小川工作室编写，本书为 LM3S 的 USB 芯片编写，上传的均为草稿，还有没修改，可能还有很多地方不足，希望各位网友原谅！

QQ:2609828265

TEL:15882446438

E-mail:paulhyde@126.com

第三章 底层库函数

3.1 底层库函数简介

寄存器级编程直接、效率高，但不易编写与移植。一般情况下，不使用寄存器级编程。为了让开发者在最短时间内完成产品设计，Luminary Micro Stellaris 外围驱动程序库是一系列用来访问 Stellaris 系列的基于 Cortex-M3 微处理器上的外设的驱动程序。尽管从纯粹的操作系统的理解上它们不是驱动程序，但这些驱动程序确实提供了一种机制，使器件的外设使用起来很容易。

对于许多应用来说，驱动程序直接使用就能满足一般应用的功能、内存或处理要求。外设驱动程序库提供二个编程模型：直接寄存器访问模型和软件驱动程序模型。根据应用的需要或者开发者所需要的编程环境，每个模型可以独立使用或组合使用。

每个编程模型有优点也有弱点。使用直接寄存器访问模型通常得到比使用软件驱动程序模型更少和更高效的代码。然而，直接寄存器访问模型一定要求了解每个寄存器、位段、它们之间的相互作用以及任何一个外设适当操作所需的先后顺序的详细内容；而开发者使用软件驱动程序模型，则不需要知道这些详细内容，通常只需更短的时间开发应用。

驱动程序能够对外设进行完全的控制，在 USB 产品开发时，可以直接使用驱动库函数编程，从而缩短开发周期。开发模型如下图：

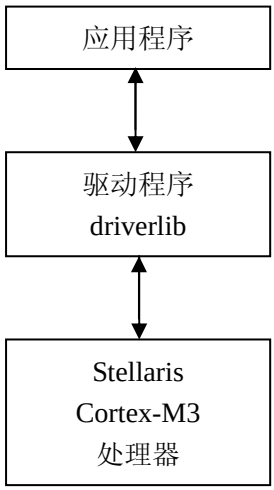

图 1 驱动程序开发模型

驱动程序包含 Stellaris 处理器的全部外设资源控制，下面就 USB 开发过程中会使用到常用底层库函数进行分类说明。

3.2 通用库函数

通用库函数，包含了内核操作、中断控制、GPIO 控制、USB 基本操作。能完成内核控制的全部操作，包器件的时钟、使能的外设、器件的配置、处理复位；能控制嵌套向量中断控制器（NVIC），使能和禁止中断、注册中断处理程序和设置中断的优先级；提供对多达 8 个独立 GPIO 管脚（实际出现的管脚数取决于 GPIO 端口和器件型号）的控制；能进行寄存器级操作 USB 外设模块。

3.2.1 内核操作

在处理器使用之前要进行必要的系统配置，包括内核电压、CPU 主频、外设资源使能等；在应用程序开发中还常常用到获取系统时钟、延时等操作。这些操作都通过内核系统操作函数进行访问。

void SysCtlLDOSet(unsigned long ulVoltage)

作用：配置内核电压。

参数：ulVoltage，内核电压参数。在使用 PLL 之前，最好设置内核电压为 2.75V。保证处理器稳定。

返回：无

例如：设置 CPU 主机为 2.75V

```
SysCtlLDOSet(SYSCTL_LDO_2_75V);
```

void SysCtlClockSet(unsigned long ulConfig)

作用：配置 CPU 统主频。

参数：ulConfig，时钟配置。格式：（振荡源 | 晶体频率 | PLL 是否使用 | 分频）。如果使用 PLL，则配置的时钟 = 200Mhz / 分频。

返回：无

例如：外接晶振 8MHz，设置系统频率为 50MHz。（200MHz / 4 = 50MHz）

```
SysCtlClockSet(SYSCTL_XTAL_8MHZ | SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );
```

unsigned long SysCtlClockGet(void)

作用：获取 CPU 统主频。

参数：无。
返回：当前 CPU 主频。

void SysCtlPeripheralEnable(unsigned long ulPeripheral)
作用：使能外设资源，外设使用前必须先使能，使能对应外设资源时钟。
参数：ulPeripheral，外设资源。
返回：无
例如：使能 GPIO 的端口 A；使能 USB0 外设资源。

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_USB0);
```

void SysCtlDelay(unsigned long ulCount)
作用：延时 ulCount ＊ 3 个系统时钟周期。
参数：ulCount，延时计数。延时为 ulCount ＊ 3 个系统时钟周期。
返回：无
例如：延时 1 秒。

```
SysCtlDelay(SysCtlClockGet() / 3);
```

例如：在使用 USB 设备时会先配置处理器，使用本节函数完成系统配置：外接晶振 6MHz，设置内核电压为 2.75V，配置 CPU 主频为 25MHz，使能 USB0 模块，使能端口 F，并延时 100ms。

```
SysCtlLDOSet(SYSCTL_LDO_2_75V);
SysCtlClockSet(SYSCTL_XTAL_6MHZ | SYSCTL_SYSDIV_8 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );
SysCtlPeripheralEnable(SYSCTL_PERIPH_USB0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
SysCtlDelay(SysCtlClockGet() / 30);
```

3.2.2 系统中断控制

下面介绍两组中断函数，进行处理器中断控制。对于 Stellaris 处理器，中断类型很多、控制相当复杂。中断层次如下图：

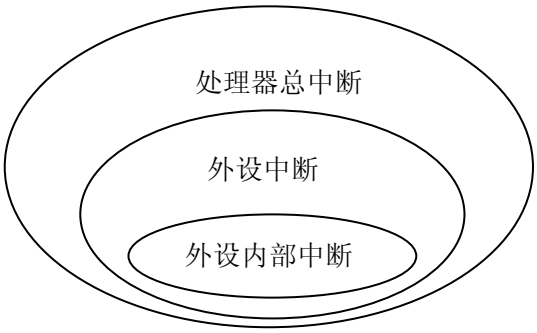

图 2 中断控制模型

void IntEnable(unsigned long ulInterrupt)
作用：使能外设资源中断。
参数：ulInterrupt，外设资源中断。
返回：无
例如：使能 USB0 中断。

```
IntEnable(INT_USB0);
```

void IntDisable(unsigned long ulInterrupt)
作用：禁止外设资源中断。
参数：ulInterrupt，外设资源中断。
返回：无
例如：禁止 USB0 中断。

```
IntDisable (INT_USB0);
```

tBoolean IntMasterEnable(void)
作用：使能总中断。
参数：无。
返回：中断使能是否成功。
例如：使能总中断。

```
IntMasterEnable();
```

tBoolean IntMasterDisable(void)
作用：禁止总中断。
参数：无。
返回：中断禁止是否成功。
例如：禁止总中断。

```
IntMasterDisable();
```

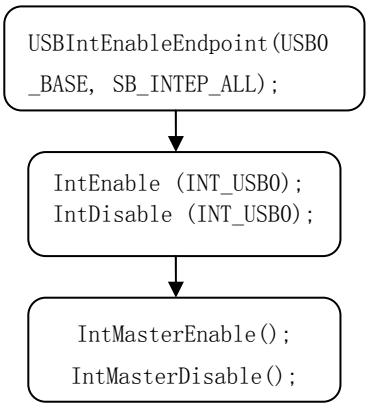

图 3 中断控制实际操作模型

从图 2 可以看出，处理器中断控制必须进行三层配置，图 3 显示中断控制的实际操作过程和配置顺序为从上到下以次配置。首先配置外设模块内部中断，比如 USBIntEnableEndpoint 使能 USB 模块的端点中断；再配置外设中断，比如 IntEnable 使能 USB0 模块中断；最后配置总中断，IntMasterEnable 使能总中断，如果不使能总中断，系统外设中断和外设内部中断都无法正常工作。

3.2.3 GPIO 控制

GPIO，通用输入输出端口。对于处理器来说，GPIO 占有重要的应用地位。无论是通信还是控制，都离不开 GPIO。Stellaris GPIO 模块由 7 个物理 GPIO 模块组成，分别对端口

A、 端口 B、 端口 C、 端口 D、 端口 E、 端口 F 和端口 G，各种 Stellaris Cortex-3 的模块数量不一样，具体模块数请参考对应数据手册；每个模块支持 8 个通用输入输出端口（GPIO），即 GPIO_PIN_0 到 GPIO_PIN_7，Stellaris Cortex-3 可支持 5-42 个独立的 GPIO；每个 GPIO 可以根据具体使用情况独立配置成输入\输出。每种 Stellaris Cortex-3 模块数量、模块上的 GPIO 数量不一样，但是使用方法都一样。

在使用下列函数前，必须 SysCtlPeripheralEnable 使能对应 GPIO 模块。

```
void GPIOPinTypeGPIOInput(unsigned long ulPort, unsigned char ucPins);
void GPIOPinTypeGPIOOutput(unsigned long ulPort, unsigned char ucPins);
void GPIOPinTypeUSBAnalog(unsigned long ulPort, unsigned char ucPins);
void GPIOPinTypeUSBDigital(unsigned long ulPort, unsigned char ucPins);
long GPIOPinRead(unsigned long ulPort, unsigned char ucPins);
void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins,
                  unsigned char ucVal);
void GPIOIntTypeSet(unsigned long ulPort, unsigned char ucPins,
                    unsigned long ulIntType);
void GPIOPinIntEnable(unsigned long ulPort, unsigned char ucPins);
void GPIOPinIntDisable(unsigned long ulPort, unsigned char ucPins);
long GPIOPinIntStatus(unsigned long ulPort, tBoolean bMasked);
void GPIOPinIntClear(unsigned long ulPort, unsigned char ucPins);
```

引脚定义：

```
#define GPIO_PIN_0              0x00000001  // GPIO pin 0
#define GPIO_PIN_1              0x00000002  // GPIO pin 1
#define GPIO_PIN_2              0x00000004  // GPIO pin 2
#define GPIO_PIN_3              0x00000008  // GPIO pin 3
#define GPIO_PIN_4              0x00000010  // GPIO pin 4
#define GPIO_PIN_5              0x00000020  // GPIO pin 5
#define GPIO_PIN_6              0x00000040  // GPIO pin 6
#define GPIO_PIN_7              0x00000080  // GPIO pin 7
```

void GPIOPinTypeGPIOInput(unsigned long ulPort, unsigned char ucPins);

作用：把 ulPort 端口上的 ucPins 引脚设置为输入。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）。

返回：无。

例如：设置端口 A 的 3、4 引脚为输入。

```
GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, (GPIO_PIN_3 | GPIO_PIN_4) );
```

void GPIOPinTypeUSBAnalog(unsigned long ulPort, unsigned char ucPins);

作用：把 ulPort 端口上的 ucPins 引脚配置为 USB 使用的模拟信号引脚。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）。

返回：无。

void GPIOPinTypeUSBDigital (unsigned long ulPort, unsigned char ucPins);

作用：把 ulPort 端口上的 ucPins 引脚配置为 USB 使用的数字信号引脚。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）。

返回：无。

GPIOPinTypeUSBAnalog 和 GPIOPinTypeUSBDigital 不能把所有引脚都设置为 USB 功能引脚，必须要相应引脚有 USB 引脚功能，才可配置。不同 USB 处理器 USB 功能引脚不一样，具体情况请参考对应芯片的数据手册。由于以上两个函数不常用，在使用 lm3s3xxxx 与 lm3s5xxxx 系列时一般不使用。Lm3s9xxxx 系列或以后更高版本由于引脚复用情况较多，所以必须使用以上两个函数，确定 USB 功能引脚。

void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins,
unsigned char ucVal);

作用：向 ulPort 端口上的 ucPins 引脚写入 ucVal 值。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）；ucVal，待写入的数据。

返回：无。

long GPIOPinRead(unsigned long ulPort, unsigned char ucPins);

作用：把 ulPort 端口的 ucPins 引脚状态读出。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）。

返回：ucPins 指定引脚状态组合。

例如： A 口输入数据从 B 口输出。

```
GPIOPinWrite(GPIO_PORTB_BASE,0xff, GPIOPinRead(GPIO_PORTA_BASE,0xff));
//GPIO 中断类型
#define GPIO_FALLING_EDGE       0x00000000  // Interrupt on falling edge
#define GPIO_RISING_EDGE        0x00000004  // Interrupt on rising edge
#define GPIO_BOTH_EDGES         0x00000001  // Interrupt on both edges
#define GPIO_LOW_LEVEL          0x00000002  // Interrupt on low level
#define GPIO_HIGH_LEVEL         0x00000007  // Interrupt on high level
```

void GPIOIntTypeSet(unsigned long ulPort, unsigned char ucPins,
unsigned long ulIntType);

作用：设置 ulPort 端口的 ucPins 引脚中断类型。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）；ulIntType，指定中断类型。

返回：无

void GPIOPinIntEnable(unsigned long ulPort, unsigned char ucPins);

作用：使能 ulPort 端口的 ucPins 引脚中断。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）。

返回：无

void GPIOPinIntDisable(unsigned long ulPort, unsigned char ucPins);

作用：禁止 ulPort 端口的 ucPins 引脚中断。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）。

返回：无

long GPIOPinIntStatus(unsigned long ulPort, tBoolean bMasked);

作用：获取 ulPort 端口的中断状态。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；bMasked，是否获取屏蔽后中断状态，一般使用 true。

返回：引脚中断状态。

void GPIOPinIntClear(unsigned long ulPort, unsigned char ucPins);

作用：清除 ulPort 端口的 ucPins 引脚中断标志。

参数：ulPort，指定端口，GPIO_PORTn_BASE，n 取对应端口号（A-G）；ucPins，指定控制引脚，GPIO_PIN_n，n 取对应引脚号（0-7）。

返回：无

例如：使用已学的函数，利用 GPIO 中断，完成 PC4-7 上按键控制 PB0、PB1、PB4、PB5 上 LED 灯的任务。KEY1 控制 LED1，KEY2 控制 LED2，KEY3 控制 LED3，KEY3 控制 LED4，当按键按下，对应 LED 取当前的相反状态，如 LED 现在亮，当按下 KEY 时，对应的 LED 灭，再次按下 KEY 时，LED 又亮。

```
#include <sysctl.h>
#include <gpio.h>
#include <hw_memmap.h>
#include <hw_ints.h>
#include <interrupt.h>
#include <lm3s8962.h>
#define  u32      unsigned long
//LED 引脚定义   PORTB
#define  LED1     GPIO_PIN_1
#define  LED2     GPIO_PIN_4
#define  LED3     GPIO_PIN_5
#define  LED4     GPIO_PIN_6
#define  LED      (LED1 | LED2 | LED3 | LED4)
//按键引脚定义    PORTC
#define  KEY1     GPIO_PIN_4
#define  KEY2     GPIO_PIN_5
#define  KEY3     GPIO_PIN_6
#define  KEY4     GPIO_PIN_7
#define  KEY      (KEY1 | KEY2 | KEY3 | KEY4)
int main(void)
{
  //设置内核电压与 CPU 主频。
  SysCtlLDOSet(SYSCTL_LDO_2_75V);
  SysCtlClockSet(SYSCTL_XTAL_6MHZ | SYSCTL_SYSDIV_8 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );
  //使能 GPIO 外设
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
  //LED IO  设置
  GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,LED);
```

```
    //KEY IO  设置
    GPIOPinTypeGPIOInput(GPIO_PORTC_BASE,KEY);
    //KEY 中断设置,配置为上升沿触发。
    GPIOIntTypeSet(GPIO_PORTC_BASE,KEY,GPIO_RISING_EDGE);
    GPIOPinIntEnable(GPIO_PORTC_BASE,KEY);
    //打开端口 C 中断
    IntEnable(INT_GPIOC);
    //打开全局中断。
    IntMasterEnable();
    //打开所有 LED
    GPIOPinWrite(GPIO_PORTB_BASE,LED,LED);
    //等待中断。
    while(1)
    {
    }
}
void GPIO_Port_C_ISR(void)
{
    u32 ulStatus = GPIOPinIntStatus(GPIO_PORTC_BASE,1);
    GPIOPinIntClear(GPIO_PORTC_BASE,KEY);
    if(ulStatus & KEY1)
    {
      GPIOPinWrite(GPIO_PORTB_BASE,LED1,~GPIOPinRead(GPIO_PORTB_BASE,LED1));
    }
    else if(ulStatus & KEY2)
      {
          GPIOPinWrite(GPIO_PORTB_BASE,LED2,~GPIOPinRead(GPIO_PORTB_BASE,LED2));
      }
      else if(ulStatus & KEY3)
      {
          GPIOPinWrite(GPIO_PORTB_BASE,LED3,~GPIOPinRead(GPIO_PORTB_BASE,LED3));
      }
      else if(ulStatus & KEY4)
      {
          GPIOPinWrite(GPIO_PORTB_BASE,LED4,~GPIOPinRead(GPIO_PORTB_BASE,LED4));
      }
}
```

掌握好以上 API 后，能够为外设资源控制打好基础，为学习 USB 底层驱动 API 函数打好基础。在实际 USB 工程开发中，以上 API 函数使用次数较多，在开发中占有重要地位。有关 MCU 外设的其它 API 函数请参考其它书籍。

3.3 USB 基本操作

USB API 提供了用来访问 Stellaris USB 设备控制器或主机控制器的函数集。API 分组如下：USBDev、USBHost 、USBOTG、USBEndpoint 和 USBFIFO。USB 设备控制器只使用 USBDev

组 API；USB 主机控制器只使用 USBHost 中 API； OTG 接口的微控制器使用 USBOTG 组 API 。USB OTG 微控制器，一旦配置完，则使用设备或主机 API。余下的 API 均可被 USB 主机和 USB 设备控制器使用,USBEndpoint 组 API 一般用来配置和访问端点,USBFIFO 组 API 则配置 FIFO 大小和位置。所以在本书中把 USB API 分为三类：USB 基本操作 API、设备库函数 API、主机库函数 API。“USB 基本操作 API”包含设备和主机都使用的 API：USBEndpoint 组 API、USBFIFO 组 API、其它公用 API。“设备库函数 API”只包含设备能够使用的 API 函数。“主机库函数 API”只包含主机能够使用的 API 函数。OTG 配置完成时，使用“设备库函数 API”和“主机库函数 AP”。

本节主要介绍 USB 基本操作 API，包含 USBEndpoint 组 API、USBFIFO 组 API、其它公用 API。

端点控制函数：

```
unsigned long USBEndpointDataAvail(unsigned long ulBase, unsigned long ulEndpoint);
long USBEndpointDataGet(unsigned long ulBase, unsigned long ulEndpoint,
                        unsigned char *pucData, unsigned long *pulSize);
long USBEndpointDataPut(unsigned long ulBase, unsigned long ulEndpoint,
                        unsigned char *pucData, unsigned long ulSize);
long USBEndpointDataSend(unsigned long ulBase, unsigned long ulEndpoint,
                         unsigned long ulTransType);
void USBEndpointDataToggleClear(unsigned long ulBase,
                                unsigned long ulEndpoint,
                                unsigned long ulFlags);
unsigned long USBEndpointStatus(unsigned long ulBase,
                                unsigned long ulEndpoint);
void USBEndpointDMAChannel(unsigned long ulBase,
                           unsigned long ulEndpoint,
                           unsigned long ulChannel);
void USBEndpointDMAEnable(unsigned long ulBase, unsigned long ulEndpoint,
                          unsigned long ulFlags);
void USBEndpointDMADisable(unsigned long ulBase,
                           unsigned long ulEndpoint,
                           unsigned long ulFlags);
```

FIFO 控制函数：

```
unsigned long USBFIFOAddrGet(unsigned long ulBase,
                             unsigned long ulEndpoint);
void USBFIFOConfigSet(unsigned long ulBase, unsigned long ulEndpoint,
                      unsigned long ulFIFOAddress,
                      unsigned long ulFIFOSize, unsigned long ulFlags);
void USBFIFOConfigGet(unsigned long ulBase, unsigned long ulEndpoint,
                      unsigned long *pulFIFOAddress,
                      unsigned long *pulFIFOSize,
                      unsigned long ulFlags);
void USBFIFOFlush(unsigned long ulBase, unsigned long ulEndpoint,
                  unsigned long ulFlags);
```

中断控制函数：

```
void USBIntEnableControl(unsigned long ulBase,
                         unsigned long ulIntFlags);
void USBIntDisableControl(unsigned long ulBase,
                          unsigned long ulIntFlags);
unsigned long USBIntStatusControl(unsigned long ulBase);
void USBIntEnableEndpoint(unsigned long ulBase,
                          unsigned long ulIntFlags);
void USBIntDisableEndpoint(unsigned long ulBase,
                           unsigned long ulIntFlags);
unsigned long USBIntStatusEndpoint(unsigned long ulBase);
void USBIntEnable(unsigned long ulBase, unsigned long ulIntFlags);
void USBIntDisable(unsigned long ulBase, unsigned long ulIntFlags);
unsigned long USBIntStatus(unsigned long ulBase);
```

其它控制函数：

```
unsigned long USBFrameNumberGet(unsigned long ulBase);
void USBOTGSessionRequest(unsigned long ulBase, tBoolean bStart);
unsigned long USBModeGet(unsigned long ulBase);
```

端点控制函数包含所有端点控制：端点设置、端点数据发送与接收、端点 DMA 控制等。是 USB 通信最基本的一组函数。使用下面函数前要进行端点控制，如果是 USB 主机，使用 USBHostEndpointConfig 配置主机端点；如果是 USB 设备，使用 USBDevEndpointConfigSet 配置设备端点。

```
unsigned long USBEndpointDataAvail(unsigned long ulBase,
                                   unsigned long ulEndpoint)
```

作用：检查接收端点中有多少数据可用。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定接收端点，USB_EP_n（n=0..15）。

返回：接收端点中的可用数据个数。

```
long USBEndpointDataGet(unsigned long ulBase, unsigned long ulEndpoint,
                        unsigned char *pucData, unsigned long *pulSize);
```

作用：从 ulEndpoint 的 RXFIFO 中读取 pulSize 指定长度的数据到 pucData 指定数据中。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定接收端点，USB_EP_n（n=0..15）。pucData 指定接收数据的数组指针。pulSize 指定接收数据的长度。

返回：函数是否成功执行。

```
long USBEndpointDataPut(unsigned long ulBase, unsigned long ulEndpoint,
                        unsigned char *pucData, unsigned long ulSize);
```

作用：pucData 中的 ulSize 个数据放入 ulEndpoint 的 TXFIFO 中，等待发送。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。pucData 指定待发送数据的数组指针。pulSize 指定发送数据的长度。

返回：函数是否成功执行。

```
long USBEndpointDataSend(unsigned long ulBase, unsigned long ulEndpoint,
                                               unsigned long ulTransType);
```

作用：触发 ulEndpoint 的 TXFIFO 发送数据。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulTransType，指定发送类型。

返回：函数是否成功执行。

ulTransType 可选参数：

```
#define USB_TRANS_OUT           0x00000102  // Normal OUT transaction
#define USB_TRANS_IN            0x00000102  // Normal IN transaction
#define USB_TRANS_IN_LAST       0x0000010a  // Final IN transaction (for
                                            // endpoint 0 in device mode)
#define USB_TRANS_SETUP         0x0000110a  // Setup transaction (for endpoint
                                            // 0)
#define USB_TRANS_STATUS        0x00000142  // Status transaction (for endpoint
                                            // 0)
```

```
void USBEndpointDataToggleClear(unsigned long ulBase,
                                unsigned long ulEndpoint,
                                unsigned long ulFlags);
```

作用：清除 Toggle。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，指定访问 IN/OUT 端点。

返回：无。

```
unsigned long USBEndpointStatus(unsigned long ulBase,
                                unsigned long ulEndpoint);
```

作用：获取 ulEndpoint 指定端点的状态。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。

返回：ulEndpoint 端点的状态。

在设备模式下返回 USB_DEV_XX，主机模式下返回 USB_HOST_XX。返回数据：

```
#define USB_HOST_IN_PID_ERROR   0x01000000  // Stall on this endpoint received
#define USB_HOST_IN_NOT_COMP    0x00100000  // Device failed to respond
#define USB_HOST_IN_STALL       0x00400000  // Stall on this endpoint received
#define USB_HOST_IN_DATA_ERROR  0x00080000  // CRC or bit-stuff error
                                            // (ISOC Mode)
#define USB_HOST_IN_NAK_TO      0x00080000  // NAK received for more than the
                                            // specified timeout period
#define USB_HOST_IN_ERROR       0x00040000  // Failed to communicate with a
                                            // device
#define USB_HOST_IN_FIFO_FULL   0x00020000  // RX FIFO full
#define USB_HOST_IN_PKTRDY      0x00010000  // Data packet ready
#define USB_HOST_OUT_NAK_TO     0x00000080  // NAK received for more than the
                                            // specified timeout period
#define USB_HOST_OUT_NOT_COMP   0x00000080  // No response from device
```

```
                                        // (ISOC mode)
#define USB_HOST_OUT_STALL      0x00000020  // Stall on this endpoint received
#define USB_HOST_OUT_ERROR      0x00000004  // Failed to communicate with a
                                            // device
#define USB_HOST_OUT_FIFO_NE    0x00000002  // TX FIFO is not empty
#define USB_HOST_OUT_PKTPEND    0x00000001  // Transmit still being transmitted
#define USB_HOST_EP0_NAK_TO     0x00000080  // NAK received for more than the
                                            // specified timeout period
#define USB_HOST_EP0_STATUS     0x00000040  // This was a status packet
#define USB_HOST_EP0_ERROR      0x00000010  // Failed to communicate with a
                                            // device
#define USB_HOST_EP0_RX_STALL   0x00000004  // Stall on this endpoint received
#define USB_HOST_EP0_RXPKTRDY   0x00000001  // Receive data packet ready
#define USB_DEV_RX_SENT_STALL   0x00400000  // Stall was sent on this endpoint
#define USB_DEV_RX_DATA_ERROR   0x00080000  // CRC error on the data
#define USB_DEV_RX_OVERRUN      0x00040000  // OUT packet was not loaded due to
                                            // a full FIFO
#define USB_DEV_RX_FIFO_FULL    0x00020000  // RX FIFO full
#define USB_DEV_RX_PKT_RDY      0x00010000  // Data packet ready
#define USB_DEV_TX_NOT_COMP     0x00000080  // Large packet split up, more data
                                            // to come
#define USB_DEV_TX_SENT_STALL   0x00000020  // Stall was sent on this endpoint
#define USB_DEV_TX_UNDERRUN     0x00000004  // IN received with no data ready
#define USB_DEV_TX_FIFO_NE      0x00000002  // The TX FIFO is not empty
#define USB_DEV_TX_TXPKTRDY     0x00000001  // Transmit still being transmitted
#define USB_DEV_EP0_SETUP_END   0x00000010  // Control transaction ended before
                                            // Data End seen
#define USB_DEV_EP0_SENT_STALL  0x00000004  // Stall was sent on this endpoint
#define USB_DEV_EP0_IN_PKTPEND  0x00000002  // Transmit data packet pending
#define USB_DEV_EP0_OUT_PKTRDY  0x00000001  // Receive data packet ready
void USBEndpointDMAChannel(unsigned long ulBase,
                           unsigned long ulEndpoint,
                           unsigned long ulChannel);
```

作用：端点 DMA 控制。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulChannel，指定 DMA 通过。

返回：无。

```
void USBEndpointDMAEnable(unsigned long ulBase, unsigned long ulEndpoint,
                          unsigned long ulFlags);
```

作用：端点 DMA 使能。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。

返回：无。

```
void USBEndpointDMADisable(unsigned long ulBase,
                           unsigned long ulEndpoint,
                           unsigned long ulFlags);
```

作用：端点 DMA 禁止。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。

返回：无。

在 USB 通信中，实际是端点与端点间通信，掌握端点控制函数是非常必要的。例如，在枚举过程中，会大量使用端点 0 与主机进行数据传输。下面程序介绍了枚举过程中端点 0 的数据传输。

```
tDeviceInstance g_psUSBDevice[1];
static void USBDEP0StateTx(unsigned long ulIndex)
{
    unsigned long ulNumBytes;
    unsigned char *pData;
    g_psUSBDevice[0].eEP0State = USB_STATE_TX;
    //设置端点待发送。
    ulNumBytes = g_psUSBDevice[0].ulEP0DataRemain;
    //端点 0 最大传输包为 64，传输数据包大于 64，就分批次传输。
    if(ulNumBytes > EP0_MAX_PACKET_SIZE)
    {
        ulNumBytes = EP0_MAX_PACKET_SIZE;
    }
    //数据指针，指向待发送数据组。
    pData = (unsigned char *)g_psUSBDevice[0].pEP0Data;
    //分批次发送
    g_psUSBDevice[0].ulEP0DataRemain -= ulNumBytes;
    g_psUSBDevice[0].pEP0Data += ulNumBytes;
    //把刚才数据放入端点 0 的 TXFIFO 中
    USBEndpointDataPut(USB0_BASE, USB_EP_0, pData, ulNumBytes);
    // 判断 ulNumBytes 大小，如果等于最大包长度。
    if(ulNumBytes == EP0_MAX_PACKET_SIZE)
    {
        //普通发送，对于主机来说，设备通过输入端点发送。
        USBEndpointDataSend(USB0_BASE, USB_EP_0, USB_TRANS_IN);
    }
    else
    {
        g_psUSBDevice[0].eEP0State = USB_STATE_STATUS;
        //发送数据结束，并发送标志位
        USBEndpointDataSend(USB0_BASE, USB_EP_0, USB_TRANS_IN_LAST);
        // 判断是否有 callback 函数。
        if((g_psUSBDevice[0].psInfo->sCallbacks.pfnDataSent) &&
           (g_psUSBDevice[0].ulOUTDataSize != 0))
```

```
            {
                //通过 callback 告诉应用程序，发送结束。
                g_psUSBDevice[0].psInfo->sCallbacks.pfnDataSent(
                    g_psUSBDevice[0].pvInstance, g_psUSBDevice[0].ulOUTDataSize);
                g_psUSBDevice[0].ulOUTDataSize = 0;
            }
        }
    }
```

从上面程序中可以看出，如果发送数据大于端点最大包大小，则分批次发送，发送到最后一个数据包时，要传送包结束标志。

```
unsigned long USBFIFOAddrGet(unsigned long ulBase,
                             unsigned long ulEndpoint);
```

作用：通过端点获取端点对应的 FIFO 地址。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。

返回：FIFO 地址。

```
void USBFIFOConfigSet(unsigned long ulBase, unsigned long ulEndpoint,
                                            unsigned long ulFIFOAddress,
                                            unsigned long ulFIFOSize,
                                            unsigned long ulFlags);
```

作用：FIFO 配置。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFIFOAddress，配置 ulEndpoint 端点的 FIFO 首地址。ulFIFOSize，FIFO 大小。ulFlags，端点类型。

返回：无。

ulFIFOSize 参数：

```
#define USB_FIFO_SZ_8           0x00000000  // 8 byte FIFO
#define USB_FIFO_SZ_16          0x00000001  // 16 byte FIFO
#define USB_FIFO_SZ_32          0x00000002  // 32 byte FIFO
#define USB_FIFO_SZ_64          0x00000003  // 64 byte FIFO
#define USB_FIFO_SZ_128         0x00000004  // 128 byte FIFO
#define USB_FIFO_SZ_256         0x00000005  // 256 byte FIFO
#define USB_FIFO_SZ_512         0x00000006  // 512 byte FIFO
#define USB_FIFO_SZ_1024        0x00000007  // 1024 byte FIFO
#define USB_FIFO_SZ_2048        0x00000008  // 2048 byte FIFO
#define USB_FIFO_SZ_4096        0x00000009  // 4096 byte FIFO
#define USB_FIFO_SZ_8_DB        0x00000010  // 8 byte double buffered FIFO
#define USB_FIFO_SZ_16_DB       0x00000011  // 16 byte double buffered FIFO
#define USB_FIFO_SZ_32_DB       0x00000012  // 32 byte double buffered FIFO
#define USB_FIFO_SZ_64_DB       0x00000013  // 64 byte double buffered FIFO
#define USB_FIFO_SZ_128_DB      0x00000014  // 128 byte double buffered FIFO
#define USB_FIFO_SZ_256_DB      0x00000015  // 256 byte double buffered FIFO
#define USB_FIFO_SZ_512_DB      0x00000016  // 512 byte double buffered FIFO
```

```
#define USB_FIFO_SZ_1024_DB     0x00000017  // 1024 byte double buffered FIFO
#define USB_FIFO_SZ_2048_DB     0x00000018  // 2048 byte double buffered FIFO
```

ulFlags 参数：

```
#define USB_EP_AUTO_SET         0x00000001  // Auto set feature enabled
#define USB_EP_AUTO_REQUEST     0x00000002  // Auto request feature enabled
#define USB_EP_AUTO_CLEAR       0x00000004  // Auto clear feature enabled
#define USB_EP_DMA_MODE_0       0x00000008  // Enable DMA access using mode 0
#define USB_EP_DMA_MODE_1       0x00000010  // Enable DMA access using mode 1
#define USB_EP_MODE_ISOC        0x00000000  // Isochronous endpoint
#define USB_EP_MODE_BULK        0x00000100  // Bulk endpoint
#define USB_EP_MODE_INT         0x00000200  // Interrupt endpoint
#define USB_EP_MODE_CTRL        0x00000300  // Control endpoint
#define USB_EP_MODE_MASK        0x00000300  // Mode Mask
#define USB_EP_SPEED_LOW        0x00000000  // Low Speed
#define USB_EP_SPEED_FULL       0x00001000  // Full Speed
#define USB_EP_HOST_IN          0x00000000  // Host IN endpoint
#define USB_EP_HOST_OUT         0x00002000  // Host OUT endpoint
#define USB_EP_DEV_IN           0x00002000  // Device IN endpoint
#define USB_EP_DEV_OUT          0x00000000  // Device OUT endpoint
```

```
void USBFIFOConfigGet(unsigned long ulBase, unsigned long ulEndpoint,
                                            unsigned long *pulFIFOAddress,
                                            unsigned long *pulFIFOSize,
                                            unsigned long ulFlags);
```

作用：获取 FIFO 配置。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFIFOAddress，获取 ulEndpoint 端点的 FIFO 首地址。ulFIFOSize，获取 FIFO 大小。ulFlags，端点类型。

返回：无。

```
void USBFIFOFlush(unsigned long ulBase, unsigned long ulEndpoint,
                                        unsigned long ulFlags);
```

作用：清空 FIFO。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，端点类型。

返回：无。

中断控制在 USB 开发中具有重要地位，USB 处理速度高达 12MHz，每一种状态都可以触发中断，并让处理器进行数据处理；USB 中断类型多于 20 个。在这种复杂的控制中，中断管理和相应数据处理是非常重要的。下面这些函数可以有效的进行 USB 中断控制。在使用这些函数前必须使能处理器总中断和外设模块中断。

```
void USBIntEnableControl(unsigned long ulBase, unsigned long ulIntFlags);
```

作用：使能 USB 通用中断，除端点中断外的所有 USB 外设内部中断。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulIntFlags，中断标志。

返回：无。

ulIntFlags 可用参数：

```
#define USB_INTCTRL_ALL         0x000003FF  // All control interrupt sources
#define USB_INTCTRL_STATUS      0x000000FF  // Status Interrupts
#define USB_INTCTRL_VBUS_ERR    0x00000080  // VBUS Error
#define USB_INTCTRL_SESSION     0x00000040  // Session Start Detected
#define USB_INTCTRL_SESSION_END 0x00000040  // Session End Detected
#define USB_INTCTRL_DISCONNECT  0x00000020  // Disconnect Detected
#define USB_INTCTRL_CONNECT     0x00000010  // Device Connect Detected
#define USB_INTCTRL_SOF         0x00000008  // Start of Frame Detected
#define USB_INTCTRL_BABBLE      0x00000004  // Babble signaled
#define USB_INTCTRL_RESET       0x00000004  // Reset signaled
#define USB_INTCTRL_RESUME      0x00000002  // Resume detected
#define USB_INTCTRL_SUSPEND     0x00000001  // Suspend detected
#define USB_INTCTRL_MODE_DETECT 0x00000200  // Mode value valid
#define USB_INTCTRL_POWER_FAULT 0x00000100  // Power Fault detected
```

void USBIntDisableControl(unsigned long ulBase, unsigned long ulIntFlags);

作用：禁止 USB 通用中断，除端点中断外的所有 USB 外设内部中断。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulIntFlags，中断标志。

返回：无。

unsigned long USBIntStatusControl(unsigned long ulBase);

作用：获取中断标志。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：返回中断标志，与 USBIntEnableControl 的 ulIntFlags 标志相同。

void USBIntEnableEndpoint(unsigned long ulBase, unsigned long ulIntFlags);

作用：使能 USB 端点中断。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulIntFlags，中断标志。

返回：无。

ulIntFlags 可选参数：

```
#define USB_INTEP_ALL           0xFFFFFFFF  // Host IN Interrupts
#define USB_INTEP_HOST_IN       0xFFFE0000  // Host IN Interrupts
#define USB_INTEP_HOST_IN_15    0x80000000  // Endpoint 15 Host IN Interrupt
#define USB_INTEP_HOST_IN_14    0x40000000  // Endpoint 14 Host IN Interrupt
#define USB_INTEP_HOST_IN_13    0x20000000  // Endpoint 13 Host IN Interrupt
#define USB_INTEP_HOST_IN_12    0x10000000  // Endpoint 12 Host IN Interrupt
#define USB_INTEP_HOST_IN_11    0x08000000  // Endpoint 11 Host IN Interrupt
#define USB_INTEP_HOST_IN_10    0x04000000  // Endpoint 10 Host IN Interrupt
#define USB_INTEP_HOST_IN_9     0x02000000  // Endpoint 9 Host IN Interrupt
```

```
#define USB_INTEP_HOST_IN_8     0x01000000  // Endpoint 8 Host IN Interrupt
#define USB_INTEP_HOST_IN_7     0x00800000  // Endpoint 7 Host IN Interrupt
#define USB_INTEP_HOST_IN_6     0x00400000  // Endpoint 6 Host IN Interrupt
#define USB_INTEP_HOST_IN_5     0x00200000  // Endpoint 5 Host IN Interrupt
#define USB_INTEP_HOST_IN_4     0x00100000  // Endpoint 4 Host IN Interrupt
#define USB_INTEP_HOST_IN_3     0x00080000  // Endpoint 3 Host IN Interrupt
#define USB_INTEP_HOST_IN_2     0x00040000  // Endpoint 2 Host IN Interrupt
#define USB_INTEP_HOST_IN_1     0x00020000  // Endpoint 1 Host IN Interrupt
#define USB_INTEP_DEV_OUT       0xFFFE0000  // Device OUT Interrupts
#define USB_INTEP_DEV_OUT_15    0x80000000  // Endpoint 15 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_14    0x40000000  // Endpoint 14 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_13    0x20000000  // Endpoint 13 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_12    0x10000000  // Endpoint 12 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_11    0x08000000  // Endpoint 11 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_10    0x04000000  // Endpoint 10 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_9     0x02000000  // Endpoint 9 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_8     0x01000000  // Endpoint 8 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_7     0x00800000  // Endpoint 7 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_6     0x00400000  // Endpoint 6 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_5     0x00200000  // Endpoint 5 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_4     0x00100000  // Endpoint 4 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_3     0x00080000  // Endpoint 3 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_2     0x00040000  // Endpoint 2 Device OUT Interrupt
#define USB_INTEP_DEV_OUT_1     0x00020000  // Endpoint 1 Device OUT Interrupt
#define USB_INTEP_HOST_OUT      0x0000FFFE  // Host OUT Interrupts
#define USB_INTEP_HOST_OUT_15   0x00008000  // Endpoint 15 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_14   0x00004000  // Endpoint 14 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_13   0x00002000  // Endpoint 13 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_12   0x00001000  // Endpoint 12 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_11   0x00000800  // Endpoint 11 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_10   0x00000400  // Endpoint 10 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_9    0x00000200  // Endpoint 9 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_8    0x00000100  // Endpoint 8 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_7    0x00000080  // Endpoint 7 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_6    0x00000040  // Endpoint 6 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_5    0x00000020  // Endpoint 5 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_4    0x00000010  // Endpoint 4 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_3    0x00000008  // Endpoint 3 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_2    0x00000004  // Endpoint 2 Host OUT Interrupt
#define USB_INTEP_HOST_OUT_1    0x00000002  // Endpoint 1 Host OUT Interrupt
#define USB_INTEP_DEV_IN        0x0000FFFE  // Device IN Interrupts
#define USB_INTEP_DEV_IN_15     0x00008000  // Endpoint 15 Device IN Interrupt
#define USB_INTEP_DEV_IN_14     0x00004000  // Endpoint 14 Device IN Interrupt
#define USB_INTEP_DEV_IN_13     0x00002000  // Endpoint 13 Device IN Interrupt
```

```
#define USB_INTEP_DEV_IN_12     0x00001000  // Endpoint 12 Device IN Interrupt
#define USB_INTEP_DEV_IN_11     0x00000800  // Endpoint 11 Device IN Interrupt
#define USB_INTEP_DEV_IN_10     0x00000400  // Endpoint 10 Device IN Interrupt
#define USB_INTEP_DEV_IN_9      0x00000200  // Endpoint 9 Device IN Interrupt
#define USB_INTEP_DEV_IN_8      0x00000100  // Endpoint 8 Device IN Interrupt
#define USB_INTEP_DEV_IN_7      0x00000080  // Endpoint 7 Device IN Interrupt
#define USB_INTEP_DEV_IN_6      0x00000040  // Endpoint 6 Device IN Interrupt
#define USB_INTEP_DEV_IN_5      0x00000020  // Endpoint 5 Device IN Interrupt
#define USB_INTEP_DEV_IN_4      0x00000010  // Endpoint 4 Device IN Interrupt
#define USB_INTEP_DEV_IN_3      0x00000008  // Endpoint 3 Device IN Interrupt
#define USB_INTEP_DEV_IN_2      0x00000004  // Endpoint 2 Device IN Interrupt
#define USB_INTEP_DEV_IN_1      0x00000002  // Endpoint 1 Device IN Interrupt
#define USB_INTEP_0             0x00000001  // Endpoint 0 Interrupt
```

void USBIntDisableEndpoint(unsigned long ulBase, unsigned long ulIntFlags);

作用：禁止 USB 端点中断。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulIntFlags，中断标志。

返回：无。

unsigned long USBIntStatusEndpoint(unsigned long ulBase);

作用：获取 USB 端点中断标志。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：中断标志，与 USBIntEnableEndpoint 的 ulIntFlags 标志相同。

void USBIntEnable(unsigned long ulBase, unsigned long ulIntFlags);

作用：使能 USB 中断，包括端点中断。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulIntFlags，中断标志。

返回：无。

ulIntFlags 可选参数：

```
#define USB_INT_ALL             0xFF030E0F  // All Interrupt sources
#define USB_INT_STATUS          0xFF000000  // Status Interrupts
#define USB_INT_VBUS_ERR        0x80000000  // VBUS Error
#define USB_INT_SESSION_START   0x40000000  // Session Start Detected
#define USB_INT_SESSION_END     0x20000000  // Session End Detected
#define USB_INT_DISCONNECT      0x20000000  // Disconnect Detected
#define USB_INT_CONNECT         0x10000000  // Device Connect Detected
#define USB_INT_SOF             0x08000000  // Start of Frame Detected
#define USB_INT_BABBLE          0x04000000  // Babble signaled
#define USB_INT_RESET           0x04000000  // Reset signaled
#define USB_INT_RESUME          0x02000000  // Resume detected
#define USB_INT_SUSPEND         0x01000000  // Suspend detected
#define USB_INT_MODE_DETECT     0x00020000  // Mode value valid
#define USB_INT_POWER_FAULT     0x00010000  // Power Fault detected
```

```
#define USB_INT_HOST_IN         0x00000E00  // Host IN Interrupts
#define USB_INT_DEV_OUT         0x00000E00  // Device OUT Interrupts
#define USB_INT_HOST_IN_EP3     0x00000800  // Endpoint 3 Host IN Interrupt
#define USB_INT_HOST_IN_EP2     0x00000400  // Endpoint 2 Host IN Interrupt
#define USB_INT_HOST_IN_EP1     0x00000200  // Endpoint 1 Host IN Interrupt
#define USB_INT_DEV_OUT_EP3     0x00000800  // Endpoint 3 Device OUT Interrupt
#define USB_INT_DEV_OUT_EP2     0x00000400  // Endpoint 2 Device OUT Interrupt
#define USB_INT_DEV_OUT_EP1     0x00000200  // Endpoint 1 Device OUT Interrupt
#define USB_INT_HOST_OUT        0x0000000E  // Host OUT Interrupts
#define USB_INT_DEV_IN          0x0000000E  // Device IN Interrupts
#define USB_INT_HOST_OUT_EP3    0x00000008  // Endpoint 3 HOST_OUT Interrupt
#define USB_INT_HOST_OUT_EP2    0x00000004  // Endpoint 2 HOST_OUT Interrupt
#define USB_INT_HOST_OUT_EP1    0x00000002  // Endpoint 1 HOST_OUT Interrupt
#define USB_INT_DEV_IN_EP3      0x00000008  // Endpoint 3 DEV_IN Interrupt
#define USB_INT_DEV_IN_EP2      0x00000004  // Endpoint 2 DEV_IN Interrupt
#define USB_INT_DEV_IN_EP1      0x00000002  // Endpoint 1 DEV_IN Interrupt
#define USB_INT_EP0             0x00000001  // Endpoint 0 Interrupt
```

void USBIntDisable(unsigned long ulBase, unsigned long ulIntFlags);

作用：禁止 USB 通用中断，包括端点中断。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulIntFlags，中断标志。

返回：无。

unsigned long USBIntStatus(unsigned long ulBase);

作用：获取中断标志。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：返回中断标志，与 USBIntEnable 的 ulIntFlags 标志相同。

这些函数中 USBIntEnableControl、 USBIntDisableControl、USBIntStatusControl 控制除端点外的 USB 通用中断；USBIntEnableEndpoint、USBIntDisableEndpoint、USBIntStatusEndpoint 控制 USB 端点中断,端点数量可达 16 个,包括端点 0;USBIntEnable、USBIntDisable、USBIntStatus 控制 USB 中断，包括通用中断和端点中断，但是端点中断只可以控制 4 个，端点 0 到端点 3。

unsigned long USBFrameNumberGet(unsigned long ulBase);

作用：获取当前帧编号。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：返回当前帧编号。

void USBOTGSessionRequest(unsigned long ulBase, tBoolean bStart);

作用：OTG 启动会话。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。bStart，会话启动还是停止。

返回：无。

unsigned long USBModeGet(unsigned long ulBase);

作用：获取 USB 工作模式。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：工作模式。

返回值为：

```
#define USB_DUAL_MODE_HOST      0x00000001  // Dual mode controller is in Host
#define USB_DUAL_MODE_DEVICE    0x00000081  // Dual mode controller is in Device
#define USB_DUAL_MODE_NONE      0x00000080  // Dual mode controller mode not set
#define USB_OTG_MODE_ASIDE_HOST 0x0000001d  // OTG A side
#define USB_OTG_MODE_ASIDE_NPWR 0x00000001  // OTG A side
#define USB_OTG_MODE_ASIDE_SESS 0x00000009  // OTG A side of cable Session Valid.
#define USB_OTG_MODE_ASIDE_AVAL 0x00000011  // OTG A side of the cable A valid.
#define USB_OTG_MODE_ASIDE_DEV  0x00000019  // OTG A side of the cable.
#define USB_OTG_MODE_BSIDE_HOST 0x0000009d  // OTG B side of the cable.
#define USB_OTG_MODE_BSIDE_DEV  0x00000099  // OTG B side of the cable.
#define USB_OTG_MODE_BSIDE_NPWR 0x00000081  // OTG B side of the cable.
#define USB_OTG_MODE_NONE       0x00000080  // OTG controller mode is not set.
```

本节主要介绍 USB 基本操作 API，包含 USBEndpoint 组 API、USBFIFO 组 API、其它公用 API。通过以上学习，可以对 USB 底层操作做一个大概了解，为以后协议理解打好基础。

3.4 设备库函数

设备库函数 API，只包含设备能够使用的 API 函数。要开发 USB 设备还需要与前面章节介绍的 API 结合，才能完成 USB 设备功能。

```
void USBDevAddrSet(unsigned long ulBase, unsigned long ulAddress);
unsigned long USBDevAddrGet(unsigned long ulBase);
void USBDevConnect(unsigned long ulBase);
void USBDevDisconnect(unsigned long ulBase);
void USBDevEndpointConfigSet(unsigned long ulBase,
                             unsigned long ulEndpoint,
                             unsigned long ulMaxPacketSize,
                             unsigned long ulFlags);
void USBDevEndpointConfigGet(unsigned long ulBase,
                             unsigned long ulEndpoint,
                             unsigned long *pulMaxPacketSize,
                             unsigned long *pulFlags);
void USBDevEndpointDataAck(unsigned long ulBase,
                           unsigned long ulEndpoint,
                           tBoolean bIsLastPacket);
void USBDevEndpointStall(unsigned long ulBase, unsigned long ulEndpoint,
                         unsigned long ulFlags);
void USBDevEndpointStallClear(unsigned long ulBase,
                              unsigned long ulEndpoint,
                              unsigned long ulFlags);
void USBDevEndpointStatusClear(unsigned long ulBase,
```

```
                                        unsigned long ulEndpoint,
                                        unsigned long ulFlags);
```

void USBDevAddrSet(unsigned long ulBase, unsigned long ulAddress);

作用：设置设备地址。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulAddress，主机 USBREQ_SET_ADDRESS 请求命令时 tUSBRequest.wValue 值，用于配置设备地址。

返回：无。

unsigned long USBDevAddrGet(unsigned long ulBase);

作用：获取设备地址。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：设备地址。

void USBDevConnect(unsigned long ulBase);

作用：软件连接到 USB 主机。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

void USBDevDisconnect(unsigned long ulBase);

作用：软件断开设备与主机的连接。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

void USBDevEndpointConfigSet(unsigned long ulBase, unsigned long ulEndpoint,
                             unsigned long ulMaxPacketSize, unsigned long ulFlags);

作用：在设备模式下，配置端点的最大数据包大小，及其它配置。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n(n=0..15)。ulMaxPacketSize，最大数据包大小。ulFlags，关于端点的其它配置。

返回：无。

ulFlags 可选参数：

```
#define USB_EP_AUTO_SET         0x00000001  // Auto set feature enabled
#define USB_EP_AUTO_REQUEST     0x00000002  // Auto request feature enabled
#define USB_EP_AUTO_CLEAR       0x00000004  // Auto clear feature enabled
#define USB_EP_DMA_MODE_0       0x00000008  // Enable DMA access using mode 0
#define USB_EP_DMA_MODE_1       0x00000010  // Enable DMA access using mode 1
#define USB_EP_MODE_ISOC        0x00000000  // Isochronous endpoint
#define USB_EP_MODE_BULK        0x00000100  // Bulk endpoint
#define USB_EP_MODE_INT         0x00000200  // Interrupt endpoint
#define USB_EP_MODE_CTRL        0x00000300  // Control endpoint
#define USB_EP_MODE_MASK        0x00000300  // Mode Mask
#define USB_EP_SPEED_LOW        0x00000000  // Low Speed
#define USB_EP_SPEED_FULL       0x00001000  // Full Speed
#define USB_EP_HOST_IN          0x00000000  // Host IN endpoint
#define USB_EP_HOST_OUT         0x00002000  // Host OUT endpoint
#define USB_EP_DEV_IN           0x00002000  // Device IN endpoint
```

```
#define USB_EP_DEV_OUT          0x00000000  // Device OUT endpoint
```

```
void USBDevEndpointConfigGet(unsigned long ulBase, unsigned long ulEndpoint,
                             unsigned long *pulMaxPacketSize, unsigned long *pulFlags);
```

作用：在设备模式下，获取端点配置。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n(n=0..15)。ulMaxPacketSize，返回最大数据包大小指针。ulFlags，返回端点配置指针。

返回：无。

```
void USBDevEndpointDataAck(unsigned long ulBase, unsigned long ulEndpoint,
                                                  tBoolean bIsLastPacket);
```

作用：在设备模式下，收到数据响应。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。bIsLastPacket，数据是否是最后一个包。

返回：无。

```
void USBDevEndpointStall(unsigned long ulBase, unsigned long ulEndpoint,
                                                unsigned long ulFlags);
```

作用：在设备模式下，停止端点。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，指定是 IN 端点还是 OUT 端点。

返回：无。

```
void USBDevEndpointStallClear(unsigned long ulBase, unsigned long ulEndpoint,
                                                    unsigned long ulFlags);
```

作用：在设备模式下，清除端点的停止条件。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，指定是 IN 端点还是 OUT 端点。

返回：无。

```
void USBDevEndpointStatusClear(unsigned long ulBase,
                                        unsigned long ulEndpoint,
                                        unsigned long ulFlags);
```

作用：在设备模式下，清除端点的状态。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，与 USBEndpointStatus 返回值相同，用于指定清除端点的状态。

返回：无。

以上介绍的 API 函数只能在设备模式下调用，并且在设备模式下使用频繁。如果不正确使用这些 API 函数，可能枚举不成功；枚举成功，数据传输也可能有问题。所以开发 USB 设备，掌握这些 API 函数是必要的。

3.5 主机库函数

主机库函数 API，只包含主机能够使用的 API 函数。要开发 USB 主机还需要与前面章节介绍的 API 结合，才能完成 USB 主机功能。

```
void USBHostAddrSet(unsigned long ulBase, unsigned long ulEndpoint,
                            unsigned long ulAddr, unsigned long ulFlags);
```

```
unsigned long USBHostAddrGet(unsigned long ulBase,
                                    unsigned long ulEndpoint,
                                    unsigned long ulFlags);
void USBHostEndpointConfig(unsigned long ulBase,
                                   unsigned long ulEndpoint,
                                   unsigned long ulMaxPacketSize,
                                   unsigned long ulNAKPollInterval,
                                   unsigned long ulTargetEndpoint,
                                   unsigned long ulFlags);
void USBHostEndpointDataAck(unsigned long ulBase,
                                    unsigned long ulEndpoint);
void USBHostEndpointDataToggle(unsigned long ulBase,
                                      unsigned long ulEndpoint,
                                      tBoolean bDataToggle,
                                      unsigned long ulFlags);
void USBHostEndpointStatusClear(unsigned long ulBase,
                                       unsigned long ulEndpoint,
                                       unsigned long ulFlags);
void USBHostHubAddrSet(unsigned long ulBase, unsigned long ulEndpoint,
                               unsigned long ulAddr, unsigned long ulFlags);
unsigned long USBHostHubAddrGet(unsigned long ulBase,
                                       unsigned long ulEndpoint,
                                       unsigned long ulFlags);
void USBHostPwrEnable(unsigned long ulBase);
void USBHostPwrDisable(unsigned long ulBase);
void USBHostPwrConfig(unsigned long ulBase, unsigned long ulFlags);
void USBHostPwrFaultEnable(unsigned long ulBase);
void USBHostPwrFaultDisable(unsigned long ulBase);
void USBHostRequestIN(unsigned long ulBase, unsigned long ulEndpoint);
void USBHostRequestStatus(unsigned long ulBase);
void USBHostReset(unsigned long ulBase, tBoolean bStart);
void USBHostResume(unsigned long ulBase, tBoolean bStart);
void USBHostSuspend(unsigned long ulBase);
void USBHostMode(unsigned long ulBase);
unsigned long USBHostSpeedGet(unsigned long ulBase);
```

```
void USBHostAddrSet(unsigned long ulBase, unsigned long ulEndpoint,
                    unsigned long ulAddr, unsigned long ulFlags);
```

作用：主机端点与设备通信时，设置端点访问的设备地址。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，指定是 IN 端点还是 OUT 端点。ulAddr，指定设备地址。

返回：无。

```
unsigned long USBHostAddrGet(unsigned long ulBase,
```

unsigned long ulEndpoint,
unsigned long ulFlags);

作用：主机端点与设备通信时，获取端点访问的设备地址。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，指定是 IN 端点还是 OUT 端点。

返回：设备地址。

void USBHostEndpointConfig(unsigned long ulBase,
unsigned long ulEndpoint,
unsigned long ulMaxPacketSize,
unsigned long ulNAKPollInterval,
unsigned long ulTargetEndpoint,
unsigned long ulFlags);

作用：主机端点配置。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulMaxPacketSize ，ulFlags，指定是 IN 端点还是 OUT 端点。ulMaxPayload ，端点的最大数据包大小。ulNAKPollInterval ，NAK 超时限制或查询间隔，这取决于端点的类型。ulTargetEndpoint ，目标端点。

返回：无。

void USBHostEndpointDataAck(unsigned long ulBase, unsigned long ulEndpoint);

作用：主机端点响应。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。

返回：无。

void USBHostEndpointDataToggle(unsigned long ulBase, unsigned long ulEndpoint,
tBoolean bDataToggle, unsigned long ulFlags);

作用：主机端点数据转换。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。bDataToggle，是否进行数据转换。ulFlags，指定是 IN 端点还是 OUT 端点。

返回：无。

void USBHostEndpointStatusClear(unsigned long ulBase,
unsigned long ulEndpoint, unsigned long ulFlags);

作用：清除主机端点状态。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，指定是 IN 端点还是 OUT 端点。

返回：无。

void USBHostHubAddrSet(unsigned long ulBase, unsigned long ulEndpoint,
unsigned long ulAddr, unsigned long ulFlags);

作用：设置集线器地址。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，指定是 IN 端点还是 OUT 端点。ulAddr，集线器地址。

返回：无。

```
unsigned long USBHostHubAddrGet(unsigned long ulBase,
                                 unsigned long ulEndpoint,
                                 unsigned long ulFlags);
```

作用：获取集线器地址。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。ulFlags，指定是 IN 端点还是 OUT 端点。

返回：集线器地址。

```
void USBHostPwrEnable(unsigned long ulBase);
```

作用：主机使能外部电源。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

```
void USBHostPwrDisable(unsigned long ulBase);
```

作用：主机禁止外部电源。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

void USBHostPwrConfig(unsigned long ulBase, unsigned long ulFlags);

作用：外部电源配置。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulFlags，电源配置。

返回：无。

void USBHostPwrFaultEnable(unsigned long ulBase);

作用：电源异常使能。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

void USBHostPwrFaultDisable(unsigned long ulBase);

作用：电源异常禁止。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

void USBHostRequestIN(unsigned long ulBase, unsigned long ulEndpoint);

作用：发送请求。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。ulEndpoint，指定操作的端点号，USB_EP_n（n=0..15）。

返回：无。

void USBHostRequestStatus(unsigned long ulBase);

作用：在端点 0 上发送设备状态标准请求。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

void USBHostReset(unsigned long ulBase, tBoolean bStart);

作用：总线复位。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。bStart，复位开始还是停止。复位开始后 20mS 后软件复位停止。

返回：无。

void USBHostResume(unsigned long ulBase, tBoolean bStart);

作用：总线唤醒。这个函数在设备模式下可以调用。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。bStart，唤醒开始还是停止。主机模式下，唤醒开始后 20mS 后软件唤醒停止。在设备模式下，大于 10ms 小于 15ms。

返回：无

void USBHostSuspend(unsigned long ulBase);

作用：总线挂起。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

void USBHostMode(unsigned long ulBase);

作用：设置 USB 处理器工作于主机模式。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：无。

unsigned long USBHostSpeedGet(unsigned long ulBase);

作用：获取与主机连接的设备速度。

参数：ulBase，指定 USB 模块，在 Stellaris USB 处理器中只有一个 USB 模块，所以参数为 USB0_BASE。

返回：设备速度。

以上介绍的 API 函数（除 USBHostResume 外）只能在主机模式下调用，并且在主机模式下使用频繁。所有主机都会使用到本节 API 函数，这些函数在 USB 主机开发中占有重要地位。

例如，主机对设备进行枚举过程控制，简单请求过程。

```
static void USBHCDEnumHandler(void)
{
    unsigned long ulEPStatus;
    unsigned long ulDataSize;
     //获取端点 0 的状态
    ulEPStatus = USBEndpointStatus(USB0_BASE, USB_EP_0);
    //判断端点 0 状态
    if(ulEPStatus == USB_HOST_EP0_ERROR)
    {
        //如果端点 0 出现问题，清除端点 0 状态。
        USBHostEndpointStatusClear(USB0_BASE, USB_EP_0, USB_HOST_EP0_ERROR);
        //清空端点 0 的 FIFO 数据
        USBFIFOFlush(USB0_BASE, USB_EP_0, 0);
        g_sUSBHEP0State.eState = EP0_STATE_ERROR;
        return;
    }
    //根据端点 0 的状态进行处理
    switch(g_sUSBHEP0State.eState)
    {
        case EP0_STATE_STATUS:
        {
            //处理接收到的数据
            if(ulEPStatus & (USB_HOST_EP0_RXPKTRDY | USB_HOST_EP0_STATUS))
            {
                //清除接收状态
                USBHostEndpointStatusClear(USB0_BASE, USB_EP_0,
                                           (USB_HOST_EP0_RXPKTRDY |
                                            USB_HOST_EP0_STATUS));
            }
            g_sUSBHEP0State.eState = EP0_STATE_IDLE;
```

```
    break;
}
case EP0_STATE_STATUS_IN:
{
    //主机发送状态请求
    USBHostRequestStatus(USB0_BASE);
    g_sUSBHEP0State.eState =  EP0_STATE_STATUS;
    break;
}
case EP0_STATE_IDLE:
{
    break;
}
case EP0_STATE_SETUP_OUT:
{
    //通过端点 0 发送数据
    USBHCDEP0StateTx();
    break;
}
case EP0_STATE_SETUP_IN:
{
    //主机发出请求
    USBHostRequestIN(USB0_BASE, USB_EP_0);
    g_sUSBHEP0State.eState =  EP0_STATE_RX;
    break;
}
case EP0_STATE_RX:
{
    if(ulEPStatus & USB_HOST_EP0_RX_STALL)
    {
        g_sUSBHEP0State.eState = EP0_STATE_IDLE;
        USBHostEndpointStatusClear(USB0_BASE, USB_EP_0, ulEPStatus);
        break;
    }
    if(g_sUSBHEP0State.ulBytesRemaining > MAX_PACKET_SIZE_EP0)
    {
        ulDataSize = MAX_PACKET_SIZE_EP0;
    }
    else
    {
        ulDataSize = g_sUSBHEP0State.ulBytesRemaining;
    }
    if(ulDataSize != 0)
    {
```

```
                //从端点 0 中获取数据
                USBEndpointDataGet(USB0_BASE, USB_EP_0, g_sUSBHEP0State.pData,
                                   &ulDataSize);
            }
            g_sUSBHEP0State.pData += ulDataSize;
            g_sUSBHEP0State.ulBytesRemaining -= ulDataSize;
            //主机响应
            USBDevEndpointDataAck(USB0_BASE, USB_EP_0, false);
            if((ulDataSize < g_sUSBHEP0State.ulMaxPacketSize) ||
               (g_sUSBHEP0State.ulBytesRemaining == 0))
            {
                g_sUSBHEP0State.eState =  EP0_STATE_STATUS;
               //数据接收完，发送 NULL 包
                USBEndpointDataSend(USB0_BASE, USB_EP_0, USB_TRANS_STATUS);
            }
            else
            {
                //主机请求
                USBHostRequestIN(USB0_BASE, USB_EP_0);
            }
            break;
        }
        case EP0_STATE_STALL:
        {
            g_sUSBHEP0State.eState = EP0_STATE_IDLE;
            break;
        }
        default:
        {
            ASSERT(0);
            break;
        }
    }
}
```

端点 0 发送数据：

```
static void USBHCDEP0StateTx(void)
{
    unsigned long ulNumBytes;
    unsigned char *pData;
    g_sUSBHEP0State.eState = EP0_STATE_SETUP_OUT;
     //设置发送数据字节数
     ulNumBytes = g_sUSBHEP0State.ulBytesRemaining;
    //判断大于端点 0 的最大包长度 64。
    if(ulNumBytes > 64)
```

```
        {
            ulNumBytes = 64;
        }
        // 指向待发数据
        pData = (unsigned char *)g_sUSBHEP0State.pData;
        g_sUSBHEP0State.ulBytesRemaining -= ulNumBytes;
        g_sUSBHEP0State.pData += ulNumBytes;
        // 放入 FIO 中
        USBEndpointDataPut(USB0_BASE, USB_EP_0, pData, ulNumBytes);
        //判断是否是最后一个包，并存入包结束标志
        if(ulNumBytes == 64)
        {
            USBEndpointDataSend(USB0_BASE, USB_EP_0, USB_TRANS_OUT);
        }
        else
        {
            USBEndpointDataSend(USB0_BASE, USB_EP_0, USB_TRANS_OUT);
            g_sUSBHEP0State.eState = EP0_STATE_STATUS_IN;
        }
    }
```

本章介绍了 USB 处理器底层 API 操作函数的使用，有助于加深读者对 USB 设备和主机控制。对于具体的 USB 开发，使用现有的 API 函数及可完成。由于 USB 协议的复杂性，从第四章开始学习使用 Luminary Micro 公司提供的 Stellaris Cortex-m3 USB 处理器的 USB 应用性库函数进行实际 USB 工程开发。