	ACTIVE	0	R	USART传送/接收主动状态 0：USART空闲 1：USART在传送或者接收模式忙碌

- U0CSR寄存器的第7位主要负责串口模式选择，当该位设置为0时，选择SPI模式；当该位设置为1时，选择UART模式。
- U0CSR寄存器的第6位主要功能为使能或禁止UART模式接收器，当该位设置为0时，关闭UART接收器；当该位设置为1时，打开UART接收器。
- U0CSR寄存器的第5位主要负责SPI模式的选择，当该位设置为0时，选择SPI主模式；当此该位设置为1时，选择SPI从模式。
- U0CSR寄存器的第4位主要负责串口帧错误状态检测，当该位设置为0时，没有帧错误检测；当该位设置为1时，字节收到不正确停止位级别。
- U0CSR寄存器的第3位主要负责UART模式的奇偶错误状态，当该位设置为0时，无奇偶错误检验；当该位设置为1时，收到奇偶检验错误。

- U0CSR 寄存器的第 2 位主要负责接收字节状态状态，当该位为 0 时，表示没有收到字节；当该位为 1 时，表示准备接收字节完毕。
- U0CSR 寄存器的第 1 位主要负责传送字节状态，当该位设置为 0 时，没有字节被传送；当该位设置为 1 时，写到数据缓存寄存器的最后字节被传送。
- U0CSR 寄存器的第 0 位主要负责串口发送和接收的主动状态，当此位设置为 0 时，串口空闲；当此位设置为 1 时，串口有数据发送或接收。

如果要设置串口 0 选择 UART 模式，其具体设置如示例 5-1 所示。

【示例 5-1】 U0CSR 寄存器配置

```
/* UART 方式 */
U0CSR |= 0x80;
```

串口 0 UART 控制寄存器 U0UCR 的主要功能为：UART 硬件流控制、UART 奇偶校验位设置、选择数据传送位、奇偶校验使能、选择停止位数、选择停止位和起始位电平。其具体设置如表 5-2 所示。

表 5-2 USART 0 UART 控制 U0UCR

位	名称	复位	R/W	描 述
7	FLUSH	0	R/W1	清除单元。当设置时，该事件将会立即停止当前操作并返回单元的空闲状态
6	FLOW	0	R/W	UART 硬件流使能。用 RTS 和 CTS 引脚选择硬件流控制的使用 0：流控制禁止 1：流控制使能
5	D9	0	R/W	UART 奇偶校验位。当使能奇偶校验，写入 D9 的值决定发送的第 9 位的值。如果收到的第 9 位不匹配收到的字节的奇偶校验，接收报告 ERR 0：奇校验 1：偶校验
4	BIT9	0	R/W	UART9 位数据使能。当该位是 1 时，使能奇偶校验位传输即第 9 位。如果通过 PARITY 使能奇偶校验，第 9 位的内容是通过 D9 给出的 0：8 位传输 1：9 位传输
3	PARITY	0	R/W	UART 奇偶校验使能。除了为奇偶校验设置该位用于计算，必须使能 9 位模式 0：禁用奇偶校验 1：使能奇偶校验
2	SPB	0	R/W	UART 停止位数。选择要传送的停止位的位数 0：1 位停止位 1：2 位停止位
1	STOP	0	R/W	UART 停止位的电平必须不同于开始位的电平 0：停止位低电平 1：停止位高电平

续表

位	名称	复位	R/W	描 述
0	START	0	R/W	UART 起始位电平，闲置线的极性采用选择的起始位级别的电平的相反的电平 0：起始位低电平 1：起始位高电平

- U0UCR 寄存器的第 7 位主要负责清除单元。如果设置了此位，事件将会立即停止当前操作并返回单元的空闲状态。
- U0UCR 寄存器的第 6 位主要负责 UART 硬件流使能，当此位设置为 0 时，禁止硬件流控制；当此位设置为 1 时，打开硬件流控制。
- U0UCR 寄存器的第 5 位主要负责 UART 奇偶校验位，当此位设置为 0 时，选择奇校验；当此位设置为 1 时，选择偶校验。
- U0UCR 寄存器的第 4 位主要负责选择停止位数，当此位设置为 0 时，为 8 位传输；当此位设置为 1 时，为 9 位传输。
- U0UCR 寄存器的第 3 位主要负责 UART 奇偶校验使能，当此位设置为 0 时，禁止奇偶校验；当此位设置为 1 时，开启奇偶校验。
- U0UCR 寄存器的第 2 位主要负责选择停止位数，当此位为 0 时，为 1 位停止位；当此位为 1 时，为 2 位停止位。
- U0UCR 寄存器的第 1 位主要负责 UART 停止位电平，当此位设置为 0 时，停止位为低电平；当此位设置为 1 时，停止位为高电平。
- U0UCR 寄存器的第 0 位主要负责 UART 起始位电平，当此位设置为 0 时，起始位为低电平；当此位设置为 1 时，起始位为高电平。

如果要设置串口在 USART 模式下采用偶校验，其具体设置如示例 5-2 所示。

【示例 5-2】 U0UCR 寄存器配置

```
/* UART 方式 */
U0UCR |= 0x08;
```

串口 0 数据接收和发送缓存寄存器 U0BUF 的主要功能是存放串口接收和发送的数据。其具体设置如表 5-3 所示。

表 5-3 USART 0 接收/发送数据缓存寄存器 U0DBUF

位	名称	复位	R/W	描 述
7～0	DATA [7：0]	0x00	R/W	USART 接收和发送数据。当写这个寄存器的时候数据被写到内部的传送数据寄存器，当读取该寄存器的时候，数据来自内部读取的数据寄存器

串口 0 发送的数据是通过写 U0DBUF 来实现的，当有数据要发送时，将数据写入 U0DBUF 寄存器中即可，具体如示例 5-2 所示。

【示例 5-3】 串口发送数据

```
void UartTX_Send_String(char *Data,int len)
{
  int j;
  for(j=0;j<len;j++)
  {
    U0DBUF = *Data++;
    while(UTX0IF == 0);
    UTX0IF = 0;
  }
}
```

波特率由两部分组成,波特率的小数部分和整数部分。其中小数部分由波特率控制寄存器 U0BAUD 来决定,其整数部分由 U0GCR 的第 0～4 位 BAUD_E 来决定,其具体设置如表 5-4 和表 5-5 所示。

表 5-4 USART 0 波特率控制 U0BAUD

位	名称	复位	R/W	描 述
7～0	BAUD_M	0x00	R/W	波特率小数部分的值。BAUD_E 和 BAUD_M 决定了 UART 的波特率和 SPI 的主 SCK 时钟频率

表 5-5 USART 0 通用控制 U0GCR

位	名称	复位	R/W	描 述
7	CPOL	0	R/W	SPI 的时钟极性 0：SPI 总线空闲时时钟极性为低电平 1：SPI 总线空闲时时钟极性为高电平
6	CPHA	0	R/W	SPI 时钟相位 0：时钟前沿采样,后沿输出 1：时钟后沿采样,前沿输出
5	ORDER	0	R/W	传送位顺序 0：LSB 先传送 1：MSB 先传送
4～0	BAUD_E[4:0]	00000	R/W	波特率指数值。BAUD_E 和 BAUD_M 决定了 UART 的波特率和 SPI 的主 SCK 时钟频率

其波特率的具体产生和设置详见 5.1.3 节内容。

5.1.3 波特率的产生和设置

串口波特率的产生除了相应的寄存器设置,还与系统主时钟的选择有关,其波特率的计算方法如公式(5-1)所示。

$$\text{波特率} = \frac{(256 + \text{BAUD\_M}) \times 2^{\text{BAUD\_E}}}{2^{28}} \times f \tag{5-1}$$

其中 BAUD_M 和 BAUD_E 由寄存器 UxBAUD 和 UxGCR 设置(x 的取值为 0 或 1),f 为主时钟频率。当系统主时钟选择 32MHz 时,BAUD_M 和 BAUD_E 的值详见表 5-6。

表 5-6 波特率的产生

波特率(bps)	U*x*BAUD. BAUD_M	U*x*GCR. BAUD_E	误差(%)
2400	59	6	0.14
4800	59	7	0.14
9600	59	8	0.14
14 400	216	8	0.03
19 200	59	9	0.14
28 800	216	9	0.03
38 400	59	10	0.14
57 600	216	10	0.03
76 800	59	11	0.14
115 200	216	11	0.03
230 400	216	12	0.03

例如,当 BAUD_M 取值为 216,BAUD_E 取值为 10 时,f 取值为 32MHz,计算波特率如式(5-2)所示。

$$\text{波特率} = \frac{(256+216)\times 2^{10}}{2^{28}} \times 32 \times 10^{6} \approx 57\,600 \tag{5-2}$$

5.1.4 串口传输

以下实例将实现 CC2530 串口发送数据,要实现 CC2530 串口发送数据首先要对串口进行初始化、然后在主函数中调用初始化函数以及串口发送数据函数。

1. 头文件

头文件需要添加 CC2530 的头文件<ioCC2530.h>,定义 LED1 和 LED2 状态作为串口发送数据的指示灯,并对相关函数进行函数声明。

C 语言的函数在使用之前,需要首先声明。在整个实例的实现过程中需要有延时函数 Delay()、串口初始化函数 initUART()和串口发送函数 UartTX_Send_String()函数的声明。

具体代码如实例 5-1"头文件"所示。

【实例 5-1】 头文件

```
串口 0 发数据
#include < ioCC2530.h >
#define uint unsigned int
#define uchar unsigned char
/* 定义控制灯的端口 */
#define LED1 P1_0
#define LED2 P1_1
/* 函数声明 */
void Delay(uint);
void initUART(void);
void UartTX_Send_String(char *Data,int len);
```

2. 串口初始化函数

串口初始化需要以下几个步骤：

(1) 系统时钟的初始化；

(2) 选择串口作为 I/O 外设的引脚连接位置，此实例中是选择备用位置 1 作为串口外设；

(3) 设置串口模式，本实例选择 UART 模式，并设置其波特率为 57 600bps。

串口初始化的具体代码如实例 5-1 InitUART()所示。

【实例 5-1】 InitUART()

```
/*********************************************************************
 * 函数功能 : 初始化串口
 * 入口参数 : 无
 * 返 回 值 : 无
 *********************************************************************/
void initUART(void)
{
    /* 晶振选择 32MHz */
    CLKCONCMD &= ~0x40;
    /* 等待晶振稳定 */
    while(!(SLEEPSTA & 0x40));
    /* TICHSPD128 分频,CLKSPD 不分频 */
    CLKCONCMD &= ~0x47;
    /* 关闭不用的 RC 振荡器 */
    SLEEPCMD |= 0x04;
    /* 使用串口备用位置 1 P0 口 */
    PERCFG = 0x00;
    /* P0 用作串口 */
    P0SEL = 0x3c;
    /* 选择串口 0 优先作为串口 */
    P2DIR &= ~0XC0;
    /* UART 方式 */
    U0CSR |= 0x80;
    /* 波特率 baud_e 的选择 */
    U0GCR |= 10;
    /* 波特率设为 57600 */
    U0BAUD |= 216;
}
```

3. 串口发送函数

串口发送函数的功能是将需要发送的数据写入到 U0DBUF 中。具体代码如实例 5-1 UartTX_Send_String()所示。

【实例 5-1】 UartTX_Send_String()

```
/*********************************************************************
 * 函数功能 : 串口发送字符串函数
```

```
 * 入口参数 : data:数据
 * len :数据长度
 * 返 回 值 : 无
 ******************************************************************/
void UartTX_Send_String(char *Data,int len)
{
  int j;
  for(j=0;j<len;j++)
  {
    U0DBUF = *Data++;
    while(UTX0IF == 0);
    UTX0IF = 0;
  }
}
```

4. 主函数

主函数完成的工作如下：

(1) 定义需要发送的字符串；

(2) 设置 LED 状态；

(3) 调用串口初始化函数；

(4) 串口发送一次，两个 LED 的状态改变一次。

主函数的具体代码如实例 5-1main()所示。

【实例 5-1】 main()

```
/******************************************************************
 * 函数功能 : 主函数
 * 入口参数 : 无
 * 返 回 值 : 无
 ******************************************************************/
void main(void)
{
        char Txdata[6]="QST";
        /*P1 输出控制 LED*/
        P1DIR = 0x03;
        /*关 LED1*/
        LED1 = 0;
        /*开 LED2*/
        LED2 = 1;
        /*串口初始化*/
        initUARTtest();
        while(1)
       {
           /*串口发送数据*/
           UartTX_Send_String(Txdata,4);
            Delay(50000);
            Delay(50000);
            Delay(50000);
```

```
            LED1 = ~LED1;
            LED2 = ~LED2;
        }
    }
```

程序编写完成之后，将程序下载至设备中，运行结果如图 5-1 所示。

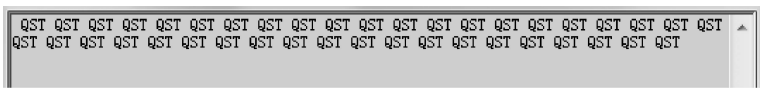


图 5-1　实例 5-1 运行结果

5.1.5　串口中断

以下实例将实现串口接收中断控制 LED 开关，与实例 5-1 不同的是需要添加中断初始化和中断处理函数以及主函数。其中头文件与实例 5-1 相同。在函数声明及变量的定义中，与实例 5-1 不同，具体代码如实例 5-2“变量的声明和定义”所示。

【实例 5-2】　变量的声明和定义

```
/* 函数的声明 */
void Delay(uint);
/* 串口初始化函数 */
void initUARTtest(void);
/* LED 初始化函数 */
void Init_LED_IO(void);
/* 字符型数组,存放要发送的字符 */
uchar Recdata[6] = "00000";
/* 字符型变量,发送数据标志 */
uchar RTflag = 1;
uchar temp;
uint  datanumber = 0;
```

串口初始化的步骤和实例 5-1 基本相同，但是在最后需要清除中断标志。具体代码如示例 5-2 InitUART()函数所示。

【实例 5-2】　InitUART()

```
/*****************************************************************
 * 函数功能 : 初始化串口 1
 * 入口参数 : 无
 * 返 回 值 : 无

 *****************************************************************/
void initUART (void)
{
    /* 晶振 */
    CLKCONCMD &= ~0x40;
    /* 等待晶振稳定 */
    while(!(SLEEPSTA & 0x40));
```

```
    /* TICHSPD128 分频,CLKSPD 不分频 */
    CLKCONCMD &= ~0x47;
    /* 关闭不用的 RC 振荡器 */
    SLEEPCMD |= 0x04;
    /* 位置 1 P0 口 */
    PERCFG = 0x00;
    /* P0 用作串口 */
    P0SEL = 0x3c;
    /* UART 方式 */
    U0CSR |= 0x80;
    /* baud_e */
    U0GCR |= 10;
    /* 波特率设为 57600 */
    U0BAUD |= 216;
    /* 串口中断标志位置 1 */
    UTX1IF = 1;
    /* 允许接收 */
    U0CSR |= 0X40;
    /* 开总中断,接收中断 */
    IEN0 |= 0x84;
}
```

LED 初始化函数将 LED1 和 LED2 关闭,具体代码如实例 5-2 Init_LED_IO()函数所示。

【实例 5-2】 Init_LED_IO()

```
void Init_LED_IO(void)
{
    /* P1.0、P1.1 控制 LED */
    P1DIR |= 0x03;
    /* 关 LED1 */
    LED1 = 0;
    /* 关 LED2 */
    LED2 = 0;
}
```

在主函数中,主要解析接收到的数据命令,如果接收到“LED11#”,打开 LED1;如果接收到“LED10#”关闭 LED1。如果接收到“LED21#”,打开 LED2;如果接收到“LED20”,关闭 LED2。

其具体代码详见实例 5-2main()函数。

【实例 5-2】 main()

```
/*****************************************************************
 * 函数功能 : 主函数
 * 入口参数 : 无
 * 返 回 值 : 无
 * 说    明 : 无
 *****************************************************************/
void main(void)
```

```
{
  uchar ii;
  Init_LED_IO();
  initUART();
  while(1)
  {
    /* 接收数据 */
    if(RTflag == 1)
    {
      if( temp != 0)
      {
        /* '*'被定义为结束字符 */
        if((temp!= '*')&&(datanumber<6))
        {
          /* 最多能接收 6 个字符 */
          Recdata[datanumber++] = temp;
        }
        else
        {
          /* 如果字符接收完毕将进入 LED 状态改变程序 */
          RTflag = 3;
        }
        /* 接收 6 个字符后进入 LED 灯控制 */
        if(datanumber == 6)
        {
          RTflag = 3;
          temp   = 0;
        }
      }
    }
    /* LED 控制程序 */
    if(RTflag == 3)
    {
      /* 判断接收的第一个字符是否为"L" */
      if(Recdata[0] == 'L')
      {
       /* 判断接收的第二个字符是否为"E" */
        if(Recdata[1] == 'E')
        {
          /* 判断接收的第三个字符是否为"D" */
          if(Recdata[2] == 'D')
          {
           /* 判断接收的第 4 个字符是否为 1,如果为 1 则控制 LED1 */
           if(Recdata[3] == '1')
            {
              /* 判断接收的第 5 个字符是否为 1,如果为 1 LED1 打开 */
              if(Recdata[4] == '1')
              {
                LED1 = 1;
              }
```

```
                /*,如果为 0LED1 关闭 */
                else
                {
                  LED1 = 0;
                }
              }
            }
          }
        }
        /* LED2 控制程序 */
        /* 判断接收的第 1 个字符是否为"L" */
        if(Recdata[0] == 'L')
        {
          /* 判断接收的第 2 个字符是否为"E" */
          if(Recdata[1] == 'E')
          {
           /* 判断接收的第 3 个字符是否为"D" */
            if(Recdata[2] == 'D')
            {
             /* 判断接收的第 4 个字符是否为"2",如果为 2 则控制 LED2 */
              if(Recdata[3] == '2')
              {
                /* 判断接收的第 5 个字符是否为 1,如果为 1 则打开 LED2 */
                if(Recdata[4] == '1')
                {
                  LED2 = 1;
                }
                /* 如果为 0 则关闭 LED2 */
                else
                {
                  LED2 = 0;
                }
              }
            }
          }
        }
        RTflag = 1;
        /* 清除接收到的数据 */
        for(ii = 0;ii < 6;ii++)Recdata[ii] = '';
        /* 指针归位 */
        datanumber = 0;
      }
    }
  }
```

中断处理函数负责在中断产生后,将接收到的数据写入到 temp 数组中,并清中断标志,具体代码如实例 5-2 UART_VECTOR()函数。

【实例 5-2】 UART_VECTOR()

```
/*********************************************************************
 * 函数功能 : 串口接收一个字符
 * 入口参数 : 无
 * 返 回 值 : 无
 * 说    明 : 接收完成后打开接收
 *********************************************************************/
#pragma vector = URX0_VECTOR
__interrupt void UART0_ISR(void)
{
    /* 清中断标志 */
    URX1IF = 0;
    /* 将接收到的数据写入到 temp 中 */
    temp = U0DBUF;
}
```

5.2 DMA 控制器

DMA 为直接存取访问控制器,可以用来减轻 8051 CPU 内核传送数据操作的负担,只需要 CPU 极少的干预,就能实现高效的电源节能管理。

5.2.1 DMA 控制器概述

DMA 控制器协调所有的 DMA 传送,确保 DMA 请求和 CPU 存储器访问之间按照优先等级协调。DMA 控制的主要功能如下:

- DMA 控制器含有若干可编程的 DMA 通道,用来实现存储器之间的数据传送。
- DMA 控制器控制整个 XDATA 存储空间的数据传送。由于大多数 SFR 寄存器映射到 XDATA 存储器空间,DMA 通道的操作能够减轻 CPU 的负担。
- DMA 控制器还可以保持 CPU 在低功耗模式下与外设单元之间传送数据,不需要唤醒,降低整个系统的功耗。

DMA 的主要特点如下:

- 5 个独立的 DMA 通道。
- 3 个可以配置的 DMA 通道优先级。
- 32 个可以配置的传送触发事件。
- 源地址和目标地址的独立控制。
- 单独传送、数据块传送和重复传送模式。
- 支持传输数据的长度域,设置可变传输长度。
- 既可以工作在字模式,又可以工作在字节模式。

5.2.2 DMA 操作

DMA 控制器有 5 个通道,即通道 0～4。每个 DMA 通道都能够从 XDATA 映射空间

的一个存储位置传送数据到另一个位置。DMA 控制器在使用之前必须对其进行配置，DMA 操作流程图如图 5-2 所示。

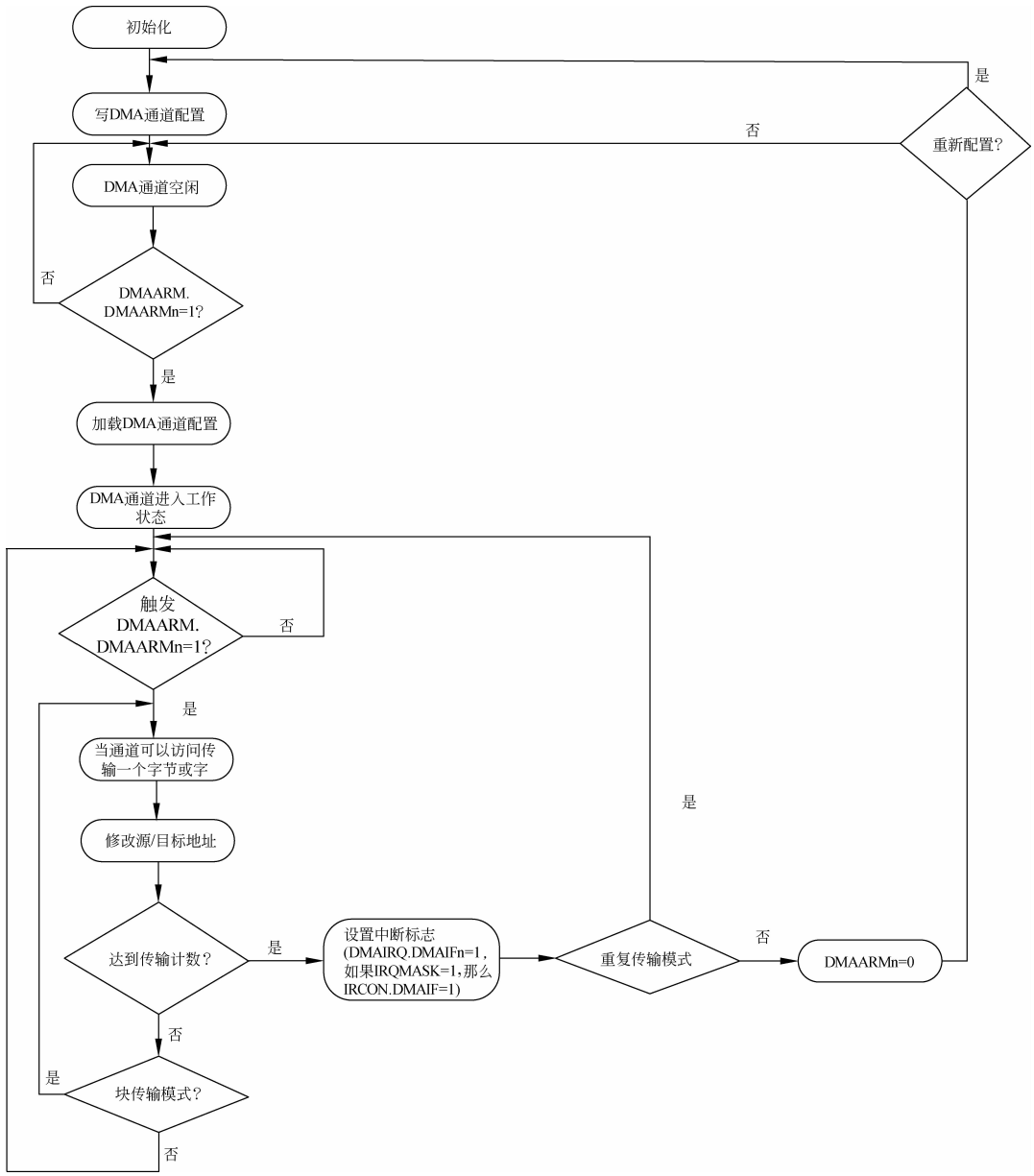


图 5-2 DMA 操作流程图

DMA 操作首先要对 DMA 进行初始化，然后写 DMA 通道配置。写之前首先判断 DMAARM.DMAARMn 寄存器是否为 1，如果不为 1，则重新判断 DMA 通道是否空闲；如果为 1，则加载 DMA 通道配置。DMA 通道进入工作状态之后判断是否触发 DMA，如果 DMA 被触发则配置 DMA 访问配置（以字或字节传输）、修改源地址或目的地址。在传输过程中如果达到传输计数，则设置中断标志。如果没有达到传输计数，则判断是否为块传输模式。如果是块传输模式，则重新配置 DMA 通道访问；如果不是块传输模式，则重新触

发 DMA。

如果设置了中断标志，则判断是否为重复传输模式，如果是，则重新触发 DMA 传输；如果没有设置重复传输模式，则设置 DMAARM 寄存器的第 0 位。然后判断是否要重新配置，如果需要重新配置，则需要重新写 DMA 通道配置；如果不需要重新配置，则判断 DMA 通道是否空闲。

在 DMA 操作中涉及 DMA 的控制寄存器，有 DMA 通道进入工作状态寄存器 DMAARM、DMA 通道开始请求和状态寄存器 DMAREQ、DMA 通道 0 配置地址寄存器高字节 DMA0CFGH 和 DMA 通道 0 配置地址寄存器低字节 DMA0CFGL、DMA 配置通道 1～4 的地址寄存器高字节 DMA1CFGH 和 DMA 配置通道 1～4 的地址寄存器低字节 DMA1CFGL、DMA 中断标志寄存器 DMAIRQ。各个寄存器的具体使用如下所述。

DMA 通道工作状态寄存器 DMAARM 负责启动或关闭 DMA 的运行以及选择 DMA 工作状态通道。DMAARM 寄存器如表 5-7 所示。

表 5-7　DMA 通道工作状态寄存器 DMAARM

位	名称	复位	R/W	描　　述
7	ABORT	0	R0/W	DMA 停止。此位是用来停止正在进行的 DMA 传输。通过设置相应的 DMAARM 位为 1，写 1 到该位停止所有选择的通道 0：正常运行 1：停止所有选择的通道
6～5	--	00	R/W	保留
4	DMAARM4	0	R/W1	DMA 进入工作状态通道 4 为了任何 DMA 传输能够在该通道上发生，该位必须置 1。对于非重复传输模式，一旦完成传送，该位自动清 0
3	DMAARM3	0	R/W1	DMA 进入工作状态通道 3 为了任何 DMA 传输能够在该通道上发生，该位必须置 1。对于非重复传输模式，一旦完成传送，该位自动清 0
2	DAMARM2	0	R/W1	DMA 进入工作状态通道 2 为了任何 DMA 传输能够在该通道上发生，该位必须置 1。对于非重复传输模式，一旦完成传送，该位自动清 0
1	DMAARM1	0	R/W1	DMA 进入工作状态通道 1 为了任何 DMA 传输能够在该通道上发生，该位必须置 1。对于非重复传输模式，一旦完成传送，该位自动清 0
0	DMAARM0	0	R/W1	DMA 进入工作状态通道 0 为了任何 DMA 传输能够在该通道上发生，该位必须置 1。对于非重复传输模式，一旦完成传送，该位自动清 0

- DMAARM 寄存器的第 7 位负责关闭或启动 DMA 的运行，如果此位设置为 0，DMA 正常运行；如果设置为 1，停止所有选择的运行通道。
- DMAARM 寄存器的第 6～5 位为保留位，暂时不用。
- DMAARM 的第 4～0 位负责设置 DMA 通道 4～0 的工作状态。当 DMA 运行时，此位必须设置为 1，当传送完毕后此位自动清 0。

DMA 通道开始请求和状态寄存器 DMAREQ 负责选择 DMA 的传输通道，当设置为 1 时则激活 DMA 通道，当传输开始时则清除此位。DMAREQ 寄存器如表 5-8 所示。

表 5-8 DMA 通道开始请求和状态 DMAREQ

位	名称	复位	R/W	描 述
7～5	--	000	R0	保留
4	DMAREQ4	0	R/W1 H0	DMA 传输请求，通道 4 当设置为 1 时，激活 DMA 通道（与一个触发事件具有相同的效果）。当 DMA 传输开始清除该位
3	DMAREQ3	0	R/W0 H0	DMA 传输请求，通道 3 当设置为 1 时，激活 DMA 通道（与一个触发事件具有相同的效果）。当 DMA 传输开始清除该位
2	DAMREQ2	0	R/W0 H0	DMA 传输请求，通道 2 当设置为 1 时，激活 DMA 通道（与一个触发事件具有相同的效果）。当 DMA 传输开始清除该位
1	DMAREQ1	0	R/W0 H0	DMA 传输请求，通道 1 当设置为 1 时，激活 DMA 通道（与一个触发事件具有相同的效果）。当 DMA 传输开始清除该位
0	DMAREQ0	0	R/W0 H0	DMA 传输请求，通道 0 当设置为 1 时，激活 DMA 通道（与一个触发事件具有相同的效果）。当 DMA 传输开始清除该位

DMA 通道 0 配置地址寄存器高字节 DMA0CFGH 和寄存器低字节 DMA0CFGL 用来存放 DMA 传输数据的开始地址，DMA0CFGH 和 DMA0CFGL 寄存器如表 5-9 和表 5-10 所示。

表 5-9 DMA 通道 0 配置地址高字节寄存器 DMA0CFGH

位	名称	复位	R/W	描 述
7～0	DMA0CFGH[15:8]	0x00	R/W	DMA 通道 0 配置地址，高位字节

表 5-10 DMA 通道 0 配置地址低字节寄存器 DMA0CFGL

位	名称	复位	R/W	描 述
7～0	DMA0CFGL[7:0]	0x00	R/W	DMA 通道 0 配置地址，低位字节

DMA 通道 1～4 配置地址高字节寄存器 DMAxCFGH 和低字节寄存器 DMAxCFGL（x 表示 1、2、3、4）是用来存放 DMA 传输数据的开始地址，以通道 1 为例，DMA1CFGH 和 DMA1CFGL 寄存器如表 5-11 和表 5-12 所示。

表 5-11 DMA 通道 1 配置地址高字节寄存器 DMA1CFGH

位	名称	复位	R/W	描 述
7～0	DMA1CFGH[15:8]	0x00	R/W	DMA 通道 1～4 配置地址，高位字节

表 5-12　DMA 通道 1 配置地址高字节寄存器 DMA1CFGL

位	名称	复位	R/W	描　　述
7～0	DMA1CFGL[7:0]	0x00	R/W	DMA 通道 1～4 配置地址,低位字节

DMA 中断标志寄存器 DMAIRQ 主要功能是判断 DMA 通道中断标志,DMAIRQ 寄存器的第 7～5 位保留暂时不用,第 4～0 位分别用于判断 DMA 通道 4～0 中断标志。DMAIRQ 寄存器如表 5-13 所示。

表 5-13　DMA 中断标志寄存器 DMAIRQ

位	名称	复位	R/W	描　　述
7～5	--	000	R/W0	保留
4	DMAIF4	0	R/W0	DMA 通道 4 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决
3	DMAIF3	0	R/W0	DMA 通道 3 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决
2	DAMIF2	0	R/W0	DMA 通道 2 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决
1	DMAIF1	0	R/W0	DMA 通道 1 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决
0	DMAIF0	0	R/W0	DMA 通道 0 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决

5.2.3　DMA 配置

DMA 配置包括 DMA 配置参数和 DMA 配置安装。DMA 配置参数包括源地址、目的地址、传送长度、可变长度(VLEN)设置、优先级别、触发事件、源地址和目标增量、传送模式、字节传送或字传送、中断屏蔽和 M8。

- 源地址：DMA 通道要读的数据的首地址。源地址可以是 XDATA 的任何地址。
- 目标地址：DMA 通道从源地址读出数据写入区域的首地址。用户必须确认该目标地址可写。目标地址可以是 XDATA 的任何地址。
- 传送长度：在 DMA 通道重新进入工作状态或者接触工作状态之前,以及警告 CPU 即将有中断请求到来之前所要传送的长度。
- 可变长度：DMA 通道可以利用源数据中的第一个字节或字(对于字使用[12:0]位)作为传送长度来进行可变长度传输。
- 优先级别：DMA 通道的 DMA 传送的优先级别与 CPU、其他 DMA 通道和访问端口相关,用于判定同时发生的多个内部存储器请求中的哪一个优先级别最高,以及

DMA 存储器存取的优先级别是否超过同时发生的 CPU 存储器存取的优先级别，DMA 优先级别有 3 级：高级、普通级和低级。

- 触发事件：所有 DMA 传输通过 DMA 触发事件产生。这个触发可以启动一个 DMA 块传输或单个 DMA 传输，DMA 通道可以通过设置指定 DMAREQ. DMAREQx 标志来触发。
- 源地址和目标地址增量：源地址和目标地址可以设置为增加或减少，或不改变，有 4 种情况：增量 0、增量 1、增量 2 和增量-1。
- 传送模式：传输模式决定当 DMA 通道开始传输数据时是如何工作的，包括单一模式、块模式、重复的单一模式和重复的块模式。
- 字节传送或字传送：确定每个 DMA 传输是 8 位字或 16 位字。
- 中断屏蔽：在完成 DMA 通道传送时，产生一个中断请求。这个中断屏蔽位控制中断产生是使能还是禁用。
- 模式 8(M8)设置：字节传送时，用来决定是采用 7 位还是 8 位的字长来传送数据。此模式仅用于字节传送模式。

DMA 配置安装包括 DMA 参数的配置和 DMA 地址的配置。其中 DMA 参数的配置是通过向寄存器写入特殊的 DMA 配置数据结构来配置的。DMA 配置数据结构由 8 个字节组成。DMA 配置数据结构如表 5-14 所示。

表 5-14 DMA 配置数据结构

字节偏移量	位	名称	描 述
0	7：0	SRCADDR[15：8]	DMA 通道源地址，高位
1	7：0	SRCADDR[7：0]	DMA 通道源地址，低位
2	7：0	DESTADDR[15：8]	DMA 通道目的地址，高位
3	7：0	DESTADDR[7：0]	DMA 通道目的地址，低位
4	7：5	VLEN[2：0]	可变长度传输模式，在字模式中，第一个字的 12：0 位被认为是传送长度的 000：采用 LEN 作为传送长度 001：传送的字节/字的长度由第一个字节/字+1 指定的长度(上限由 LEN 指定的最大值)。因此，传输长度不包括字节/字的长度 010：传送通过第一个字节/字指定的字节/字的长度(上限到由 LEN 指定的最大值)。因此传输长度包括字节/字的长度 011：传送通过第一个字节/字指定的字节/字的长度+2(上限到由 LEN 指定的最大值)。因此，传输长度不包括字节/字的长度 100：传送通过第一个字节/字指定的字节/字的长度+3(上限到由 LEN 指定的最大值)。因此，传输长度不包括字节/字的长度 101：保留 110：保留 111：使用 LEN 作为传输长度的备用

续表

字节偏移量	位	名称	描　　述
4	4：0	LEN[12:8]	DMA 通道的传送长度高位 当 VLEN 从 000 到 111 时采用最大允许长度。当处于 WORDSIZE 模式时，DMA 通道以字为单位，否则以字节为单位
5	7：0	LEN[7:0]	DMA 通道的传送长度低位 当 VLEN 从 000 到 111 时采用最大允许长度。当处于 WORDSIZE 模式时，DMA 通道数以字为单位，否则以字节为单位
6	6：5	TMOD[1:0]	DMA 通道传送模式 00：单个 01：块 10：重复单一 11：重复块
6	4：0	TRIG[4:0]	选择要使用的 DMA 触发 00000：无触发 00001：前一个 DMA 通道完成 00010-11110：选择触发源
7	7：6	SRCINC[1:0]	源地址递增模式(每次传送之后)： 00：0 字节/字 01：1 字节/字 10：2 字节/字 11：-1 字节/字
7	5：4	DESTINC[1:0]	目的地址递增模式(每次传送之后)： 00：0 字节/字 01：1 字节/字 10：2 字节/字 11：-1 字节/字
7	3	IRQMASK	该通道中断屏蔽 0：禁止中断发生 1：DMA 通道完成时使能中断发生
7	2	M8	采用 VLEN 的第 8 位模式作为传送单位长度：仅与应用在 WORDSIZE＝0 且 VLEN 从 000 到 111 时 0：采用所有 8 位作为传送长度 1：采用字节的低 7 位作为传送长度
7	1：0	PRIORITY[1:0]	DMA 通道的优先级别 00：低级 CPU 优先 01：保证级，DMA 至少在每秒一次尝试中优先 10：高级，DMA 优先 11：保留

DMA 配置安装一般在 C 语言中设计成结构体，其结构体如示例 5-4 所示。

【示例 5-4】 DMA 配置安装结构体

```
typedef  struct
{
   /* 源地址高 8 位 */
   unsigned char  SRCADDRH;
   /* 源地址低 8 位 */
   unsigned char  SRCADDRL;
   /* 目的地址高 8 位 */
   unsigned char  DESTADDRH;
   /* 目的地址低 8 位 */
   unsigned char  DESTADDRL;
   /* 长度域模式选择 */
   unsigned char  VLEN          :3;
   /* 传输长度高字节 */
   unsigned char  LENH          :5;
   /* 传输长度低字节 */
   unsigned char  LENL          :8;
   /* 字节或字传输 */
   unsigned char  WORDSIZE      :1;
   /* 传输模式选择 */
   unsigned char  TMODE         :2;
   /* 触发事件选择 */
   unsigned char  TRIG          :5;
   /* 源地址增量 : -1/0/1/2 */
   unsigned char  SRCINC        :2;
   /* 目的地址增量 : -1/0/1/2 */
   unsigned char  DESTINC       :2;
   /* 中断屏蔽 */
   unsigned char  IRQMASK       :1;
   /* 7 或 8bit 传输长度，仅在字节传输模式下适用 */
   unsigned char  M8            :1;
   /* 优先级 */
   unsigned char  PRIORITY      :2;
} DMA_CFG;
```

5.2.4 DMA 触发

DMA 触发可以通过设置 DMA 的触发源，来判定 DMA 通道会接受哪一个事件的触发。DMA 有 31 个触发源，包括定时器触发、I/O 控制器触发、定时器触发、串口的发送和接收触发、ADC 传输触发等。其触发源如表 5-15 所示。

表 5-15 DMA 触发源

DMA 触发器		功能单元	描 述
号码	名称		
0	NONE	DMA	没有触发器，设置 DMAREQ. DMAREQx 位开始传送
1	PREV	DMA	DMA 通道是通过完成前一个通道来触发的
2	T1_CH0	定时器 1	定时器 1，比较，通道 0
3	T1_CH1	定时器 1	定时器 1，比较，通道 1

续表

DMA 触发器		功能单元	描　　述
号码	名称		
4	T1_CH2	定时器 1	定时器 1,比较,通道 2
5	T2_EVENT1	定时器 2	定时器 2,事件脉冲 1
6	T2_EVENT2	定时器 2	定时器 2,事件脉冲 2
7	T3_CH0	定时器 3	定时器 3,比较,通道 0
8	T3_CH1	定时器 3	定时器 3,比较,通道 1
9	T4_CH0	定时器 4	定时器 4,比较,通道 0
10	T4_CH1	定时器 4	定时器 4,比较,通道 1
11	ST	睡眠定时器	睡眠定时器比较
12	IOC_0	I/O 控制器	端口 I/O 引脚输入转换
13	IOC_1	I/O 控制器	端口 1I/O 引脚输入转换
14	URX0	USART_0	USART 0 接收完成
15	UTX0	USART_0	USART 0 发送完成
16	URX1	USART_1	USART 1 接收完成
17	UTX1	USART_1	USART 1 发送完成
18	Flash	闪存控制器	写闪存数据完成
19	RADIO	无线模块	接收 RF 字节包
20	ADC_CHALL	ADC	ADC 结束一次转换,采样已经准备好
21	ADC_CH11	ADC	ADC 结束通道 0 的一次转换,采样已经准备好
22	ADC_CH21	ADC	ADC 结束通道 1 的一次转换,采样已经准备好
23	ADC_CH32	ADC	ADC 结束通道 2 的一次转换,采样已经准备好
24	ADC_CH42	ADC	ADC 结束通道 3 的一次转换,采样已经准备好
25	ADC_CH53	ADC	ADC 结束通道 4 的一次转换,采样已经准备好
26	ADC_CH63	ADC	ADC 结束通道 5 的一次转换,采样已经准备好
27	ADC_CH74	ADC	ADC 结束通道 6 的一次转换,采样已经准备好
28	ADC_CH84	ADC	ADC 结束通道 7 的一次转换,采样已经准备好
29	ENC_DW	AES	AES 加密处理器请求下载输入数据
30	ENC_UP	AES	AES 加密处理器请求上传输入数据
31	DBG_BW	调试接口	调试接口突发写操作

5.2.5　DMA 传输

本节以下内容将实现通过串口触发 DMA 传输实例,在串口触发 DMA 传输实例中需要做以下工作：DMA 初始化、串口初始化、串口传输、DMA 触发传输。

DMA 初始化：按照 DMA 配置安装结构体的结构对 DMA 进行初始化,首先对其进行源地址配置,将传输的源地址配置为需要传输的字符数组 a[]的地址,传输的目的地址设置为 X_U0DBUF；传输长度采用 LEN 作为传输长度；将数组 a[]的长度的高位设置为 LENH,低位设置为 LENL；选择字节传送；DMA 通道传送模式选用单个传送模式；DMA 触发方式设置为串口触发方式；设置源地址增量为 1,目的地址增量为 0；选择 8 位字节传送,并将 DMA 的优先级设置为高级；最后将 DMA 配置结构体的地址赋予寄存器。具体代

码如实例 5-3 DMA 初始化函数 DMA_Init()所示。

【实例 5-3】 DMA_Init()

```
void DMA_Init()
{
    /* 配置源地址 */
    dmaConfig.SRCADDRH = (unsigned char)((unsigned int)&a >> 8);
    dmaConfig.SRCADDRL = (unsigned char)((unsigned int)&a );
    /* 配置目的地址 */
    dmaConfig.DESTADDRH = (unsigned char)((unsigned int)&X_U0DBUF >> 8);
    dmaConfig.DESTADDRL = (unsigned char)((unsigned int)&X_U0DBUF);
    /* 选择 LEN  作为传送长度 */
    dmaConfig.VLEN = 0x00;
    /* 设置传输长度 */
    dmaConfig.LENH = (unsigned char)((unsigned int)sizeof(a)>>8);
    dmaConfig.LENL = (unsigned char)((unsigned int)sizeof(a));
    /* 选择字节 byte 传送 */
    dmaConfig.WORDSIZE = 0x00;
    /* 选择单个传送模式 */
    dmaConfig.TMODE = 0x00;
    /* 串口触发 */
    dmaConfig.TRIG  = 15;
    /* 源地址增量为 1 */
    dmaConfig.SRCINC = 0x01;
    /* 目的地址增量为 0 */
    dmaConfig.DESTINC = 0x00;
    /* 清除 DMA 中断标志 */
    dmaConfig.IRQMASK = 0x00;
    /* 选择 8 位长的字节来传送数据 */
    dmaConfig.M8 = 0x00;
    /* 传送优先级为高 */
    dmaConfig.PRIORITY = 0x02;
    /* 将配置结构体的首地址赋予相关 SFR */
    DMA0CFGH = (unsigned char)((unsigned int)&dmaConfig >>8);
    DMA0CFGL = (unsigned char)((unsigned int)&dmaConfig);
    asm("nop");
}
```

在 main 函数中首先调用了 LED 初始化函数、DMA 初始化函数、串口初始化函数，然后进行 DMA 传输配置：选择 DMA 通道 0 进行传输。具体代码如实例 5-3 main()所示。

【实例 5-3】 main()

```
void main( void )
{
  /* LED 初始化 */
  LED_init();
  /* DMA 初始化 */
  DMA_Init();
  /* 串口初始化 */
  initUART();
```

```
    while(1)
    {
      /* LED2 和 LED3 状态改变 */
      delay();
      LED2 = ~LED2;
      delay();
      LED3 = ~LED3;
      delay();
      /* 串口传输 string 字符数组 */
      UartTX_Send_String(string,12);
      /* 停止 DMA 所有通道进行传输 */
      DMAARM = 0x80;
      /* 启用 DMA 通道 0 进行传输 */
      DMAARM = 0x01;
      /* 清中断标志 */
      DMAIRQ = 0x00;
      /* DMA 通道 0 传送请求 */
      DMAREQ = 0x01;
      /* 等待 DMA 传送完成 */
      while(!(DMAIRQ&0x01));
    }
  }
```

头文件中包含了 CC2530 的头文件，DMA 配置结构体以及函数声明，具体代码如实例 5-3“头文件”所示。

【实例 5-3】 头文件

```
#include "ioCC2530.h"
typedef unsigned char       BYTE;
/* DMA 配置结构体
#pragma bitfields = reversed
typedef struct {
   BYTE SRCADDRH;
   BYTE SRCADDRL;
   BYTE DESTADDRH;
   BYTE DESTADDRL;
   BYTE VLEN           : 3;
   BYTE LENH           : 5;
   BYTE LENL           : 8;
   BYTE WORDSIZE       : 1;
   BYTE TMODE          : 2;
   BYTE TRIG           : 5;
   BYTE SRCINC         : 2;
   BYTE DESTINC        : 2;
   BYTE IRQMASK        : 1;
   BYTE M8             : 1;
   BYTE PRIORITY       : 2;
} DMA_DESC;
#pragma bitfields = default
```

```
DMA_DESC dmaConfig;
/* DMA 串口 */
#define DMATRIG_UTX0            15
/* DMA 配置源地址 */
unsigned char a[4] = "QST";
/* 串口传输字符数组 */
unsigned char string[] = "\nDMA USART :";
/* 函数声明 */
void delay();
void UartTX_Send_String(unsigned char *Data,int len);
void DMA_Init();   //DMA 初始化
```

由于 LED 初始化函数和串口初始化函数在前面已经介绍过，在此实例中不再赘述，将程序下载至 CC2530 单片机中，并连接串口调试助手，可以观察 DMA 串口传送数据，如图 5-3 所示。

图 5-3　DMA 串口触发

5.3　定时器

CC2530 有 5 个定时器、一个 16 位定时器(定时器 1)、两个 8 位定时器(定时器 3 和定时器 4)、一个用于休眠的定时器(睡眠定时器)和一个 MAC 定时器。本节将讲解 CC2530 定时器的使用。

5.3.1 定时器 1

定时器 1 是一个独立的 16 位定时器，支持定时和计数功能，有输入捕获、输出比较和 PWM 的功能。定时器 1 有 5 个独立的输出捕获和输入比较通道。每个通道使用一个 I/O 引脚。定时器 1 的主要功能如下：

- 5 个独立的捕获、比较通道；
- 上升沿、下降沿或任何边沿的输入捕获；
- 设置、清除或切换输出比较；
- 3 种运行模式：自由运行、模计数模式和正/倒计数操作模式；
- 可被 1、8、32 或 128 整除的时钟分频器；
- 在捕获/比较和最终计数上生成中断请求；
- 具有 DMA 触发功能。

定时器 1 带有一个 16 位计数器，其计数器的工作是在每个活动时钟边沿递增或递减。活动时钟周期由相应的寄存器来配置。定时器 1 的计数器有 4 种工作模式：自由运行模式、模计数模式、正计数/倒计数模式、通道控制模式。

1. 自由运行模式

自由运行模式下，计数器从 0x0000 开始，每个活动时钟边沿增加 1。当计数器达到 0xFFFF 会产生自动溢出，然后计数器重新载入 0x0000，继续递增计数，当达到最终计数值 0xFFFF 产生溢出。当产生溢出之后，相应的寄存器会自动产生溢出标志(后面将讲解定时器的寄存器使用)。自由运行模式如图 5-4 所示。

2. 模计数模式

定时器 1 运行在模模式下，16 位计数器从 0x0000 开始，每个活动时钟边沿增加 1。当计数器达到用户设定的溢出值 T1CC0 时(溢出值 T1CC0 可以通过设置相应的寄存器获得)，计数器将复位至 0x0000，由此可见，模计数模式可以用于周期不是 0xFFFF 的应用程序。然后周而复始的递增。如果定时器的计数器开始于用户设定的初始值时，最终的计数值将终止于 0xFFFF。如果产生溢出，相应的标志寄存器将会自动置 1。模计数模式运行过程如图 5-5 所示。

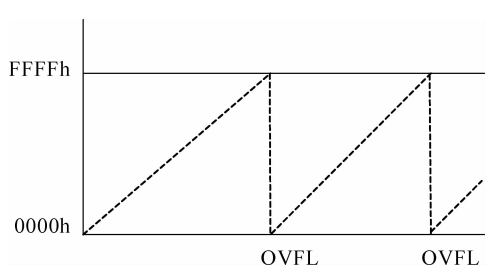


图 5-4 自由运行模式

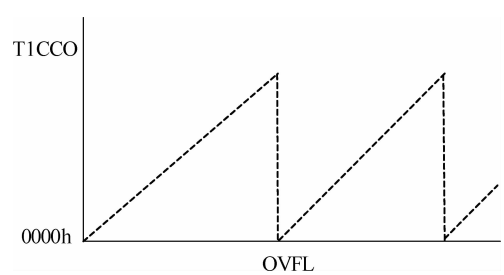


图 5-5 模模式

3. 正计数/倒计数模式

在正计数/倒计数模式下，计数器从 0x0000 开始，正计数直到达到设定值 T1CC0H：T1CC0L，计数器将倒计数至 0x0000。在此模式下计数器用于周期必须是对称输出脉冲而不是 0xFFFF 的应用程序，因此，在此模式下可以实现中心对称的 PWM 信号输出。在正计数/倒计数模式下如果设置了中断使能，当计数达到一定值时会产生中断。正计数/倒计数模式运行过程如图 5-6 所示。

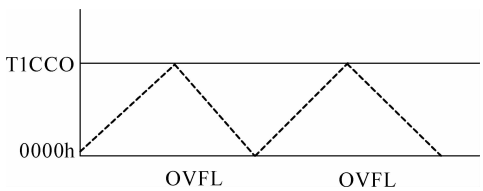


图 5-6 正计数/倒计数模式

4. 通道控制模式

通道模式控制由控制和状态寄存器 T1CCTL*n* 设置，包括输入捕获模式和输出比较模式。

- 输入捕获模式：当一个通道配置为输入捕获通道时，和该通道相关的 I/O 引脚配置为外设模式，并且通过寄存器配置为输入模式。在启动定时器之后，输入引脚的上升沿或下降沿或任何边沿都将触发一个捕获，将 16 位计数器的内容捕获至相关的寄存器中。
- 输出比较模式：在输出比较模式下，与通道相关的 I/O 引脚通过寄存器设置为输出模式。在定时器启动之后，将比较“计数器”和“通道比较寄存器”的内容。如果计数器和通道比较寄存器的数值相同，输出引脚将根据比较输出模式寄存器的设置进行相应的动作。

5. 定时器 1 寄存器

定时器 1 除了有独特的运行模式之外还可以产生定时器中断和定时器 DMA 触发。

- 定时器的中断由计数器、输入捕获事件和输出比较事件触发。当设置了中断寄存器时，就会产生一个中断。
- 定时器 1 的 DMA 触发方式有 3 种，即通道 0 比较、通道 1 比较和通道 2 比较触发，其中通道 3 比较和通道 4 比较不能触发 DMA。DMA 触发是通过定时器 1 相应的寄存器来设置的。

以下内容将讲解定时器的寄存器，定时器 1 有 7 个寄存器：定时器 1 计数高位寄存器 T1CNTH 和定时器 1 计数低位寄存器 T1CNTL、定时器 1 控制寄存器 T1CTL、定时器 1 状态寄存器 T1STAT、定时器 1 通道 *n* 捕获/比较控制寄存器 T1CCTL*n*、定时器 1 通道 *n* 捕获/比较高位寄存器 T1CC*n*H、定时器 1 通道 *n* 捕获/比较低位寄存器 T1CC*n*L(其中 *n* 的取值为 0、1、2、3、4)。

定时器1计数高位寄存器T1CNTH主要负责定时器1计数器的高8位，在读取数值时经常和定时器1计数低位寄存器T1CNTL一起使用，才能读出16位数值。T1CNTH寄存器和T1CNTL寄存器如表5-16和表5-17所示。

表5-16　定时器1计数高位寄存器T1CNTH

位	名称	复位	R/W	描　　述
7～0	CNT[15～8]	0x00	R	定时器计数器高位。包含在读取T1CNTL的时候缓存的16位计数器值的高8位

表5-17　定时器1计数低位寄存器T1CNTL

位	名称	复位	R/W	描　　述
7～0	CNT[7～0]	0x00	R/W	定时器计数器低字节。包括16位定时器计数器低字节。往该寄存器中写任何值，导致计数器被清零，初始化所有向通道的输出引脚

定时器1控制寄存器T1CTL主要功能是选择定时器1的工作模式和分频器频率划分，T1CTL寄存器具体设置如表5-18所示。

表5-18　定时器1控制寄存器T1CTL

位	名称	复位	R/W	描　　述
7～4	--	00000	R0	保留
3～2	DIV[1～0]		R/W	分频器划分值。产生主动的时钟边缘用来更新计数器，如下： 00：标记频率/1 01：标记频率/8 10：标记频率/32 11：标记频率/128
1～0	MODE[1～0]		R/W	选择定时器1模式。定时器操作模式通过下列方式选择： 00：暂停运行 01：自由运行，从0x0000到0xFFFF反复计数 10：模，从0x0000到T1CC0反复计数 11：正计数/倒计数，从0x0000到T1CC0反复计数且从T1CC0倒计数到0x0000

T1CTL寄存器的第3位和第2位负责分频器划分，分频从1分频至128分频；第1位和第0位为定时器1功能模式选择位MODE[1～0]，当MODE[1～0]设置为00时，为暂停运行定时器1模式；设置为01时，为自由运行模式；设置为10时，运行模式为模计数模式；当设置为11时，运行模式为正计数/倒计数模式。如果需要配置定时器1为自由运行模式，那么详细配置如示例5-5所示。

【示例 5-5】 定时器 1 自由运行模式

```
T1CTL = 0x01;
```

定时器 1 状态寄存器 T1STAT 只负责定时器 1 中断标志，包括定时器 1 计数器溢出中断标志和定时器 1 通道 0～4 的中断标志。T1STAT 寄存器如表 5-19 所示。

表 5-19 定时器 1 状态寄存器 T1STAT

位	名称	复位	R/W	描 述
7～6	--	00	R0	保留
5	OVFIF	0	R/W0	定时器 1 计数器溢出中断标志。当计数器在自由运行或模模式下达到最终计数值时置 1，当在正/倒计数模式下达到零时开始倒计数。写 1 没有影响
4	CH4IF	0	R/W0	定时器 1 通道 4 中断标志。当通道 4 中断条件发生时置位。写 1 没有影响
3	CH3IF	0	R/W0	定时器 1 通道 3 中断标志。当通道 3 中断条件发生时设置。写 1 没有影响
2	CH2IF	0	R/W0	定时器 1 通道 2 中断标志。当通道 2 中断条件发生时设置。写 1 没有影响
1	CH1IF	0	R/W0	定时器 1 通道 1 中断标志。当通道 1 中断条件发生时设置。写 1 没有影响
0	CH0IF	0	R/W0	定时器 1 通道 0 中断标志。当通道 0 中断条件发生时设置。写 1 没有影响

- T1STAT 寄存器的第 5 位负责定时器 1 计数器溢出中断标志，当发生定时器 1 计数器溢出中断之后，此位自动设置为 1。
- T1STAT 寄存器的第 4 位负责定时器 1 通道 4 中断标志，当发生定时器 1 通道 4 中断发生之后，此位自动设置为 1。
- T1STAT 寄存器的第 3 位负责定时器 1 通道 3 中断标志，当发生定时器 1 通道 3 中断发生之后，此位自动设置为 1。
- T1STAT 寄存器的第 2 位负责定时器 1 通道 2 中断标志，当发生定时器 1 通道 2 中断发生之后，此位自动设置为 1。
- T1STAT 寄存器的第 1 位负责定时器 1 通道 1 中断标志，当发生定时器 1 通道 1 中断发生之后，此位自动设置为 1。
- T1STAT 寄存器的第 0 位负责定时器 1 通道 0 中断标志，当发生定时器 1 通道 0 中断发生之后，此位自动设置为 1。

T1CCTL*n*(其中 *n* 的取值为 0,1,2,3,4)主要负责定时器 1 通道 0～4 的中断设置和比较/捕获模式设置。

下面以 T1CCTL1 为例来讲解 T1CCTL*n* 寄存器的使用。T1CCTL1 寄存器的具体设置如表 5-20 所示。

表 5-20　定时器 1 通道 0 捕获/比较控制寄存器 T1CCTL1

位	名称	复位	R/W	描　　述
7	RFIRQ	0	R/W	当设置为 1 时，使用 RF 中断捕获，而非常规的捕获输入
6	IM	1	R/W	通道 1 中断屏蔽设置，当设置为 1 时中断请求产生
5～3	CMP	000	R/W	通道 1 比较模式选择，当定时器值等于在 T1CC1 的比较值时选择输出操作 000：在比较设置输出 001：在比较清除输出 010：在比较切换输出 011：在向上比较设置输出，在 0 清除 100：在向上比较清除输出，在 0 清除 101：当等于 T1CC0 时清除，当等于 T1CC1 时设置 110：当等于 T1CC0 时设置，当等于 T1CC1 时清除 111：初始化输出引脚
2	MODE	0	R/W	定时器 1 通道 1 捕获/比较模式选择 0：捕获模式 1：比较模式
1～0	CAP	00	R/W	通道 1 捕获模式选择 00：未捕获 01：上升沿捕获 10：下降沿捕获 11：所有沿捕获

- T1CCTL1 的第 7 位为 RF 中断捕获设置，当设置为 1 时开启 RF 中断捕获，当设置为 0 时禁止 RF 中断捕获。
- T1CCTL1 的第 6 位为通道 1 中断设置，当设置为 1 时开启定时器 1 通道 1 中断请求，当设置为 0 时，禁止中断请求。
- T1CCTL1 的第 5～3 位为定时器 1 的通道 1 比较模式选择，当定时器值等于寄存器 T1CC1 设置的比较值时，选择以哪种方式输出。当设置为 000 时，在设置为比较状态中输出；当设置为 001 时，在设置为比较清除状态时输出；当设置为 010 时，在设置为比较切换状态时输出。当设置为 011 时，在设置为向上比较状态时输出；当设置为 100 时，在设置为向上比较清楚状态下输出；在设置为 101 时，在等于 T1CC0 中的值时清除，在等于 T1CC1 时设置；在设置为 110 时，在等于 T1CC0 时设置，在等于 T1CC1 时清除；当设置为 111 时，主要功能是初始化输出引脚。
- T1CCTL1 的第 2 位设置定时器 1 的通道 1 的捕获和比较模式，当设置为 0 时，为捕获模式；当设置为 1 时为比较模式。
- T1CCTL1 的第 1～0 位设置定时器 1 通道 1 的捕获模式。当设置为 00 时，未设置捕获模式；当设置为 01 时为上升沿触发捕获；当设置为 10 时为下降沿触发捕获；当设置为 11 时，所有沿都可以触发捕获。

定时器 1 通道 n 捕获/比较高位寄存器 T1CCnH 和捕获/比较低位寄存器 T1CCnL 主要功能是存储捕获/比较值。以定时器 1 的通道 1 的 T1CC1H 和 T1CCH1L 为例，寄存器 T1CC1H 是用来存放通道 1 捕获/比较值的高 8 位；寄存器 T1CC1L 是用来存放通道 1 捕

获比较值的低 8 位。寄存器 T1CC1H 和 T1CC1L 如表 5-21 和表 5-22 所示。

表 5-21 定时器 1 通道 1 捕获/比较高位寄存器 T1CC1H

位	名称	复位	R/W	描 述
7～0	T1CC1[15～8]	0x00	R/W	定时器 1 通道 1 捕获/比较值,高位字节。当 T1CCTL0.MODE=1(比较模式)时,寄存器写会更新 T1CC1[15:0]的值导致比较延迟,直至 T1CNT=0x0000

表 5-22 定时器 1 通道 1 捕获/比较低位寄存器 T1CC1L

位	名称	复位	R/W	描 述
7～0	T1CC1[7～0]	0x00	R/W	定时器 1 通道 1 捕获/比较值,低位字节。写到该寄存器的数据被存储在一个缓存中,不写入 T1CC1[7～0],之后与 T1CC1H 一起写入生效

以下实例将讲解定时器 1 在模计数模式的工作方式。定时器 1 在模计数模式下工作,首先要设置比较值,当计数器达到设定的比较值时就会产生中断。具体实现如实例 5-4 所示。

头文件包含了 CC2530 的头文件,定时器 1 模计数模式初始化函数声明,以及 LED 的引脚连接方式,具体代码如实例 5-4“头文件”所示。

【实例 5-4】 头文件

```
#include <ioCC2530.h>
#define uint8  unsigned char
#define uint16 unsigned int
#define LED1 P1_0
/* 定时器 1 模模式初始化函数声明 */
void initial(void);
```

初始化函数首先设置 LED1 为关闭状态,然后设置时钟运行方式、最后设置定时器 1 模计数模式运行方式。定时器 1 模计数模式的设置方式如下:

- 设置定时器 1 为 128 分频;
- 设置 T1CC0L 载入定时器 1 的初值;
- 设置捕获/比较通道为比较模式,用于触发中断;
- 打开定时器中断与总中断。

具体代码详见实例 5-4 initial()函数。

【实例 5-4】 initial()

```
void initial(void)
{
  /* 设置 P1.0 为输出模式 */
  P1DIR |= 0x01;
  /* 关闭 LED1 */
  LED1 = 1;
  /* 选择外部石英晶振 */
  CLKCONCMD &= ~0x40;
  /* 等待晶振稳定 */
```

```
  while(!(SLEEPSTA & 0x40));
  /* TICHSPD 二分频,CLKSPD 不分频 */
  CLKCONCMD &= ~0x47;
  /* 关闭 RC 振荡器 */
  SLEEPCMD |= 0x04;
  /* 设置定时器 T1,128 分频,模计数模式,从 0 计数到 T1CC0 */
  T1CTL |= 0x0E;
  /* 装入定时器初值(比较值) */
  T1CC0L = 62500 % 256;
  T1CC0H = 62500/256;
  /* 设置捕获比较通道 0 为比较模式,用以触发中断 */
  T1CCTL0 |^ = 0x04;
  /* 使能 Timer1 中断 */
  T1IE = 1;
  /* 开启总中断 */
  EA = 1;
}
```

主函数中在调用定时器 1 模计数模式的初始化函数之后,接下来的工作只需要等待中断的发生即可。具体代码见实例 5-4 main()函数。

【实例 5-4】 main()

```
void main(void)
{
  initial();
  while(1)
  {

  }
}
```

当中断发生时改变 LED1 的状态,具体代码如实例 5-4 T1_ISR()所示。

【实例 5-4】 T1_ISR()

```
#pragma vector = T1_VECTOR
__interrupt void T1_ISR(void)
{
  LED1 = !LED1;
}
```

5.3.2 定时器 3 和定时器 4

定时器 3 和定时器 4 是两个 8 位定时器,每个定时器有两个独立的比较通道,每个通道上使用一个 I/O 引脚。其定时器 3 和定时器 4 的主要特点如下:

- 每个定时器有两个捕获/比较通道;
- 具有设置、清除或切换输出比较的功能;
- 可以设置时钟分频器,可以被 1、2、4、8、32、64、128 整除;
- 在每次捕获/比较和最终计数事件发生时可以产生中断请求;
- 具有 DMA 触发功能。

本节将讲解定时器3和定时器4的运行模式、通道模式控制、定时器中断、DMA触发和定时器3、定时器4寄存器。

1. 运行模式

定时器3与定时器4分别具有一个8位的计数器，提供定时、计数功能，定时器3和定时器4所有的定时功能都是通过该计数器来实现的。计数器有4种运行模式：自由运行模式、倒计数模式、模计数模式和正/倒计数模式。

- 自由运行模式：在自由运行模式下，定时器的计数器从0x00开始，在每个时钟活动的边沿递增，当计数器达到0xFF，计数器将重新载入0x00。如果设置了中断，当达到最终计数值0xFF时，将会产生一个中断请求。
- 倒计数模式：在倒计数模式下，定时器启动之后，计数器载入预先设置好的数值，通过计数器倒计时，当达到0x00时，会产生一个中断标志。如果设置了中断，就会产生一个中断申请。
- 模计数模式：当定时器运行在模计数模式时，8位计数器在0x00启动，每个活动时钟边沿递增。当计数器达到相应寄存器所设置的最终计数值时，计数器复位至0x00，并继续递增。如果设置了中断，还会产生一个中断请求。模计数模式还可以用于周期不是0xFF的应用程序。
- 正/倒计数模式：在正/倒计数定时器模式下，计数器从0x00开始正计数，直到达到设置的数值，然后自动启用倒计数，直到达到0x00。

2. 定时器通道模式控制

对于定时器3和定时器4的通道0和通道1，每个通道的模式是由控制和状态寄存器来控制的，设置模式包括输入捕获模式和输出比较模式。

1）输入捕获模式

当通道配置为输入捕获通道，通道相关的I/O引脚配置为一个输入。定时器启动之后，输入引脚上的一个上升沿、下降沿或任何边沿都会触发一个捕获，即捕获8位计数器内容到相关的捕获寄存器中，因此定时器能够捕获一个外部事件发生的时间。通道输入引脚与内部系统时钟是同步的。因此输入引脚上的脉冲的最小持续时间必须大于系统时钟周期。当发生一个捕获且设置了相应的中断，输入捕获产生时，就会产生一个中断请求。

2）输出比较模式

在输出比较模式下，与该通道相关的I/O引脚必须设置为输出。定时器启动之后，将对比计数器的内容和通道比较寄存器的内容。如果计数器的内容等于比较寄存器的内容，根据比较输出模式的设置，输出引脚将被设置。

3. 定时器中断

定时器3和定时器4各有一个中断向量，当中断事件发生时，将产生一个中断请求，中断事件由以下几种触发方式。

- 计数器达到最终计数值；
- 比较事件；

- 捕获事件。

4. DMA 触发

定时器 3 和定时器 4 有两个相关的 DMA 触发，DMA 触发是通过定时器通道的捕获/比较事件来触发的。定时器 3 和定时器 4 的 DMA 触发事件有以下几种：

- 定时器 3 通道 0 捕获/比较触发；
- 定时器 3 通道 1 捕获/比较触发；
- 定时器 4 通道 0 捕获/比较触发；
- 定时器 4 通道 1 捕获/比较触发。

5. 定时器 3 和定时器 4 控制寄存器

定时器 3 和定时器 4 相关寄存器有计数器、定时器控制寄存器、定时器通道捕获/比较控制寄存器、定时器通道捕获比较值寄存器。以上寄存器，定时器 3 和定时器 4 基本相同。

以下内容以定时器 3 寄存器为例来讲解定时器 3 和定时器 4 寄存器的使用。定时器 3 计数器的主要功能是计数，此寄存器存放 8 位计数器的当前值。定时器 3 计数器 T3CNT 如表 5-23 所示。

表 5-23　定时器 3 计数器 T3CNT

位	名称	复位	R/W	描　　述
7～0	CNT[7～0]	0x00	R/W	定时器计数字节，包含 8 位计数器当前值

定时器 3 控制寄存器 T3CTL 主要负责对定时器 3 的分频器的划分、停止/运行、中断设置、计数器清除和选择定时器 3 的功能模式。定时器 3 寄存器 T3CTL 具体设置如表 5-24 所示。

表 5-24　定时器 3 控制寄存器 T3CTL

位	名称	复位	R/W	描　　述
7～5	DIV[2～0]	000	R/W	分频器划分值。产生有效时钟沿用于来自 CLKCON.TICKSPD 的定时器时钟，如下： 000：标记频率/1 001：标记频率/2 010：标记频率 4 011：标记频率/8 100：标记频率/16 101：标记频率/32 110：标记频率/64 111：标记频率/128
4	START	0	R/W	启动定时器。正常运行时设置，暂停时清除
3	OVFIM	1	R/W0	溢出中断屏蔽 0：中断禁止 1：中断使能
2	CLR	0	R0/W1	清除计数器。写 1 到 CLR 复位计数器到 0x00，并初始化相关通道所有的输出引脚。总是读作 0

续表

位	名称	复位	R/W	描 述
1～0	MODE［1～0］	00	R/W	选择定时器3模式。定时器操作模式通过下列方式选择： 00：自由运行，从0x00到0xFF反复计数 01：倒计数，从T3CC0到0x00计数 10：模计数，从0x0000到T1CC0反复计数 11：正计数/倒计数，从0x00到T3CC0反复计数且从T1CC0倒计数到0x00

- T3CTL寄存器的第7～5位负责定时器3分频器的划分，标记率在1～128分频之间。当设置为000时，分频值为1；当设置为001时，分频值为2；当设置为010时，分频值为4；当设置为011时，分频值为8；当设置为100时，分频值为16；当设置为101时，分频值为32；当设置为110时，分频值为64；当设置为111时，分频值为128；
- T3CTL寄存器的第4位负责启动定时器。当设置为1时，正常运行寄存器；当设置为0时，暂停运行寄存器。
- T3CTL的第3位主要负责中断设置。当该位设置为1时，中断使能；当该位设置为0时，中断禁止。
- T3CTL的第2位主要功能是清除计数器，当该位设置为1时，复位寄存器到0x00。
- T3CTL寄存器的第1～0位负责设置定时器3的运行模式。当设置为00时为自由运行模式；当设置为01时为倒计数运行模式；当设置为10时，为模计数模式；当设置为11时，为正计数/倒计数模式。

定时器3通道捕获/比较控制寄存器T3CCTL*n*(其中*n*的取值为0或1)，主要负责通道的中断设置、通道比较输出模式选择、通道模式选择和捕获模式选择。

以下内容以定时器3通道0捕获/比较控制寄存器T3CCTL0为例来讲解T3CCTL0寄存器的使用。定时器3通道0捕获/比较控制寄存器T3CCTL0具体设置如表5-25所示。

表5-25 定时器3通道0捕获/比较控制寄存器T3CCTL0

位	名称	复位	R/W	描 述
7	--	0	R0	未使用
6	IM	1	R/W	通道中断屏蔽 0：中断禁止 1：中断使能
5～3	CMP[2:0]	000	R/W	通道比较输出模式选择。当时钟值与T3CC0中的比较值相等时输出特定的操作 000：当时钟值与T3CC0中的比较值相等时设置输出 001：当时钟值与T3CC0中的比较值相等时清除输出 010：当时钟值与T3CC0中的比较值相等时切换输出 011：在比较正计数时设置输出，当定时器值为0时清除 100：在比较正计数时清除输出，当定时器值为0时设置 101：当时钟值与T3CC0中的比较值相等时设置输出，当T3CC0为0xFF时清除 110：当T3CC0为0x00时设置输出模式，当时钟值与T3CC0中的比较值相等时清除输出 111：初始化输出引脚。CMP[2:0]不变

续表

位	名称	复位	R/W	描　　述
2	MODE	0	R/W	通道模式，选择定时器 3 通道 0 捕获或者比较模式 0：捕获模式 1：比较模式
1～0	CAP	00	R/W	捕获模式选择 00：无捕获 01：在上升沿捕获 10：在下降沿捕获 11：在两个边沿都捕获

- T3CCTL0 寄存器的第 6 位负责通道中断设置。当该位设置为 0 时，禁止通道中断发生；当该位设置为 1 时，允许通道中断发生。
- T3CCTL0 寄存器的第 5～3 位负责选择通道比较输出模式。
- T3CCTL0 寄存器的第 2 位负责捕获/比较模式选择，当设置为 0 时为捕获模式；当设置为 1 时为比较模式。
- T3CCTL0 寄存器的第 1～0 位负责选择捕获模式。当设置为 00 时，没有捕获发生；当设置为 01 时，在上升沿捕获；当设置为 10 时，在下降沿捕获；当设置为 11 时，在任何边沿都可以捕获。

定时器 3 通道捕获/比较值寄存器 T3CC*n*（其中 *n* 的取值为 0 或 1）主要功能是设置定时器捕获/比较数值，以通道 0 为例，定时器 3 通道捕获/比较值寄存器 T3CC0 的具体设置如表 5-26 所示。

表 5-26　定时器 3 通道 0 捕获/比较值寄存器 T3CC0

位	名称	复位	R/W	描　　述
7～0	VAL[7～0]	0	R/W	定时器捕获比较通道 0 值。当 T3CCTL0.MODE＝1（比较模式）时写该寄存器会导致 T3CC0.VAL[7:0]更新写入值延迟到 T3CNT.CNT[7:0]＝0x00

定时器中断标志寄存器 TIMIF 负责判断定时器 1、定时器 3 和定时器 4 的中断标志，具体设置如表 5-27 所示。

表 5-27　定时器 1/3/4 中断标志寄存器 TIMIF

位	名称	复位	R/W	描　　述
7	--	0	R0	保留
6	T1OVFIM	1	R/W	定时器 1 溢出中断屏蔽
5	T4CH1IF	0	R/W0	定时器 4 通道 1 中断标志 0：无中断发生 1：发生中断
4	T4CH0IF	0	R/W0	定时器 4 通道 0 中断标志 0：无中断发生 1：发生中断

续表

位	名称	复位	R/W	描 述
3	T4OVFIF	0	R/W0	定时器 4 溢出中断标志 0：无中断发生 1：发生中断
2	T3CH1IF	0	R/W0	定时器 3 通道 1 中断标志 0：无中断发生 1：发生中断
1	T3CH0IF	0	R/W0	定时器 3 通道 0 中断标志 0：无中断发生 1：发生中断
0	T3OVFIF	0	R/W0	定时器 3 溢出中断标志 0：无中断发生 1：发生中断

- TIMIF 寄存器的第 6 位用于定时器 1 溢出中断，默认值为 1，当定时器 1 计数器达到设定的溢出值时，即发生溢出中断。
- TIMIF 寄存器的第 5 位用于判断定时器 4 通道 1 是否产生中断。当产生中断时，此位自动置 1；若无中断产生，则此位为 0。
- TIMIF 寄存器的第 4 位用于判断定时器 4 通道 0 是否产生中断。当产生中断时，此位自动置 1；若无中断产生，则此位为 0。
- TIMIF 寄存器的第 3 位用于判断定时器 4 溢出中断是否产生。当产生中断时，此位自动置 1；若无中断产生，则此位为 0。
- TIMIF 寄存器的第 2 位用于判断定时器 3 通道 1 是否产生中断。当产生中断时，此位自动置 1；若无中断产生，则此位为 0。
- TIMIF 寄存器的第 1 位用于判断定时器 3 通道 0 是否产生中断。当产生中断时，此位自动置 1；若无中断产生，则此位为 0。
- TIMIF 寄存器的第 3 位用于判断定时器 3 溢出中断是否产生。当产生中断时，此位自动置 1；若无中断产生，则此位为 0。

以下内容将实现实例 5-5 定时器 3 模计数模式溢出中断控制 LED 闪烁。此实例由 3 部分组成：定时器 3 与 LED 初始化函数、主函数和中断函数。

定时器 3 与 LED 初始化函数，首先将 LED1 和 LED2 初始化为点亮状态，其次初始化 T3，将运行模式设置为模计数模式，并装载 T3CC0 的初值，打开中断，当计数器的值达到装载的比较值后将会产生中断。具体代码如实例 5-5 Init_T3()所示。

【实例 5-5】 Init_T3()

```
#include <ioCC2530.h>
#define LED1 P1_0
#define LED2 P1_1
#define uchar unsigned char
/*定义全局变量：溢出中断次数 counter*/
int counter = 0;
```

```
/* 定时器 3 与 LED 初始化 */
void Init_T3(void)
{
    /* P1.0 和 P1.1 都设为输出 */
    P1DIR = 0x03;
    /* 打开 LED1 */
    LED1 = 1;
    /* 打开 LED2 *
    LED2 = 1;
    /* 清除计数器且设置定时器 3 的模式为模计数模式 */
    T3CTL  = 0x06;
    /* 初始化 T3 */
    T3CCTL0 = 0x00;
    T3CC0  = 0x00;
    T3CCTL1 = 0x00;
    T3CC1  = 0x00;
    /* 开 T3 中断 */
    T3CTL | = 0x08;
    /* 开中断 */
    EA = 1;
    /* 打开 T3 中断 */
    T3IE = 1;
    /* T3CTL | = 0X80; 时钟 16 分频 */
    T3CTL| = 0x80;
    /* 自动重装 00 -> 0xff */
    T3CTL & = ~0X03;
    /* 定时器计数字节 */
    T3CC0 = 0Xf0;
    /* 启动定时器 3 */
    T3CTL | = 0X10;
};
```

主函数中调用初始化函数并等待中断，具体代码如实例 5-5 main()所示。

【实例 5-5】 main()

```
void main(void)
{
  /* 初始化 */
  Init_T3();
  /* 关闭 LED2 */
  LED2 = 0;
  /* 等待中断 */
  while(1);
}
```

进入中断后，在中断函数中首先要消除中断标志，并且对中断次数进行计数，当中断次数达到 2000 次时，LED1 和 LED2 状态改变，具体代码如实例 5-5 T3_ISR ()所示。

【实例 5-5】 T3_ISR()

```
#pragma vector = T3_VECTOR
__interrupt void T3_ISR(void)
{
    /* 消中断标志,可不清中断标志,硬件自动完成 */
    IRCON = 0x00;
    /* 200 次中断 LED 闪烁一轮 */
    if(counter < 2000)
    {
        counter++;
    }
    else
    {
          /* 计数清零 */
          counter = 0;
          /* LED1 灯状态改变 */
          LED1 = !LED1;
          /* LED2 灯的状态改变 */
          LED2 = !LED2;
        }
}
```

5.4 贯穿项目实现

传感信息采集完成之后,需要将数据传输至 PC 或其他上位机,数据的传输可以通过多种方式进行,比如串口、USB 和以太网等。本书中采用串口将采集信息传输至 PC。

4.8 节讲解了传感信息的采集,本节的主要任务是将传感信息采集的数据通过串口传输至 PC。

首先需要对串口进行初始化。初始化的过程如下:

- 选择晶振为外部 32MHz 晶体振荡器;
- 设置 P0 口优先作为串口;
- 采用串口 USART 方式传输;
- 设置波特率为 57 600bps。

具体实现如实例 5-6 所示。

【实例 5-6】 initUARTtest()

```
void initUARTtest(void)
{
    /* 选择晶振为 32MHz */
    CLKCONCMD &= ~0x40;
    /* 等待晶振稳定 */
    while(!(SLEEPSTA & 0x40));
    /* TICHSPD128 分频,CLKSPD 不分频 */
    CLKCONCMD &= ~0x47;
    /关闭不用的 RC 振荡器 */
```

```
    SLEEPCMD |= 0x04;
    /* 位置 1 P0 口 */
    PERCFG = 0x00;
    /* P0 用作串口 */
    P0SEL = 0x3c;
    /* UART 方式 */
    U0CSR |= 0x80;
    /* baud_e */
    U0GCR |= 10;
    /* 波特率设为 57600 */
    U0BAUD |= 216;
    UTX0IF = 1;
    /* 允许接收 */
    U0CSR |= 0X40;
    /* 开总中断,接收中断 */
    IEN0 |= 0x84;
}
```

CC2530 串口向 PC 传输数据通过 U0DBUF 寄存器,获得数据的长度后,将数据写入 U0DBUF 中,具体实现如实例 5-7 所示。

【实例 5-7】 UartTX_Send_String()

```
void UartTX_Send_String(unsigned char *Data,int len)
{
  int j;
  for(j=0;j<len;j++)
  {
    U0DBUF = *Data++;
    while(UTX0IF == 0);
    UTX0IF = 0;
  }
}
```

在主函数中调用了温度转换函数,并且获取温度信息和光照信息,并通过串口传输至 PC。具体实现如实例 5-8 所示。

【实例 5-8】 main()

```
void main()
{
  unsigned char i;
  unsigned char* temp;
  unsigned char* guangM;
  initUARTtest();
  while(1)
  {
    /* 开始转换 */
    DS18B20_SendConvert();
    /* 延时 1S */
    for(i=20; i>0; i--)
```

```
        delay_nμs(50000);

        /* 获取传感信息 */
        temp = DS18B20_GetTem();
        send_buf[0] = temp[0];
        send_buf[1] = temp[1];
        guangM = getGuangM();
        send_buf[2] = guangM[0];
        send_buf[3] = guangM[1];
        /* 串口输出采集的温度 */
        UartTX_Send_String(send_buf,4);
        asm("NOP");
      }
    }
```

温度信息的采集直接调用温度获取函数,在本任务中光照信息的获取需要返回值,具体实现如实例 5-9 所示。

【实例 5-9】 getGuangM()

```
unsigned char* getGuangM(void)
{
  P0DIR &= 0xfd;
  ADCIF = 0;
  /* 清 EOC 标志 */
  ADCH &= 0X00;
  /* P0.7 做 ad 口 */
  APCFG |= 0X02;
  /* 单次转换,,对 P01 进行采样 */
  ADCCON3 = 0x71;
  /* 等待转化是否完成 */
  while(!(ADCCON1&0x80));
  /* 送 A/D 转换的高位 */
  Guang[0] =  ADCH;
  /* 送数据的第 6 个字节 A/D 转换的低位 */
  Guang[1] =  ADCL;
  return (Guang);
}
```

本任务头文件如实例 5-10 所示。

【实例 5-10】 头文件

```
#include "ioCC2530.h"
#include "DS18B20.h"
unsigned char Guang[2];
unsigned char send_buf[4];
unsigned char i;
unsigned char* getGuangM(void);
void initUARTtest(void);
void UartTX_Send_String(unsigned char *Data,int len);
```

本章总结

小结

- CC2530有两个串行通信接口：USART0和USART1。
- CC2530的两个串口由两种串口模式：UART和SPI模式。
- 串口有5个寄存器，分别是串口控制和状态寄存器UxCSR、串口UART控制寄存器UxUCR、串口接收/传送数据缓存寄存器UxDBUF寄存器、串口波特率控制寄存器UxBAUD寄存器和串口通用控制UxGCR寄存器。
- DMA控制器含有若干可编程的DMA通道，用来实现存储器之间的数据传送。
- DMA控制器控制整个XDATA存储映射空间的数据传送。由于大多数SFR寄存器映射到DMA存储器空间，DMA通道的操作能够减轻CPU的负担。
- DMA控制器还可以保持CPU在低功耗模式下与外设单元之间传送数据，不需要唤醒，降低整个系统的功耗。
- CC2530有5个定时器、一个16位定时器（定时器1和定时器2）、一个8位定时器（定时器3和定时器4）、2个用于休眠的定时器（定时器2和MAC定时器）。
- 定时器1支持定时和计数功能，有捕获、输出比较和PWM的功能。
- 定时器3和定时器4每个定时器有两个独立的比较通道。

Q&A

问题：DMA通道1～4配置地址高字节寄存器DMAxCFGH和低字节寄存器DMAxCFGL（x取值为1、2、3、4）之间的关系。

回答：寄存器DMA1CFGH和寄存器DMA1CFGL给出DMA通道1配置数据结构的开始地址，其后跟着DMA通道2～4的配置数据结构。DMA通道1～4的DMA配置数据将存储器连续的区域内，以DMA1CFGH和DMA1CFGL所保存的地址开始，包含32个字节。

章节练习

习题

1. 以下寄存器属于“串口0控制和状态寄存器”的是_____。

   A. U0CSR　　B. U0UCR　　C. U0BUF　　D. U0GCR

2. 下列对于定时器1说法中，错误的是_____。

   A. 5个独立的捕获、比较通道

   B. 只有上升沿具有输入捕获

   C. 可被1、8、32或128整除的时钟分频器

   D. 具有DMA触发功能

3. CC2530有两个串行通信接口：_____和_____。

4. 定时器1支持定时和计数功能，有_____、_____和_____功能。

5. 简述异步串口UART模式操作的特点。

6. 简述DMA控制的主要功能。

7. 参照表5-28配置串口0为SPI的从模式。

表5-28 U0CSR寄存器

位	名称	复位	R/W	描 述
7	MODE	0	R/W	USART模式选择 0：SPI模式 1：UART模式
6	RE	0	R/W	UART接收器使能，但是在UART完全配置之前不能接收 0：禁止接收器 1：使能接收器
5	SLAVE	0	R/W	SPI主或者从模式选择 0：SPI主模式 1：SPI从模式
4	FE	0	R/W0	UART帧错误状态 0：无帧错误检测 1：字节收到不正确停止位级别
3	FRR	0	R/W0	UART奇偶校验错误状态 0：无奇偶校验检测 1：字节收到奇偶错误
2	RX_BYTE	0	R/W0	接收字节状态，UART模式和SPI模式。当读U0DBUF该位自动清零，通过写0清除它，这样有效丢弃U0BUF中的数据 0：没有收到字节 1：接收字节就绪
1	TX_BYTE	0	R/W0	传送字节状态，UART和SPI从模式 0：字节没有传送 1：写到数据缓存寄存器的最后字节已经传送
0	ACTIVE	0	R	USART传送/接收主动状态 0：USART空闲 1：USART在传送或者接收模式忙碌

8. 根据表5-29判断DMA通道1是否发生中断。

表5-29 DMAIRQ寄存器

位	名称	复位	R/W	描 述
7～5	--	000	R/W0	保留
4	DMAIF4	0	R/W0	DMA通道4中断标志 0：DMA通道传送标志 1：DMA通道传送完成/中断未决

续表

位	名称	复位	R/W	描　　述
3	DMAIF3	0	R/W0	DMA 通道 3 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决
2	DAMIF2	0	R/W0	DMA 通道 2 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决
1	DMAIF1	0	R/W0	DMA 通道 1 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决
0	DMAIF0	0	R/W0	DMA 通道 0 中断标志 0：DMA 通道传送标志 1：DMA 通道传送完成/中断未决