# 第 2-18 讲：直接存储器访问（DMA）

## 1. 学习目的

1. 掌握 STC8A8K64D4 系列单片机 DMA 的特点以及编程方法。
2. 学习 STC8A8K64D4 系列单片机串口和 ADC 使用 DMA 的编程方法。

## 2. DMA 简介

DMA 全称是 Direct Memory Access，即直接存储器访问。DMA 的作用是从一个地址空间在无需 CPU 干预的情况下将数据“搬运”到另一个地址空间，由此实现外设和存储器之间或者存储器和存储器之间的高速数据传输。

对比一下使用 DMA 和不使用 DMA 时串口接收并将数据存储到数组的操作，便于我们理解 DMA 的功能。

- 普通操作：串口接收到数据后，产生中断，中断服务函数中，软件从串口硬件接收 FIFO 中读出数据并将数据保存到内存。这个过程中，CPU 参与了读取数据、保存数据到数组，串口中断需要保护现场和恢复现场等。
- DMA 操作：初始化时配置好串口 DMA 的源地址、目的地址和接收数据长度并启动 DMA 传输，之后，串口接收到数据后会自动存入目的地址，此过程无需 CPU 干预，也没有中断保护现场和恢复现场的过程。

由此可见，DMA 的优点是速度快和无需 CPU 干预，使用 DMA 可以有效降低 CPU 的负载。

1. STC8A8K64D4 支持的 CPU 操作

STC8A8K64D4 并不是所有的外设都支持 DMA 的，如 I2C 总线就是不支持 DMA 的，STC8A8K64D4 支持的 DMA 操作如下：

- M2M_DMA： XRAM 存储器到 XRAM 存储器的数据读写
- ADC_DMA：自动扫描使能的 ADC 通道并将转换的 ADC 数据自动存储到 XRAM 中
- SPI_DMA：自动将 XRAM 中的数据和 SPI 外设之间进行数据交换
- UR1T_DMA：自动将 XRAM 中的数据通过串口 1 发送出去
- UR1R_DMA：自动将串口 1 接收到的数据存储到 XRAM 中
- UR2T_DMA：自动将 XRAM 中的数据通过串口 2 发送出去
- UR2R_DMA：自动将串口 2 接收到的数据存储到 XRAM 中
- UR3T_DMA：自动将 XRAM 中的数据通过串口 3 发送出去
- UR3R_DMA：自动将串口 3 接收到的数据存储到 XRAM 中
- UR4T_DMA：自动将 XRAM 中的数据通过串口 4 发送出去
- UR4R_DMA：自动将串口 4 接收到的数据存储到 XRAM 中
- LCM_DMA： 自动将 XRAM 中的数据和 LCM 设备之间进行数据交换

2. STC8A8K64D4 的 DMA 数据传输长度

STC8A8K64D4 一次 DMA 数据传输的最大数据量为 256 字节。

3. STC8A8K64D4 的 DMA 中断优先级

1) 每种 DMA 对 XRAM 的读写操作都可设置 4 级访问优先级，硬件自动进行 XRAM 总线的访问仲裁，不会影响 CPU 的 XRAM 的访问。

2) 相同优先级下，不同 DMA 对 XRAM 的访问顺序如下：SPI_DMA，UR1R_DMA，UR1T_DMA，UR2R_DMA，UR2T_DMA，UR3R_DMA，UR3T_DMA， UR4R_DMA，UR4T_DMA， LCM_DMA， M2M_DMA， ADC_DMA。

# 3. 软件设计

## 3.1. 串口 DMA 收发数据实验

✧ 注：本节的实验是在“实验 2-6-1：串口 1 数据收发实验”的基础上修改，本节对应的实验源码是：“实验 2-18-1：串口 1 使用 DMA 收发数据”。

✧ 关于串口相关的内容，读者可以参阅《第 2-6 讲：串口通信》，这里，我们关注的是串口使用 DMA 通信。

### 3.1.1. 实验内容

串口 1 使用 DMA 接收电脑端串口调试助手发过来的数据，并将接收到的数据原样返回，数据传输长度为 20 个字节。

✧ 本实验旨在让读者理解串口 DMA 的使用，没有考虑串口接收不定长数据（这个不属于串口 DMA 自身的功能），在下一个实验里面会处理串口不定长数据的接收。

### 3.1.2. 代码编写

1. 新建一个名称为“uart_dma.c”的文件及其头文件“uart_dma.h”并保存到工程的“Source”文件夹，并将“uart_dma.c”加入到 Keil 工程中的“SOURCE”组。

2. 引用头文件

因为在 uart_dma.c”文件中使用了“uart_dma.c”文件中的函数，所以需要引用下面的头文件“uart_dma.h”。

**代码清单：**引用头文件

```
1.  //引用 uart_dma 的头文件
2.  #include   " uart_dma.h"
```

3. 串口 DMA 初始化

使用串口 DMA 之前，需要对 DMA 进行相应的配置，包含中断配置、串口发送 DMA 和接收 DMA 的配置。初始化时需要注意以下几点：

1) 串口 DMA 发送时 DMA 的源地址以及串口 DMA 接收时 DMA 的目的地址必须位于 XRAM，即我们定义的用于 DMA 发送或接收的数组必须用关键字“xdata”修饰。

2) 串口 DMA 发送和接收的实际数据长度（字节数）等于传输总字节寄存器设置的值加 1，如串口发送总字节寄存器设置的值是“1”，那么，实际发送传输长度是 2，也就是发送两个字节后才会置位发送中断请求标志位。

3) 串口接收具有随机性，因此，初始化函数中应启动 DMA 接收，即初始化完成后即具备数据接收功能。至于发送，则不需要在初始化函数中启动，在有数据需要发送的时候启动即可。

串口 DMA 初始化配置代码清单如下。

**代码清单：**定义串口发送和接收数组

```
1.  u8 xdata uart_rx[UART_BUF_LEN]; //定义串口接收缓存数组，位于 XRAM 区域
2.  u8 xdata uart_tx[UART_BUF_LEN]; //定义串口接收缓存数据，位于 XRAM 区域
```

**代码清单：**串口 DMA 初始化

```
1.  /*******************************************************************************
2.  功能描述：串口 1 DMA 配置。因为接收具有不确定性，因此，串口接收初始化函数中启动了 DMA 接收.
3.            ：串口发送无需预先启动，在发送数据的时候启动即可。
4.  参     数：无
5.  返 回 值：无
6.  *******************************************************************************/
7.  void uart1_dma_config(void)
8.  {
9.      P_SW2 = 0x80;
10.     DMA_UR1T_CFG = 0x80;          //开启 UART1 DMA 发送中断
11.     DMA_UR1T_STA = 0x00;          //UART1 DMA 发送状态寄存器清零
12.     DMA_UR1T_AMT = DMA_AMT_LEN;   //设置传输总字节数：  n+1
13.     DMA_UR1T_TXAH = (u8)((u16)&uart_tx >> 8);//设置 UART1 DMA 发送源地址
14.     DMA_UR1T_TXAL = (u8)((u16)&uart_tx);
15.     DMA_UR1R_CFG = 0x80;          //开启 UART1 DMA 接收中断
16.     DMA_UR1R_STA = 0x00;          //UART1 DMA 接收状态寄存器清零
17.     DMA_UR1R_AMT = DMA_AMT_LEN;   //设置传输总字节数：  n+1
18.     DMA_UR1R_RXAH = (u8)((u16)&uart_rx >> 8);//设置 UART1 DMA 接收目的地址
19.     DMA_UR1R_RXAL = (u8)((u16)&uart_rx);
20.     DMA_UR1R_CR = 0xA1;           //启动 UART1 DMA 接收，清除 FIFO
21. }
```

4. 串口 DMA 中断

串口 DMA 中断服务函数中，使用变量“uart1_tx_complete_flag”标记串口 DMA 发送完成，变量“uart1_rx_complete_flag”标记串口 DMA 接收完成。

当串口 DMA 发送完指定长度的数据后，会触发 DMA 发送完成中断，当串口 DMA 接收到指定长度的数据（本例中配置的是 20 个字节），会触发 DMA 接收完成中断。中断服务函数中读取串口 DMA 状态寄存器，以此判断中断类型，然后根据中断类型置位“uart1_tx_complete_flag”或“uart1_rx_complete_flag”，应用程序即可通过这两个标志判断串口 DMA 发送是否完成和串门 DMA 是否接收到数据，代码清单如下。

**代码清单：**串口 DMA 中断服务函数

```
1.  /*******************************************************************************
```

```
2.  功能描述：DMA 中断服务程序。因为 DMA 中断号大于 31，所以这里借用了 13 号中断
3.  参    数：无
4.  返 回 值：无
5.  ********************************************************************************/
6.  void UART1_DMA_Interrupt(void) interrupt 13
7.  {
8.      if (DMA_UR1T_STA & 0x01) //发送完成
9.      {
10.         DMA_UR1T_STA &= ~0x01;
11.         uart1_tx_complete_flag = 1;  //DMA 发送完成标志置位
12.     }
13.     if (DMA_UR1T_STA & 0x04) //数据覆盖
14.     {
15.         DMA_UR1T_STA &= ~0x04;
16.     }
17.     if (DMA_UR1R_STA & 0x01) //接收完成
18.     {
19.         DMA_UR1R_STA &= ~0x01;
20.         uart1_rx_complete_flag = 1;  //DMA 接收完成标志置位
21.     }
22.     if (DMA_UR1R_STA & 0x02) //数据丢弃
23.     {
24.         DMA_UR1R_STA &= ~0x02;
25.     }
26. }
```

5. 串口 DMA 发送数据

串口 DMA 发送数据时，设置好发送的数据长度、串口 DMA 发送源地址，之后启动 DMA 发送即可。串口 DMA 发送数据需要注意以下几点：

1) DMA 发送源地址必须位于 XRAM；

2) 串口 DMA 一次传输数据的长度范围为 1~256 个字节，串口 DMA 实际发送的数据长度是发送长度配置寄存器设置的值加 1，如“DMA_UR1T_AMT”设置的值为 0，实际串口 1 的 DMA 发送的数据长度是 1 个字节。

串口 DMA 发送的代码清单如下：

**代码清单：**串口 DMA 发送数据

```
1.  /********************************************************************************
2.  功能描述：串口 1DMA 发送指定长度的数据
3.  参    数：p_buf[in]：发送的数据
4.          ：length：发送的数据长度（字节数量），范围：1~256
5.  返 回 值：无
6.  ********************************************************************************/
7.  u8 uart1_dma_send(u8 *p_buf,u16 length)
8.  {
```

```
    if((length == 0) || (length > 255))//检查发送的数据长度是否合法
    {
        return ERR_SEND_LENGTH;
    }
    DMA_UR1T_AMT = (u8)(length-1);  //设置 UART1 传输总字节数：n+1
    DMA_UR1T_TXA = (u16)p_buf;      //设置 UART1 DMA 发送源地址
    DMA_UR1T_CR = 0xC0;             //启动 UART1_DMA 发送

    return SUCCESS;
}
```

6. 串口 DMA 接收数据

主循环中查询到变量“uart1_rx_complete_flag”置位，表示串口已经接收到 20 个字节的数据，并且这 20 个字节已经存入到我们定义在 XRAM 区域的串口 DMA 接收缓存数组“uart_rx”。这时，我们可以将数据拷贝到串口 DMA 发送数组“uart_tx”，并通过串口 DMA 将数据原样发回。

**代码清单**：串口 DMA 接收数据

```
/*******************************************************************************
功能描述：查询串口 DMA 接收标志“uart1_rx_complete_flag”，如果置位，读出接收的数据并原样返回
参    数：无
返 回 值：无
*******************************************************************************/
void uart1_dma_handle(void)
{
    if(uart1_rx_complete_flag == 1)//接收标志置位
    {
        uart1_rx_complete_flag = 0;  //接收标志清零
        memcpy(uart_tx,uart_rx,DMA_AMT_LEN+1);  //将接收的数据拷贝到串口 DMA 发送源地址
        DMA_UR1R_AMT = DMA_AMT_LEN;  //设置传输总字节数：  n+1
        DMA_UR1R_RXAH = (u8)((u16)&uart_rx >> 8);//设置 UART1 DMA 接收目的地址
        DMA_UR1R_RXAL = (u8)((u16)&uart_rx);
        DMA_UR1R_CR = 0xA1;          //开启 UART1 DMA 接收功能，开始 UART1_DMA 自动接收，清除 FIFO
        uart1_dma_send(uart_tx,DMA_AMT_LEN+1);  //将接收的数据通过串口发送出去
    }
}
```

7. 主函数

主函数中，初始化用到的引脚、串口 1 及其 DMA，之后在主循环中调用“uart1_dma_handle()”函数查询串口 1 接收 DMA 标志是否置位，如置位，则读出数据，并调用串口 1 的 DMA 发送函数“uart1_dma_send”将读取的数据发送，由此完成串口接收数据的回环。

**代码清单**：主函数

```
/*******************************************************************************
```

```
2.  功能描述：主函数
3.  参    数：无
4.  返 回 值：int 类型
5.  ******************************************************************************/
6.  int main(void)
7.  {
8.      P2M1 &= 0x3F;   P2M0 &= 0x3F;     //设置 P2.6、P2.7 为准双向口
9.      P7M1 &= 0xF9;   P7M0 &= 0xF9;     //设置 P7.1、P7.2 为准双向口
10.
11.     P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
12.     P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）
13.
14.     uart1_dma_config();           //串口 DMA 初始化
15.     uart1_init();                 //串口 1 初始化
16.     EA = 1;                       //使能总中断
17.
18.     while(1)
19.     {
20.         uart1_dma_handle();//将串口接收到的数据（长度 20 个字节）原样返回
21.     }
22. }
```

### 3.1.3. 硬件连接

本实验使用的是USB转串口，按照下图所示短接J5的跳线帽，将串口1的P3.0和P3.1连接到USB转串口电路。


图 1：硬件连接

### 3.1.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-18-1：串口 1 使用 DMA 收发数据”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（**这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题**）。
2) 双击“…\uart1_dma\project”目录下的工程文件“uart1_dma.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“uart1_dma.hex”位于工程的

“…\uart1_dma\Project\Object”目录下。

4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。

5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps，之后在发送框输入 20 个字节的数据，点击发送按钮发送数据。

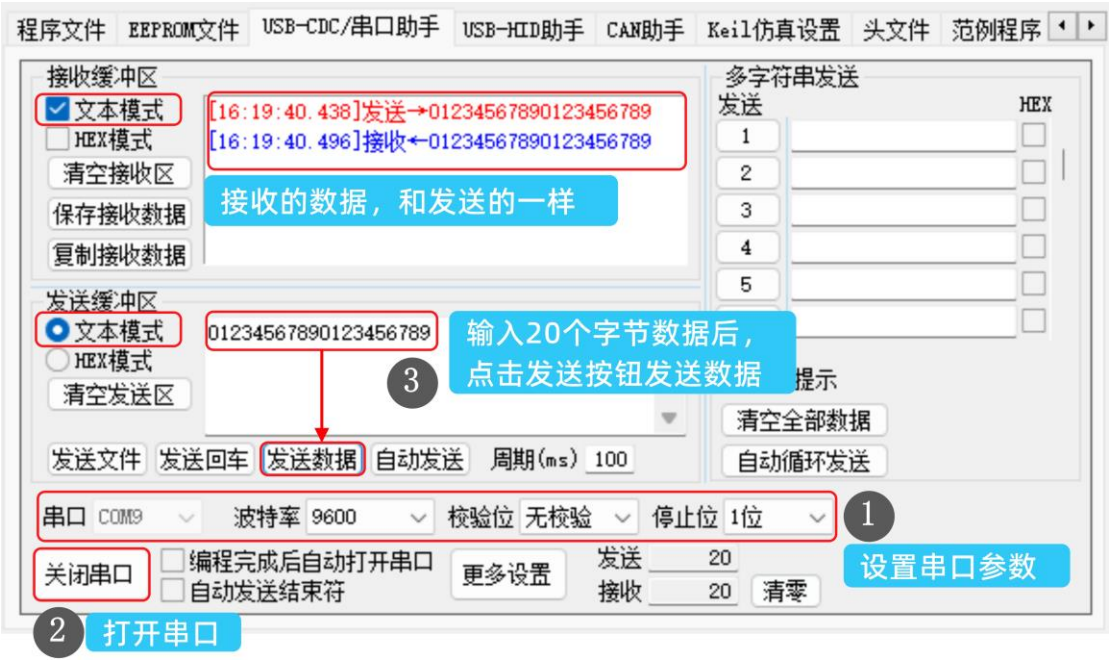


图 2：串口调试助手收发数据

6) 观察串口接收的数据，应和发送的数据一样。

## 3.2. 串口 DMA 接收不定长数据实验

✧ 注：本节的实验是在“实验 2-18-1：串口 1 使用 DMA 收发数据”的基础上修改，本节对应的实验源码是：“实验 2-18-2：串口 1 使用 DMA 接收不定长数据”。

■ 关于串口接收不定长数据：

本例中通过定时器配合串口 DMA 实现串口接收不定长数据，DMA 用于将串口接收的数据存放到指定的 XRAM 区域（我们定义在 XRAM 的缓存数组），定时器用于软件实现串口接收超时，实现的流程如下图所示。

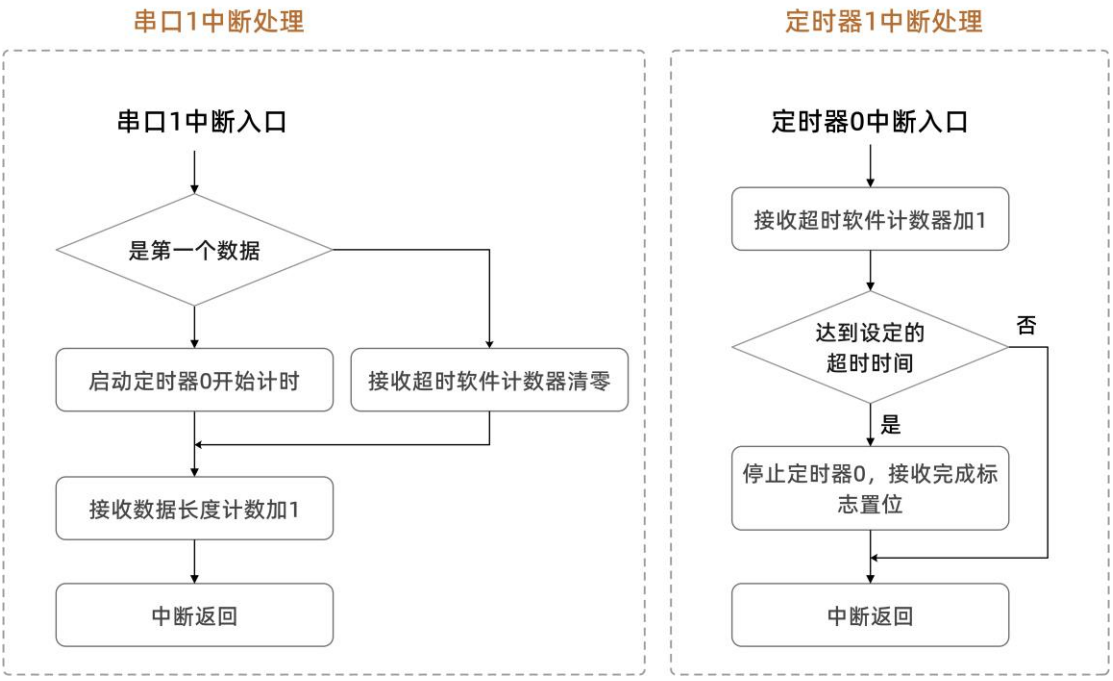


图 3：串口接收不定长数据的处理

从上面的流程可以看到，串口不定长接收需要使用串口 1 的中断和 Timer0 中断，而串口 DMA 的中断并不是必须使用的。这里，我们使用串口 DMA 的主要目的是利用 DMA “搬运”数据的便利性，即串口接收的数据会自动存储到我们定义在 XRAM 区域的数组，而不必在串口中断中逐个读取。如果我们不使用串口 DMA，不定长数据的流程和上图基本一样，区别是在串口中断中需要执行接收数据读取操作。

### 3.2.1. 实验内容

串口 1 使用 DMA 接收电脑端串口调试助手发过来的不定长的数据，并将接收到的数据原样返回，本例的功能需求如下：

1) 串口 1 的 DMA 发送/接收长度设置为最大值 256 个字节。
2) Timer0 的超时时间设置为 1ms，即 1ms 产生一次中断。
3) 串口 1 接收超时时间：5ms，即 5ms 没有接收到新数据，则认为数据接收完成，置位数据接收完成标志。主循环中查询到数据接收完成标志置位，通过串门发送 DMA 将接收的数据原样发回。

### 3.2.2. 代码编写

1. 初始化串口 1

串口 1 初始化函数代码清单如下，该函数中设置了串口 1 使用的引脚：RxD 为 P3.0，TxD 为 P3.1。串口 1 配置为 8 位数据位、波特率 9600bps，开启串口 1 的中断。

**代码清单**：串口 1 初始化

```
1.  /*********************************************************************************
2.  功能描述：初始化串口 1，设置为 8 位数据位、波特率 9600bps （系统时钟使用 24MHz）
3.  参    数：无
```

```
4.  返 回 值：无
5.  ******************************************************************************/
6.  void uart1_init(void)
7.  {
8.      ES = 0;          //初始化前关闭 UART1 中断
9.      P_SW1 &= 0x3F;   //设置串口 1 使用的引脚为：RxD--P3.0;TxD--P3.1
10.
11.     PCON &= 0x3f;    //波特率不倍速，串行口工作方式由 SM0、SM1 决定
12.     SCON = 0x50;     //8 位数据,可变波特率（SM0=0,SM1=1）
13.     AUXR |= 0x40;    //定时器时钟 1T 模式
14.     AUXR &= 0xFE;    //串口 1 选择定时器 1 为波特率发生器
15.
16.     TMOD &= 0x0F;    //设置定时器 1 模式：16 位自动重装方式
17.     TL1 = 0x8F;      //设置定时初始值
18.     TH1 = 0xFD;      //设置定时初始值
19.     ET1 = 0;         //禁止定时器 1 中断
20.     TR1 = 1;         //启动定时器 1
21.     ES = 1;          //开启串口 1 中断
22. }
```

2. 初始化 Timer0

配置 Timer0 为定时器、超时时间 1ms，代码清单如下。注意，这里只初始化 Timer0，但不启动 Timer0。

**代码清单：**串口 1 初始化

```
1.  /******************************************************************************
2.  功能描述：初始化 Timer0（系统时钟使用 24MHz），定时时间 1ms,1T
3.  参    数：无
4.  返 回 值：无
5.  ******************************************************************************/
6.  static void timer0_init(void)
7.  {
8.      AUXR |= 0x80;        //定时器时钟 1T 模式
9.      TMOD &= 0xF0;        //配置 Timer0 为定时器
10.     TL0 = 0x40;          //设置定时初始值
11.     TH0 = 0xA2;          //设置定时初始值
12.     TF0 = 0;             //清除 TF0 标志
13.     IP |= 0x02;          //中断优先级配置为最高优先级
14.     IPH |= 0x02;
15.     ET0 = 1;             //使能定时器 0 中断
16. }
```

3. 定义串口 DMA 接收缓存及相关的标志

为了方便处理串口 1 不定长接收，我们声明一个名称为“uart_comm_info_t”的结构体，其成员包括串口 DMA 接收缓存及相关的标志。

**代码清单**：串口 1 不定长接收相关软件标志定义

```
typedef struct UartComm_Info
{
    u8   started;               //串口接收第一个字节数据的时候置位
    u8   complete;              //数据接收完成标志
    u8   delay_count;           //串口接收超时软件计数器
    u16  byte_count;            //串口接收字节软件计数
    u8   uart_tx[UART_BUF_LEN]; //串口 DMA 发送缓存
    u8   uart_rx[UART_BUF_LEN]; //串口 DMA 接收缓存
}uart_comm_info_t;
```

4. 串口 1 中断服务函数

串口 1 接收到数据进入中断服务函数后，通过“uart_comm_info.started”是否置位判断该数据是不是第一个数据，如果是第一个数据，启动 Timer0 开始计时并置位“uart_comm_info.started”，否则，清零串口接收超时软件计数器，对串口接收超时重新计时。

**代码清单**：串口 1 中断服务函数

```
/*******************************************************************************
 * 描  述 : 串口 1 中断服务函数
 * 入  参 : 无
 * 返回值 : 无
 *******************************************************************************/
void uart1_isr() interrupt 4 using 1
{
    if (RI)                             //是接收中断（接收中断请求标志位为 1）
    {
        RI = 0;                         //清零 RI 位（该位必须软件清零）
        if(uart_comm_info.started == 0)   //是第一个数据
        {
            TR0 = 1;                    //定时器 0 开始计时
            uart_comm_info.started = 1;
        }
        else
        {
            uart_comm_info.delay_count = 0; //清零串口接收超时软件计数器
        }
        uart_comm_info.byte_count++;        //接收数据长度加 1
    }
    if(TI)
    {
```

```
24.       TI = 0;
25.    }
26. }
```

5. Timer0 中断服务函数

Timer0 每 1ms 产生一次中断，中断服务函数中将串口接收超时软件计数器“uart_comm_info.delay_count”加 1，当串口接收超时软件计数器的值为 5，即超时时间达到 5ms 时，认为一包数据接收完成，置位数据接收完成标志“uart_comm_info.complete”并停止 Timer，代码清单如下。

**代码清单**：Timer0 中断服务函数

```
1.  /*************************************************************
2.   * 描  述 : 定时器 0 中断服务函数
3.   * 入  参 : 无
4.   * 返回值 : 无
5.  *************************************************************/
6.  void timer0_isr() interrupt 1
7.  {
8.     uart_comm_info.delay_count++;                          //串口接收超时软件计数器加 1
9.     if(uart_comm_info.delay_count >= UART_DELAY_TIME_CNT)  //达到超时时间，认为一包数据接收完成
10.    {
11.       TR0 = 0;                      //停止定时器
12.       uart_comm_info.complete = 1; //数据接收完成标志置位
13.    }
14. }
```

✧ 说明：本例中，串口 DMA 接收的数据原样返回的处理和“实验 2-18-1：串口 1 使用 DMA 收发数据”的流程一样，这里不再赘述，读者可以参阅“实验 2-18-1”代码编写部分的文档和本例的实验源码。

### 3.2.3. 硬件连接

本实验使用的是 USB 转串口，按照下图所示短接 J5 的跳线帽，将串口 1 的 P3.0 和 P3.1 连接到 USB 转串口电路。


图 4：硬件连接

### 3.2.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-18-2：串口 1 使用 DMA 接收不定长数据”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（**这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题**）。
2) 双击“…\uart1_dma_random_length \project”目录下的工程文件“uart1_dma_random_length.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“uart1_dma_random_length.hex”位于工程的“…\uart1_dma_random_length\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps，之后在发送框输入（1~256）之间任意长度的数据，点击发送按钮发送数据。


图 5：串口调试助手收发数据

6) 观察串口接收的数据，应和发送的数据一样。

## 3.3. ADC 使用 DMA 实验

✧ 注：本节的实验是在“实验 2-11-1：ADC 采样电位器电压（查询方式）”的基础上修改，本节对应的实验源码是：“实验 2-18-3：ADC 使用 DMA（1 个 ADC 通道）”。

✧ 关于 ADC 应用相关的内容，读者可以参阅《第 2-11 讲：模数转换 ADC》，本节我们关注的是 ADC 如何使用 DMA。

关于 ADC DMA，我们需要关注两个方面，一是 ADC 使用 DMA 后增加了哪些功能，了解这些新增的功能有助于我们在实际开发时决定是否使用 DMA，二是 ADC 采样数据的存储格式，这关系到我们如何正确的读取 ADC 采样数据。

■ ADC 使用 DMA 后增加的功能：

1) 通过使用 DMA，ADC 采样结果由硬件自动储存到我们指定的位于 XRAM 区域的地址。

2) 可以进行多个 ADC 通道转换，通过 ADC_DMA 通道使能寄存器（DMA_ADC_CHSWx）使能通道，使能的通道由硬件自动完成轮询采样，轮询采样的顺序按照已使能通道的编号从小到大执行。

3) 通过 ADC_DMA 配置寄存器 2（DMA_ADC_CFG2）可以配置 ADC 转换的次数，配置后，由硬件自动完成设定次数的采样，并且，硬件会自动计算出 ADC 采样的平均值以及余数。

■ ADC DMA 采样数据的存储格式

ADC DMA 转换完成后，存储到 XRAM 的数据包含采样结果、ADC 通道编号、平均值和余数，这些数据在 XRAM 中是如何分布的？

ADC 转换结果右对齐时的存储格式如下表所示，注意表中的 ADC 通道顺序是已使能 ADC 通道编号从小到大排序的。如使能了两个 ADC 通道：通道 2 和通道 14，那么下表中“使能的第 1 通道”对应的是 ADC 通道 2，“使能的第 2 通道”对应的是 ADC 通道 14。

表 1：ADC DMA 采样数据的存储格式

| **ADC 通道** | **偏移地址** | **存储的数据** |
|---|---|---|
| 使能的第 1 通道 | 0 | 第 1 次 ADC 转换结果的高字节。 |
| | 1 | 第 1 次 ADC 转换结果的低字节。 |
| | …… | …… |
| | 2n-2 | 第 n 次 ADC 转换结果的高字节。 |
| | 2n-1 | 第 n 次 ADC 转换结果的低字节。 |
| | 2n | ADC 通道号。 |
| | 2n+1 | n 次 ADC 转换结果取完平均值之后的余数。 |
| | 2n+2 | n 次 ADC 转换结果平均值的高字节。 |
| | 2n+3 | n 次 ADC 转换结果平均值的低字节。 |
| 使能的第 2 通道 | (2n+4)+0 | 第 1 次 ADC 转换结果的高字节。 |
| | (2n+4)+1 | 第 1 次 ADC 转换结果的低字节。 |
| | …… | …… |
| | (2n+4)+2n-2 | 第 n 次 ADC 转换结果的高字节。 |
| | (2n+4)+ 2n-1 | 第 n 次 ADC 转换结果的低字节。 |
| | (2n+4)+ 2n | ADC 通道号。 |
| | (2n+4)+ 2n+1 | n 次 ADC 转换结果取完平均值之后的余数。 |
| | (2n+4)+ 2n+2 | n 次 ADC 转换结果平均值的高字节。 |
| | (2n+4)+ 2n+3 | n 次 ADC 转换结果平均值的低字节。 |
| …… | …… | …… |
| 使能的第 m 通道 | (m-1)(2n+4)+0 | 第 1 次 ADC 转换结果的高字节。 |
| | (m-1)(2n+4)+1 | 第 1 次 ADC 转换结果的低字节。 |
| | …… | …… |

| | | |
|---|---|---|
| | (m-1)(2n+4)+2n-2 | 第 n 次 ADC 转换结果的高字节。 |
| | (m-1)(2n+4)+ 2n-1 | 第 n 次 ADC 转换结果的低字节。 |
| | (m-1)(2n+4)+ 2n | ADC 通道号。 |
| | (m-1)(2n+4)+2n+1 | n 次 ADC 转换结果取完平均值之后的余数。 |
| | (m-1)(2n+4)+2n+2 | n 次 ADC 转换结果平均值的高字节。 |
| | (m-1)(2n+4)+2n+3 | n 次 ADC 转换结果平均值的低字节。 |

### 3.3.1. 实验内容

使用 ADC 模拟输入通道 14（即引脚 P0.6）采样电位器抽头电压，程序中每秒执行一次电压采样，采样结果计算为电压值后通过串口输出（包括 ADC 采样值、通道号、平均值和余数）。

本实验对于 ADC 配置部分仍然使用“实验 2-11-1：ADC 采样电位器电压（查询方式）”中的配置，对于 DMA 部分的配置如下：

1) ADC 通道：通道 14。
2) ADC 转换次数：8 次。
3) ADC DMA 中断：开启（因为ADC DMA 中断号大于31，所以这里借用了 13 号中断），用于监视 ADC DMA 转换是否完成。

### 3.3.2. 代码编写

1. 定义 ADC DMA 使用的通道数量、转换次数、转换完成标志和存储转换结果的数组，即 ADC DMA 的目的地址（必须位于 XRAM 区域）。

**代码清单**：定义 ADC DMA 相关标志和 ADC DMA 的目的地址

```
1.  //1~16，使用的 ADC 转换通道数量，必须和[ADC_DMA 通道使能寄存器(DMA_ADC_CHSWx)]中启用的 ADC 通道数量一致
2.  #define ADC_CH_NUM          1
3.  //ADC 转换次数，必须和[ADC_DMA 配置寄存器 2(DMA_ADC_CFG2)]设置的一致
4.  #define ADC_SAMPLES_NUM     8
5.  //每个通道 ADC 转换数据总字节数=2*转换次数+4
6.  #define ADC_DATA_SIZE       (ADC_SAMPLES_NUM*2 + 4)
7.
8.  //存储 ADC DMA 转换结果，即 ADC DMA 的目的地址
9.  u8 xdata adc_samples_buff[ADC_CH_NUM][ADC_DATA_SIZE];
10. //ADC DMA 转换完成标志
11. bit adc_dma_complete;
```

2. 初始化 ADC DMA。

使能 ADC 通道 14，ADC 转换次数设置为 8 次，开启 ADC DMA 中断，代码清单如下。

**代码清单**：初始化 ADC DMA

```
1.  /*********************************************************************************
2.  功能描述：ADC DMA 配置
3.  参    数：无
```

```
4.  返 回 值：无
5.  ******************************************************************************/
6.  void adc_dma_config(void)
7.  {
8.      P_SW2 = 0x80;
9.      BMM_ADC_STA = 0x00;          //清零 ADC DMA 状态寄存器
10.     BMM_ADC_CFG = 0x80;          //开启 ADC DMA 中断
11.     BMM_ADC_RXAH = (u8)((u16)(&adc_samples_buff) >> 8); //ADC 转换数据存储地址，即 ADC DMA 的目的地址
12.     BMM_ADC_RXAL = (u8)((u16)(&adc_samples_buff));
13.     BMM_ADC_CFG2 = 0x0A;     //每个通道 ADC 转换次数:8
14.     BMM_ADC_CHSW0 = 0x40;    //使能 ADC 通道 14
15.     BMM_ADC_CHSW1 = 0x00;    //ADC 通道 7~通道 0：关闭
16.     BMM_ADC_CR = 0xC0;       //启动 ADC DMA 转换
17. }
```

3. ADC DMA 中断服务函数。

ADC DMA 完成扫描所有使能的 ADC 通道后，触发 ADC_DMA 中断，ADC_DMA 中断请求标志位置位，进入中断服务函数。

中断服务函数中软件清零 ADC DMA 状态寄存器，之后将置位转换完成标志“adc_dma_complete”，应用程序通过查询“adc_dma_complete”标志即可知道 ADC DMA 转换是否完成。

**代码清单**：初始化 ADC DMA

```
1.  /*******************************************************************************
2.  功能描述：ADC DMA 中断服务程序。因为 DMA 中断号大于 31，所以这里借用了 13 号中断
3.  参     数：无
4.  返 回 值：无
5.  *******************************************************************************/
6.  void ADC_DMA_Interrupt(void) interrupt 13
7.  {
8.      BMM_ADC_STA = 0;        //清零 ADC DMA 状态寄存器
9.      adc_dma_complete = 1;  //转换完成标志置位
10.     led_toggle(LED_1);
11. }
```

4. ADC DMA 转换结果的处理

查询到 ADC DMA 转换完成标志“adc_dma_complete”置位后，即可读取 ADC DMA 转换数据，包含：

1) ADC 通道号；
2) 8 次采样的数据，读取后计算为实际的电压值；
3) 8 次采样的数据的平均值，读取后计算为实际的电压值；
4) 计算平均值后的余数。

读取的数据通过串口输出，以便我们在电脑上可以使用串口调试助手观察到这些数据。

最后，启动 ADC DMA 转换，开启新的一轮 ADC 转换。为了方便其他程序模块调用，我们将 ADC DMA 转换结果处理的相关代码封装到名称为“adc_dma_handle”的函数中，代码清单如下。

**代码清单**：ADC DMA 转换结果的处理

```
/*******************************************************************************
功能描述：查询 ADC DMA 转换完成标志“adc_dma_complete”，如果置位，读出 ADC 采样结果，
        ：并将其转换为实际电压值
参    数：无
返 回 值：无
*******************************************************************************/
void adc_dma_handle(void)
{
    u8    i,j;
    u16 adc_value;  //存放 ADC 采样值
    float voltage;  //存放 ADC 采样值计算后的电压值

    if(adc_dma_complete)
    {
        adc_dma_complete = 0;
        for(i=0; i<ADC_CH_NUM; i++)
        {
            printf("ADC Channel:  %bd\r\n",adc_samples_buff[i][ADC_SAMPLES_NUM*2]);//ADC 通道
            for(j=0; j<ADC_SAMPLES_NUM; j++)//8 次 ADC 采样对应的电压值
            {
                adc_value=(adc_samples_buff[i][j*2]<<8)+adc_samples_buff[i][j*2+1]; //读取 ADC 采样值
                voltage   = (2.5*adc_value)/4096;        //将 ADC 采样值转换为电压（单位 V）
                printf("voltage: %.2fV\r\n",voltage);    //串口打印 ADC 采样电压
            }
            //8 次 ADC 采样的平均值
            adc_value = (adc_samples_buff[i][ADC_SAMPLES_NUM*2+2]<<8)+adc_samples_buff[i][ADC_SAMPL
                        ES_NUM*2+3];
            voltage   = (2.5*adc_value)/4096;        //将 ADC 采样值转换为电压（单位 V）
            printf("average: %.2fV\r\n",voltage);    //串口打印 ADC 采样电压
            //串口打印计算平均值后的余数
            printf("remainder:%bd\r\n",adc_samples_buff[i][ADC_SAMPLES_NUM*2+1]);
        }
        BMM_ADC_CR = 0xc0;       //启动 ADC DMA 转换
    }
}
```

5. 主函数

主函数中分别调用 ADC 和 ADC DMA 初始化函数完成 ADC 和 DMA 的初始化，之后在主循环中调用 adc_dma_handle()函数处理 ADC DMA。注意，主循环中延时 1000ms 是为

了方便在我们串口调试助手中观察实验数据，实际使用 ADC DMA 时，无需增加长延时。

**代码清单：**主函数

```
/*************************************************************************************
功能描述：主函数
参    数：无
返 回 值：int 类型
*************************************************************************************/
int main(void)
{
    P2M1 &= 0xBF;   P2M0 &= 0xBF;     //设置 P2.6 为准双向口（LED1）
    P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口（串口 1 的 RxD）
    P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出（串口 1 的 TxD）

    uart1_init();                 //串口 1 初始化
    adc_config();                 //初始化 ADC
    adc_dma_config();             //初始化 ADC DMA
    EA = 1;                       //使能总中断

    while(1)
    {
        adc_dma_handle();  //ADC DMA 处理
        delay_ms(1000);    //延时 1000ms，方便在串口调试助手中观察实验数据
    }
}
```

### 3.3.3. 硬件连接

本实验需要使用 ADC 模拟通道输入 14（即引脚 P0.6）采样电位器抽头电压，因此需要将 P06 引脚和电位器电路通过跳线帽连接，如下图所示。

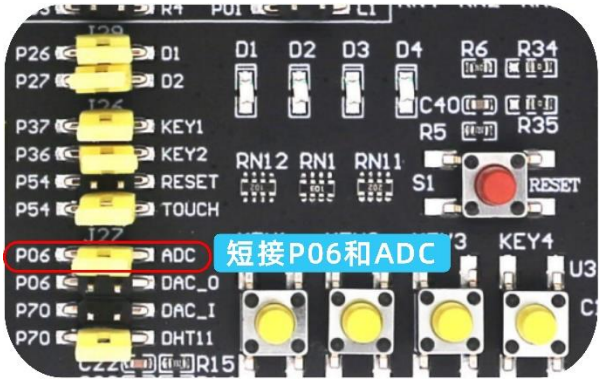


图 6：硬件连接

### 3.3.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-18-3：ADC 使用 DMA（1 个 ADC 通道）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（**这样**

**做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题**）。

2) 双击“…\adc_dma\project”目录下的工程文件“adc_dma.uvproj”。

3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“adc_dma.hex”位于工程的“…\adc_dma\Project\Object”目录下。

4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。

5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps，程序运行后，在串口接收窗口可以看到开发板上报的 ADC 采样的电压值，如下图所示。


图 7：串口调试助手中观察电压值

6) 旋转电位器改变电位器抽头电压，观察串口接收的数据，可以看到电压值的变化。

✧ **说明：**使用多个 ADC 通道和使用 1 个 ADC 通道类似，在软件上只需做少量的修改即可。如我们在使用 1 个 ADC 通道（通道 14）的例子上把他改为使用两个 ADC 通道（通道 14 和通道 2），只需修改下面两个关联的参数即可。

1) 修改使能的 ADC 通道：在 adc_dma_config()函数中修改 BMM_ADC_CHSW0 和 ADC_CH_NUM 寄存器。

```
BMM_ADC_CHSW0 = 0x40;    //使能 ADC 通道 14
BMM_ADC_CHSW1 = 0x00;    //关闭 ADC 通道 7~通道 0
```

改为：

```
BMM_ADC_CHSW0 = 0x40;    //使能 ADC 通道 14
BMM_ADC_CHSW1 = 0x04;    //使能 ADC 通道 2
```

2) 修改 ADC 通道数量的宏 ADC_CH_NUM，将其数值由 1 改为 2。

我们也编写好了 ADC DMA 使用 2 个通道的例子，例子源码在资料的“…\第 3 部分：配套例程源码目录下，名称为“实验 2-18-4：ADC 使用 DMA（2 个 ADC 通道）”，读者在编写的过程中可以参考。