第 13 章

移植到 ARM 微控制器上

在上一章中，我们介绍了如何将代码移植到 AVR 微控制器中，也许读者觉得这还不够过瘾。而现在的 ARM 微控制器还是挺流行的，已经发展到 Cortex - A8/A9 了。那么本章就将简单介绍如何将之前的代码移植到 ARM7 微控制器 AT91SAM7S64 芯片中。之所以选择这个平台，是因为 21IC 之前 DIY 过这个芯片的开发板，而且芯片内部资源很丰富。这个开发板的原理图可以在光盘中的 AT91SAM7 文件夹中找到。

13.1 AT91SAM7S64 芯片介绍

AT91SAM7S64 芯片是一款集成了 USB2.0 全速设备的 32 位高性能、低功耗 ARM7 微控制器。它使用了先进的精简指令集(RISC)架构，支持 ARM 和 THUMB 指令集，最大运行速度可达 55 MHz。内置 64 KB 的高速片内 FLASH，可支持一万次擦写。内置 16 KB 的 SRAM。在 USB 方面，具有如下特点：

- 可通过 USB 口进行在线烧录升级。
- 328 字节可编程的 FIFO。
- 端点 0(控制端点)的包长度为 8 字节。
- 3 个可编程的输入/输出端点，可支持批量/中断/等时传输，最大支持 64 字节，可选的单/双缓冲。

13.2 硬件抽象层的移植

这里，我们以上一章 AVR 的 USB 转串口代码为基础进行移植，因此在阅读本章前，请先阅读前一章的内容。

将原来的代码中，AT90USB.h 和 AT90USB.c 的文件改名为 AT91SAMxUSB.h 和 AT91SAMxUSB.c，分别实现 AT91SAM7 芯片的硬件抽象层。同样，AT91SAMxUSB.h 文件中主要是一些宏定义和函数声明，这里不再进行说明，请读者自行阅读光盘中的代码。

首先我们修改 USB 芯片初始化的函数 UsbChipInit，这个函数的实现代码如下：

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

```
函数功能:USB 芯片初始化。
入口参数:无。
返　　回:无。
备　　注:无。
*****************************************/
void UsbChipInit(void)
{
 * AT91C_UDP_TXVC = 0;                    //enable the transceiver
 * AT91C_UDP_IDR = 0xFFFFFFFF;            //disable all USB interrupts
 * AT91C_PIOA_PER = (1 << 16);             //Enable PA16
 * AT91C_PIOA_OER = (1 << 16);             //Output Enable PA16
 ConfigValue = 0;                        //配置值初始化为 0
 UsbDisconnect();                        //先断开 USB 连接
 UsbConnect();                           //将 USB 连接上
}
//////////////////////////End of function//////////////////////////////
```


同样,在该函数中,调用函数 UsbDisconnect,将设备从 USB 总线上断开 USB 的连接,然后调用 UsbConnect 函数,将 1.5 kΩ 上拉电阻接到总线上。由于这个芯片内部并没有集成上拉电阻,所以外部接了一个上拉电阻 R24(见图 13.2.1)。这个电阻通过一个三极管 Q2 连接到了 3.3 V 电源,Q2 的基极由主芯片的 PA16 口控制。当 PA16 输出高电平时,三极管截止,断开上拉电阻与 3.3 V 电源之间的连接;当 PA16 输出低电平时,三极管导通,接通上拉电阻与 3.3 V 电源之间的连接。UsbDisconnect 和 UsbConnect 函数的实现代码如下,至于具体的寄存器功能,请参看芯片的数据手册。在断开连接的函数中,加延时 1 s 的目的是为了确保让主机检测到设备已经断开连接了。

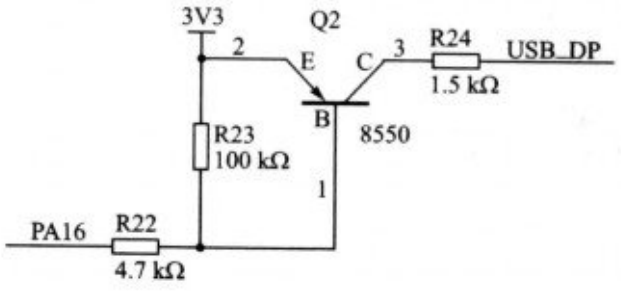

图 13.2.1　通过 GPIO 口控制 1.5 kΩ 上拉电阻的原理图

```
/*****************************************
函数功能:USB 断开连接函数。
入口参数:无。
```

```
返　　回:无。
备　　注:无。
*************************************/
void UsbDisconnect(void)
{
#ifdef DEBUG0
 Prints("断开 USB 连接。\r\n");
#endif
 * AT91C_PIOA_SODR = (1 << 16);        //Disconnect pull - up resistor
 DelayXms(1000);                       //延迟 1 s
}
//////////////////////End of function////////////////////////////

/*************************************
函数功能:USB 连接函数。
入口参数:无。
返　　回:无。
备　　注:无。
*************************************/
void UsbConnect(void)
{
#ifdef DEBUG0
 Prints("连接 USB。\r\n");
#endif
 * AT91C_PIOA_CODR = (1 << 16);        //Connect pull - up resistor
}
//////////////////////End of function////////////////////////////
```

当主机检测到设备连接到 USB 总线上后，主机就会复位 USB 总线，在这个事件中，我们需要对芯片的端点进行复位，并设置配置值为 0，对一些变量进行重新初始化等。下面是 USB 复位事件时的处理函数：

```
/*************************************
函数功能:总线复位中断处理函数。
入口参数:无。
返　　回:无。
备　　注:无。
*************************************/
void UsbBusReset(void)
{
#ifdef DEBUG0
 Prints("USB 总线复位。\r\n");
#endif
```

```
    UsbChipResetEndpoint();                    //复位端点
    ConfigValue = 0;                           //配置值初始化为 0
    UsbChipSetConfig(0);                       //设置芯片的配置值为 0
    Ep1InIsBusy = 0;                           //复位后端点 1 输入缓冲区空闲。
    Ep3InIsBusy = 0;                           //复位后端点 3 输入缓冲区空闲。
}
//////////////////////End of function////////////////////////////
```

在这个函数中，调用到了 UsbChipResetEndpoint 函数和 UsbChipSetConfig 这两个函数，它们的实现代码都在 AT91SAMxUSB.c 文件中，代码如下：

```
/*************************************
函数功能:USB 端点复位。
入口参数:无。
返    回:无。
备    注:无。
*************************************/
void UsbChipResetEndpoint(void)
{
 *AT91C_UDP_RSTEP = 0x0E;              //复位端点
 *AT91C_UDP_RSTEP = 0x00;              //复位端点完成
}
//////////////////////End of function////////////////////////////
/*************************************
函数功能:设置芯片配置状态。
入口参数:Value:配置值。
返    回:无。
备    注:无。
*************************************/
void UsbChipSetConfig(uint8 Value)
{
 if(Value == 0)
 {
  *AT91C_UDP_GLBSTATE& = ~(1 << 1);   //配置值为 0,清除配置
 }
 else
 {
  *AT91C_UDP_GLBSTATE| = (1 << 1);    //配置值非 0,设置配置
 }
}
//////////////////////End of function////////////////////////////
```

复位端点的函数中，对芯片的一些 USB 寄存器进行了操作，具体的含义请读者参看芯片的数据手册。另外还要修改使能非 0 端点的函数 UsbChipSetEndpointEnable，代码如下：

```
/*****************************************
函数功能：使能端点函数。
入口参数：Enable：是否使能。0 值为不使能，非 0 值为使能。
返　　回：无。
备　　注：无。
*****************************************/
void UsbChipSetEndpointEnable(uint8 Enable)
{
 if(Enable! = 0)
 {
  SetCsr(0,1 << 15); //使能端点 0
  SetCsr(1,(6 << 8)|(1 << 15)); //使能端点 1，并设置为批量 IN 端点
  SetCsr(2,(2 << 8)|(1 << 15)); //使能端点 2，并设置为批量 OUT 端点
  SetCsr(3,(7 << 8)|(1 << 15)); //使能端点 3，并设置为中断 IN 端点
 }
 else
 {
 }
}
//////////////////////End of function//////////////////////////////
```

这里用到了一个新的函数 SetCsr，因为该芯片对 CSR 寄存器的访问比较特殊，所以专门给它写了两个函数，其中一个就是 SetCsr，另一个就是 ClrCsr，分别用来设置和清除 CSR 寄存器中的某些位。关于 CSR 寄存器的功能，请参看芯片的数据手册。这两个函数实现的代码如下：

```
#define REG_NO_EFFECT_1_ALL 0x4F
/*****************************************
函数功能：设置 CSR 寄存器对应的位。
入口参数：endpoint：端点号；flags：要设置的位。
返　　回：无。
备　　注：无。
*****************************************/
void SetCsr(int endpoint, unsigned int flags)
{
 volatile unsigned int reg;
 reg = AT91C_UDP_CSR[endpoint];
```

```
    reg| = REG_NO_EFFECT_1_ALL;
    reg| = (flags);
    AT91C_UDP_CSR[endpoint] = reg;
    while((AT91C_UDP_CSR[endpoint]&(flags))! = (flags));
}
//////////////////////////End of function//////////////////////////////
/*****************************************
函数功能:清除 CSR 寄存器对应的位。
入口参数:endpoint:端点号;flags:要设置的位。
返　　回:无。
备　　注:无。
*****************************************/
void ClrCsr(int endpoint, unsigned int flags)
{
    volatile unsigned int reg;
    reg = AT91C_UDP_CSR[endpoint];
    reg| = REG_NO_EFFECT_1_ALL;
    reg&= ~(flags);
    AT91C_UDP_CSR[endpoint] = reg;
    while((AT91C_UDP_CSR[endpoint]&(flags)) == (flags));
}
//////////////////////////End of function//////////////////////////////
```

当总线复位完成后，主机就会通过端点 0 进行枚举了，这时使用的是控制传输。控制传输首先要发送 SETUP 包，因此我们要从 USB 模块那里知道，当前端点 0 中的数据是否为 SETUP 包，判断是否为 SETUP 包的函数代码如下：

```
/*************************************
函数功能:判断是否是 SETUP 包。
入口参数:无。
返　　回:0:不是 SETUP;非 0:是 SETUP。
备　　注:无。
*************************************/
int UsbChipIsSetup(uint8 Endp)
{
    if((AT91C_UDP_CSR[Endp])&(1 << 2))
    {
        return 0xFF;  //是 setup 包,返回非 0
    }
    else
    {
        return 0;
```

```
    }
}
//////////////////////////End of function//////////////////////////////
```

收到 SETUP 包后，我们需要把数据从端点缓冲区中提取出来，这要用到函数 UsbChipReadEndpointBuffer；数据读出来之后，还要清除掉端点的缓冲区，这样它才能接收下一次数据，这要用到函数 UsbChipClearBuffer；而对于 SETUP 包，考虑到它的重要性，需要一个特殊的动作才能清除掉端点缓冲区，这要用到函数 UsbChipAcknowledgeSetup。以下是这三个函数的代码：

```
/*************************************
函数功能：清除接收端点缓冲区的函数。
入口参数：无。
返　　回：无。
备　　注：只有使用该函数清除端点缓冲后，该接收端点才能接收新的数据包。
*************************************/
void UsbChipClearBuffer(uint8 Endp)
{
 if(CurrentBank[Endp] == 0)              //清除 BANK0
 {
  ClrCsr(Endp,1 << 1);                   //将 RX_DATA_BK0 置 0
 }
 else
 {
  ClrCsr(Endp,1 << 6);                   //将 RX_DATA_BK1 置 0
 }
}
//////////////////////////End of function//////////////////////////////
extern uint8 Buffer[16];                 //读端点 0 用的缓冲区
/*************************************
函数功能：应答建立包的函数。
入口参数：Endp：端点号。
返　　回：无。
备　　注：无。
*************************************/
void UsbChipAcknowledgeSetup(uint8 Endp)
{
 SetCsr(Endp,1 << 7);                    //设置 DIR 位
 ClrCsr(Endp,1 << 2);                    //清除 RX SETUP 位
}
//////////////////////////End of function//////////////////////////////
```

```
/*****************************************
函数功能:读取端点缓冲区函数。
入口参数:Endp:端点号;Len:需要读取的长度;Buf:保存数据的缓冲区。
返　　回:实际读到的数据长度。
备　　注:无。
*****************************************/
uint8 UsbChipReadEndpointBuffer(uint8 Endp, uint8 Len, uint8 *Buf)
{
 uint8 i,j;
#ifdef DEBUG2
 Prints("UDP_CSR");
 Printc('0' + Endp);
 Prints(" is ");
 PrintLongIntHex(AT91C_UDP_CSR[Endp]);
 Prints("\r\n");
#endif
 if(! UsbChipIsSetup(Endp))              //如果不是 SETUP 包,则要检查是哪个端点缓冲区
 {
  switch(AT91C_UDP_CSR[Endp]&((1 << 1)|(1 << 6)))
  {
   case ((1 << 1)|(1 << 6)):             //两个缓冲区都满了
    if(CurrentBank[Endp] == 0)           //如果前面读的是 BANK0,那么本次读 BANK1
    {
     CurrentBank[Endp] = 1;
    }
    else                                 //如果前面读的是 BANK1,那么本次读 BANK0
    {
     CurrentBank[Endp] = 0;
    }
   break;
   case (1 << 1):                        //如果只是 BANK0 满,那么就读它
    CurrentBank[Endp] = 0;
   break;
   case (1 << 6):                        //如果只是 BANK1 满,那么就读它
    CurrentBank[Endp] = 1;
   break;
   default:                              //没有缓冲区有数据,则返回 0,没有数据
   return 0;
  }
 }

 j = (AT91C_UDP_CSR[Endp]) >> 16;        //获取数据长度
```

```
 if(j>Len)                              //如果要读的字节数比实际接收到的数据长
 {
  j = Len;                              //则只读指定的长度数据
 }
 #ifdef DEBUG1                          //如果定义了 DEBUG1,则需要显示调试信息
 Prints("读端点");
 PrintLongInt(Endp);
 Prints("缓冲区");
 PrintLongInt(j);                       //实际读取的字节数
 Prints("字节。\r\n");
 #endif
 for(i = 0;i<j;i++)
 {
   *(Buf + i) = AT91C_UDP_FDR[Endp];    //从 FIFO 中读一字节数据
 #ifdef DEBUG1
  PrintHex(*(Buf + i));                 //如果需要显示调试信息,则显示读到的数据
  if(((i + 1) % 16) == 0)Prints("\r\n"); //每 16 字节换行一次
 #endif
 }
 #ifdef DEBUG1
 if((j % 16)! = 0)Prints("\r\n");       //换行。
 #endif
 return j;                              //返回实际读取的字节数。
}
//////////////////////End of function//////////////////////////////
```

收到 SETUP 包之后，需要将数据返回给主机，这就需要用到把数据写到端点的函数 UsbChipWriteEndpointBuffer。由于 ARM 使用的是统一编址，所以这里只需要实现一个函数就可以了，同时将 UsbEp0SendData 函数中的调用和 pSendData 赋值等恢复原样。该函数的代码实现如下：

```
/*****************************************
函数功能:将数据写入端点缓冲区函数。
入口参数:Endp:端点号;Len:需要发送的长度;Buf:保存数据的缓冲区。
返　　回:Len 的值。
备　　注:无。
*****************************************/
uint8 UsbChipWriteEndpointBuffer(uint8 Endp,uint8 Len,uint8 * Buf)
{
 uint8 i;
 #ifdef DEBUG1 //如果定义了 DEBUG1,则需要显示调试信息
 Prints("写端点");
```

```
    PrintLongInt(Endp);
    Prints("缓冲区");
    PrintLongInt(Len);                          //写入的字节数
    Prints("字节。\r\n");
#endif
    for(i = 0;i<Len;i ++ )
    {
     AT91C_UDP_FDR[Endp] = * (Buf + i);         //将数据写到 FIFO 中
#ifdef DEBUG1
     PrintHex( * (Buf + i));                    //如果需要显示调试信息,则显示发送的数据
     if(((i + 1) % 16) == 0)Prints("\r\n");     //每 16 字节换行一次
#endif
    }
#ifdef DEBUG1
    if((Len % 16)! = 0)Prints("\r\n");          //换行
#endif
    UsbChipValidateBuffer(Endp);                //使端点数据有效
    return Len;                                 //返回 Len
}
//////////////////////End of function////////////////////////////
```

在上面的函数中,我们还调用另外一个函数 UsbChipValidateBuffer,它是用来启动端点发送数据的函数。当我们把数据写入到端点缓冲区后,必须通知 USB 控制器:"端点中数据已经写好了,主机下次来取时你就可以发送出去了"。该函数的代码如下:

```
/*********************************
函数功能:使能发送端点缓冲区数据有效的函数。
入口参数:Endp:端点号。
返　　回:无。
备　　注:只有使用该函数使能发送端点数据有效之后,数据才能发送出去。
*********************************/
void UsbChipValidateBuffer(unsigned char Endp)
{
 SetCsr(Endp,1 << 4);       //TXPKTRDY 置 1,使能发送数据
}
//////////////////////End of function////////////////////////////
```

当主机发送设置地址命令后,要将收到的新地址写入到 USB 控制器中,这部分操作需要调用到一个函数 UsbChipWriteAddress,而在该函数中,又会调用到另外一个函数 UsbChipSetAddressStatus,让芯片启用新的地址。这两个函数的代码实现如下:

```
/*****************************************
函数功能:设置芯片进入设置地址状态。
入口参数:Value:地址状态,非 0 为设置地址,0 为未设置地址。
返　　回:无。
备　　注:无。
*****************************************/
void UsbChipSetAddressStatus(uint8 Value)
{
 if(Value == 0)
 {
  *AT91C_UDP_GLBSTATE& = ~(1 << 0);              //默认状态
 }
 else
 {
  *AT91C_UDP_GLBSTATE| = (1 << 0);               //进入到设置地址阶段
 }
}
//////////////////////End of function//////////////////////////////

/*****************************************
函数功能:设置 USB 芯片功能地址函数。
入口参数:Addr:要设置的地址值。
返　　回:无。
备　　注:无。
*****************************************/
void UsbChipWriteAddress(uint8 Addr)
{
 //等待前一个数据包(实际上是状态阶段)发送完毕
 //等待中断产生
 while(1)
 {
  if((*AT91C_UDP_ISR)&(1 << 0))break;            //发送完毕
  if((*AT91C_UDP_ISR)&((1 << 8)|(1 << 12)))return; //如果产生复位、挂起则直接返回
 }

 *AT91C_UDP_FADDR = (1 << 8)|Addr;               //使能功能端点并设置地址
 if(Addr! = 0)                                   //如果地址非 0
 {
  UsbChipSetAddressStatus(1);                    //设置进入到设置地址阶段
 }
}
//////////////////////End of function//////////////////////////////
```

13.3　main.c 和 usbcore.c 的修改

在 main 函数中，同样要对芯片的一些模块进行初始化，然后在主循环中查询 USB 的各个中断寄存器，并调用相关的函数进行处理，这部分代码和之前的代码结构基本上是一样的，具体代码如下：

```
/*****************************************
函数功能：主函数。
入口参数：无。
返　　回：无。
备　　注：无。
*****************************************/
void main(void)
{
#ifdef DEBUG0
 int i;
#endif
 int InterruptSource;
 SystemClockInit();                    //系统时钟初始化
 Uart0Init();                          //串口 0 初始化
#ifdef DEBUG0
 for(i=0;i<15;i++)                     //显示头信息
 {
  Prints(HeadTable[i]);
 }
#endif
 UsbChipInit();                        //初始化 USB 部分
 while(1)
 {
  InterruptSource=(*AT91C_UDP_ISR)&(0x0F|(1<<8)|(1<<12)); //取出需要的中断
  if(InterruptSource)                  //如果监视的中断发生
  {
   if(InterruptSource&(1<<8))
   {
    *AT91C_UDP_ICR=1<<8;               //清除中断
    UsbBusSuspend();                   //总线挂起中断处理
   }
   if(InterruptSource&(1<<12))
```

```
    {
      * AT91C_UDP_ICR = 1 << 12;          //清除中断
      UsbBusReset();                      //总线复位中断处理
    }
    if(InterruptSource&(1 << 0))
    {
      if(AT91C_UDP_CSR[0]&((1 << 1)|(1 << 2)|(1 << 6)))  //如果是 SETUP 包、缓冲未空等
      {
        UsbEp0Out();                      //端点 0 输出中断处理
      }
      if(AT91C_UDP_CSR[0]&(1 << 0))       //如果是端点 0 输入完成
      {
        UsbEp0In();                       //端点 0 输入中断处理
      }
    }
    if(InterruptSource&(1 << 1))
    {
      UsbEp1In();                         //端点 1 输入中断处理
    }
    if(InterruptSource&(1 << 2))
    {
      UsbEp2Out();                        //端点 2 输出中断处理
    }
    if(InterruptSource&(1 << 3))
    {
      UsbEp3In();                         //端点 3 输入中断处理
    }
  }
  if(ConfigValue! = 0)    //如果已经设置为非 0 的配置,则可以返回和发送串口数据
  {
    if(Ep1InIsBusy == 0)                  //如果端点 1 空闲,则发送串口数据到端点 1
    {
      SendUartDataToEp1();                //调用函数将缓冲区数据发送到端点 1
    }
    if(UsbEp2ByteCount! = 0) //端点 2 接收缓冲区中还有数据未发送,则发送到串口
    {
      //发送一字节到串口
      UartPutChar(UsbEp2Buffer[UsbEp2BufferOutputPoint]);
      UsbEp2BufferOutputPoint ++ ;        //发送位置后移 1
      UsbEp2ByteCount -- ;                //计数值减 1
    }
  }
```

```
    }
}
//////////////////////////End of function//////////////////////////////
```

在 usbcore.c 中，主要修改了几个描述符，PID 由原来的 0x2202 改成 0x2105；配置描述符里更换了传输数据的批量端点的地址；函数 UsbEp0SendData 中恢复了之前调用一个函数发送数据的方式；pSendData 变量的定义和赋值也恢复到了原来的方式。

另外还有两个批量端点数据的处理，分别增加了 UsbEp1In 和 UsbEp2Out 函数。当然，相应的缓冲区的变量名也进行了修改。

建议读者还是使用文本对比工具，对源代码进行对比阅读。移植好的 USB 转串口代码见光盘中的 AT91SAM7\ Usb2Uart\文件夹。

13.4　其他几个例子的移植

圈圈已经将 USB 鼠标、键盘、多媒体键盘、MIDI 键盘、假 U 盘、自定义 HID 设备等代码移植好了，分别见光盘目录 AT91SAM7 下的 UsbMouse、UsbKeyboard、UsbMMKeyboard、UsbMidiKeyboard、UsbDiskF、MyUsbHid 文件夹。

13.5　本章小节

本章介绍了将 USB 转串口移植到 AT91SAM7S64 芯片的过程。由于已有上一章的移植经验，本章只做了非常简单的介绍。