USB Full Speed Device Datasheet USBFS V 2.10

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY7C64215, CY7C643xx, CY8C20396/A, CY8C20496/A/L, CY8C20646/A/AS/L, CY8C20666/A/AS/L, CY8C24794, CY8C24894-24LTXI, CY8C24994, CY8CLED04, CYONS2000, CYONS2100, CYONS2110, CYONS2010, CYONSFN2162, CYONSTB2010, CYONS2S8OCD, CYRF89235, CY8CTMA120, CY8CTMG120, CY8CTST120, CY8CTST200/A, CY8CTMG200/A, CY8CTMG201/A, CY8C24493, CY7C69xxx						
	–	–	–	1911	46	2

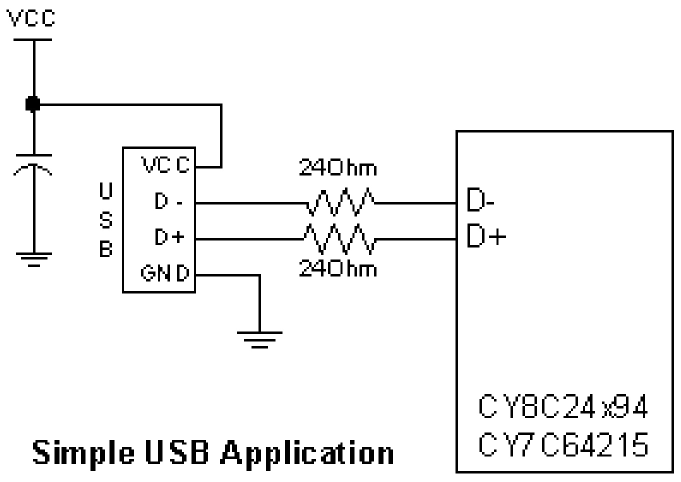
Note

1. Expect an expansion of Flash and RAM when adding additional interfaces, HID classes, and other USBFS extensions.
2. SysClk*2 is needed at all times for proper USB timing. In Global Resources, set **SysClk*2 Disable** to **No** for proper USB operation.

Features and Overview

- USB Full Speed device interface driver
- Support for interrupt and control transfer types
- Setup wizard for easy and accurate descriptor generation
- Runtime support for descriptor set selection
- Optional USB string descriptors
- Optional USB HID class support

Figure 1. USBFS Device Block Diagram



Functional Description

The USBFS User Module gives a USB full-speed-compliant (Chapter 9) device framework. This user module gives a low level driver for the control endpoint that decodes and dispatches requests from the USB host. In addition, this user module gives a USBFS Setup Wizard to enable easy descriptor construction.

When you place the USBFS User Module, you can choose to construct an HID based device or a generic USB Device. After you add an instance of a USBFS User Module, switch between an HID device and a generic device by deleting and then adding a new instance of the USBFS User Module.

USB Compliance

USB drivers may present various bus conditions to the device, including Bus Resets, and different timing requirements. Not all of these can be correctly illustrated in the given examples. When designing applications, ensure that your applications conform to the USB spec.

USB Compliance for Self Powered Devices

In the *USB Compliance Checklist* there is a question that reads, “Is the device’s pull up active only when V_{BUS} is high?”

The question lists Section 7.1.5 in the *Universal Serial Bus Specification Revision 2.0* as a reference. This section reads, in part, “The voltage source on the pull up resistor must be derived from or controlled by the power supplied on the USB cable such that when V_{BUS} is removed, the pull up resistor does not supply current on the data line to which it is attached.”

If you are creating a self-powered device, connect a GPIO pin to V_{BUS} through a resistive network and write firmware to monitor the status of the GPIO. Application Note AN15813, *Monitoring the EZ-USB FX2LP VBUS*, explains the necessary hardware and software components. You can use the USBFS_Start() and USBFS_Stop() API routines to control the D+ and D- pin pull ups. The pull up resistor does not supply power to the data line until you call USBFS_Start(). USBFS_Stop() disconnects the pull up resistor from the data pin. Section 9.1.1.2 in the *Universal Serial Bus Specification Revision 2.0* says, “Devices report their power source capability through the configuration descriptor. The current power source is reported as part of a device’s status. Devices may change their power source at any time, for example, from self- to bus-powered.“The device responds to GET_STATUS requests based on the status set with the USBFS_SetPowerStatus() function. To set the correct status, USBFS_SetPowerStatus() must be called at least once if your device is configured as self-powered. You should also call the USBFS_SetPowerStatus() function any time your device changes status.

Timing

The USBFS User Module supports USB 2.0 full speed operation on the CY7C64215, CY7C643xx, CY8C20396/A, CY8C20496/A/L, CY8C20646/A/AS/L, CY8C20666/A/AS/L, CY8C24794, CY8C24894-24LTXI, CY8C24994, CY8CLED04, CYONS2000, CYONS2100, CYONS2110, CYONS2010, CYONSFN2162, CYONSTB2010, CYONS2S8OCD, CYRF89235, CY8CTMA120, CY8CTMG120, CY8CTST120, CY8CTST200/A, CY8CTMG200/A, CY8CTMG201/Adevices.

Parameters and Resources

The USBFS User Module does not use the PSoC Designer User Module Properties Window for personalization. Instead, it uses a form driven USBFS Setup Wizard to define the USB descriptors for the application. From the descriptors, the wizard personalizes the user module.

The user module is driven by information generated by the USBFS Setup Wizard. This wizard facilitates the construction of the USB descriptors and integrates the information generated into the driver firmware used for device enumeration.

The USBFS User Module does not function without first running the wizard, selecting the appropriate attributes, and generating code. The project may have build errors if the wizard is not run. Also, if HID Device Class support is used, build errors may occur if the following actions in the wizard are not done:

1. A HID Report must be added in the wizard in the HID Report Descriptor Root section.
2. A HID Item must be added to the HID Report.
3. A specific HID Report must be selected for the HID Class Descriptor in the USB User Module Descriptor Root section of the wizard.

Be advised of the location of the USB ISRs in the interrupt vector table. Servicing USB interrupts, especially EP0, can be delayed due to the higher priority interrupts such as VC3, GPIO, DBB Blocks, and DCB Blocks. Depending on the application, this could affect overall functionality. Refer to the boot.asm file for details regarding the interrupt priority.

Application Programming Interface

The Application Programming Interface (API) routines in this section allow programmatic control of the USBFS User Module. The following sections describe descriptor generation and integration. The sections also list the basic and device specific API functions. As a developer you need a basic understanding of the USB protocol and familiarity with the USB 2.0 specification, especially Chapter 9, USB Device Framework.

The USBFS User Module supports control, interrupt, bulk, and isochronous transfers. Some or a group of functions, such as LoadInEP and EnableOutEP, are designed for use with bulk and interrupt endpoints. Other functions, such as USBFS_LoadINISOCEP, are designed for use with isochronous endpoints. Refer to the Technical Reference Manual (TRM) for more information on how to do these transfer types.

Note The API routines for the USB User Modules are not reentrant. Because they depend on internal global variables in RAM, executing these routines from an interrupt is not supported by the API support supplied with this user module. If this is a requirement for a design, contact the local Cypress Field Application Engineer.

.

Table 1. Basic USBFS Device API

Function	Description
void USBFS_Start(BYTE bDevice, BYTE bMode)	Activate the user module for use with the device and specific voltage mode.
void USBFS_Stop(void)	Disable user module.
BYTE USBFS_bCheckActivity(void)	Checks and clears the USB bus activity flag. Returns 1 if the USB was active since the last check, otherwise returns 0.
BYTE USBFS_bGetConfiguration(void)	Returns the currently assigned configuration. Returns 0 if the device is not configured.
BYTE USBFS_bGetEPState(BYTE bEPNumber)	Returns the current state of the specified USBFS endpoint. 2 = NO_EVENT_ALLOWED 1 = EVENT PENDING 0 = NO_EVENT_PENDING
BYTE USBFS_bGetEPAckState(BYTE bEPNumber)	Identifies whether ACK was set by returning a non-zero value.
BYTE USBFS_wGetEPCount(BYTE bEPNumber)	Returns the current byte count from the specified USBFS endpoint.
void USBFS_LoadInEP(BYTE bEPNumber, BYTE *pData, WORD wLength, BYTE bToggle) void USB_LoadInISOCEP(BYTE bEPNumber, BYTE *pData, WORD wLength, BYTE bToggle)	Loads and enables the specified USBFS endpoint for an IN transfer.
BYTE USBFS_bReadOutEP(BYTE bEPNumber, BYTE *pData, WORD wLength)	Reads the specified number of bytes from the Endpoint RAM and places it in the RAM array pointed to by pSrc. The function returns the number of bytes sent by the host.

Function	Description
void USB_EnableOutEP(BYTE bEPNumber) void USB_EnableOutISOCEP(BYTE bEPNumber)	Enables the specified USB endpoint to accept OUT transfers
void USBFS_DisableOutEP(BYTE bEPNumber)	Disables the specified USB endpoint to NAK OUT transfers
void USBFS_SetPowerStatus(BYTE bPowerStatus)	Sets the device to self powered or bus powered
USBFS_Force(BYTE bState)	Forces a J, K, or SE0 State on the USB D+/D- pins. Normally used for remote wakeup. bState Parameters are: USBFS_FORCE_J 0xA0 USBFS_FORCE_K 0x80 USBFS_FORCE_SE0 0xC0 USBFS_FORCE_NONE 0x00 **Note**: When using this API Function and GPIO pins from Port 1 (P1.2-P1.7), the application uses the Port_1_Data_SHADE shadow register to ensure consistent data handling. From assembly language, access the Port_1_Data_SHADE RAM location directly. From C language, include an extern reference: `extern BYTE Port_1_Data_SHADE;`

Table 2. Human Interface Device (HID) Class Support API

Function	Description
BYTE USBFS_UpdateHIDTimer(BYTE bInterface)	Updates the HID Report timer for the specified interface and returns 1 if the timer expired and 0 if not. If the timer expired, it reloads the timer.
BYTE USBFS_bGetProtocol(BYTE bInterface)	Returns the protocol for the specified interface

USBFS_Start

Description:

Performs all required initialization for USBFS User Module.

C Prototype:

```
void USBFS_Start(BYTE bDevice, BYTE bMode)
```

Assembly:

```
mov   A, 0                     ; Select the device
mov   X, USB_5V_OPERATION      ; Select the Voltage level
lcall  USBFS_Start              ; lcall the Start Function
```

Parameters:

Register A: contains the device number from the desired device descriptor set entered with the USBFS Setup Wizard.

Register X: contains the operating voltage at which the chip runs. This determines whether the voltage regulator is enabled for 5V operation or if pass through mode is used for 3.3V operation. Symbolic names are given in C and assembly, and their associated values are given in the following table.

Mask	Value	Description
USB_3V_OPERATION	0x02	Disable voltage regulator and pass-through vcc for pull up
USB_5V_OPERATION	0x03	Enable voltage regulator and use regulator for pull up

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is responsibility of the calling function to preserve the values across calls to fastcall16 functions. Currently only the IDX_PP and the CUR_PP page pointer registers are modified.

USBFS_Stop

Description:

Performs all necessary shutdown task required for the USBFS User Module.

C Prototype:

```
void  USBFS_Stop(void)
```

Assembly:

```
lcall  USBFS_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions. Currently only the CUR_PP page pointer register is modified.

USBFS_bCheckActivity

Description:

Checks for USBFS Bus Activity.

C Prototype:

```
BYTE  USBFS_bCheckActivity(void)
```

Assembly:

```
lcall  USBFS_bCheckActivity
```

Parameters:

None

Return Value:

Returns 1 in A if the USB was active since the last check, otherwise returns 0.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions.

USBFS_bGetConfiguration

Description:

Gets the current configuration of the USB device.

C Prototype:

```
BYTE  USBFS_bGetConfiguration(void)
```

Assembly:

```
lcall  USBFS_bGetConfiguration
```

Parameters:

None

Return Value:

Returns the currently assigned configuration in A. Returns 0 if the device is not configured.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions. Currently only the CUR_PP page pointer register is modified.

USBFS_bGetEPState

Description:

Gets the Endpoint state for the specified endpoint. The Endpoint state describes, from the perspective of the foreground application, the endpoint status. The endpoint has one of three states, two of the states mean different things for IN and OUT endpoints. The following table outlines the possible states and their meaning for IN and OUT endpoints.

C Prototype:

```
BYTE  USBFS_bGetEPState(BYTE bEPNumber)
```

Assembly:

```
MOV   A, 1                           ; Select endpoint 1
lcall  USBFS_bGetEPState
```

Parameters:

Register A contains the endpoint number.

Return Value:

Returns the current state of the specified USBFS endpoint. Symbolic names given in C and assembly, and their associated values are given in the following table. Use these constants whenever the user writes code to change the state of the Endpoints such as ISR code to handle data sent or received.

State	Value	Description
NO_EVENT_PENDING	0x00	Indicates that the endpoint is awaiting SIE action
EVENT_PENDING	0x01	Indicates that the endpoint is awaiting CPU action
NO_EVENT_ALLOWED	0x02	Indicates that the endpoint is locked from access
IN_BUFFER_FULL	0x00	The IN endpoint is loaded and the mode is set to ACK IN
IN_BUFFER_EMPTY	0x01	An IN transaction occurred and more data can be loaded
OUT_BUFFER_EMPTY	0x00	The OUT endpoint is set to ACK OUT and is waiting for data
OUT_BUFFER_FULL	0x01	An OUT transaction has occurred and data can be read

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions. Currently only the IDX_PP page pointer register is modified.

USBFS_bGetEPAckState

Description:

Determines whether or not an ACK transaction occurred on this endpoint by reading the ACK bit in the control register of the endpoint. This function does not clear the ACK bit.

C Prototype:

```
BYTE  USBFS_bGetEPAckState(BYTE bEPNumber)
```

Assembly:

```
MOV   A, 1                             ; Select endpoint 1
lcall  USBFS_bGetEPAckState
```

Parameters:

Register A contains the endpoint number.

Return Value:

If an ACKed transaction occurred then this function returns a non-zero value. Otherwise a zero is returned.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions.

USBFS_wGetEPCount

Description:

This functions returns the value of the endpoint count register. The Serial Interface Engine (SIE) includes two bytes of checksum data in the count. This function subtracts two from the count before returning the value. Call this function only for OUT endpoints after a call to USB_GetEPState returns EVENT_PENDING.

C Prototype:

```
WORD  USBFS_wGetEPCount(BYTE bEPNumber)
```

Assembly:

```
MOV   A, 1                             ; Select endpoint 1
lcall  USBFS_wGetEPCount
```

Parameters:

Register A contains the endpoint number.

Return Value:

Returns the current byte count from the specified USBFS endpoint in A and X.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions.

USBFS_LoadInEP and USBFS_LoadInISOCEP

Description:

Loads and enables the specified USB endpoint for an IN Interrupt or Bulk transfer (.._LoadInEP) and Isochronous transfer (..._LoadInISOCEP).

C Prototype:

```
void  USBFS_LoadInEP(BYTE bEPNumber, BYTE * pData, WORD wLength, BYTE bToggle)
void  USBFS_LoadInISOCEP(BYTE bEPNumber, BYTE * pData, WORD wLength, BYTE bToggle)
```

Assembly:

```
mov A, USBFS_TOGGLE
push A
mov A, 0
push A
mov A, 32
push A
mov A, >pData
push A
mov A, <pData
push A
mov A, 1
```

```
push A
lcall  USBFS_LoadInEP
```

Parameters:

bEPNumber is the Endpoint Number between 1 and 4.

pData is a pointer to a data array from which the Data for the Endpoint space is loaded.

wLength is the number of bytes to transfer from the array and then sent as a result of an IN request. Valid values are between 0 and 256.

bToggle is a flag indicating whether or not the Data Toggle bit is toggled before setting it in the count register. For IN transactions toggle the Data bit after every successful data transmission. This makes certain that the same packet is not repeated or lost. Symbolic names for the flag are given in C and assembly, and their associated values are shown here:.

Mask	Value	Description
USB_NO_TOGGLE	0x00	The Data Toggle does not change
USB_TOGGLE	0x01	The Data bit is toggled before transmission

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions. Currently only the IDX_PP and the CUR_PP page pointer registers are modified.

USBFS_bReadOutEP

Description:

Moves the specified number of bytes from endpoint RAM to data RAM. The number of bytes actually transferred from endpoint RAM to data RAM is the lesser of the actual number of bytes sent by the host and the number of bytes requested by the wCount argument.

C Prototype:

```
BYTE  USB_bReadOutEP(BYTE bEPNumber, BYTE * pData, WORD wLength)
```

Assembly:

```
mov A, 0
push A
mov A, 32
push A
mov A, >pData
push A
mov A, <pData
push A
mov A, 1
push A
lcall  USB_bReadOutEP
```

Parameters:

bEPNumber is the Endpoint Number between 1 and 4

pData is a pointer to a data array to which the Data from the Endpoint space is loaded.

wLength is the number of bytes to transfer from the array and then sent as a result of an IN request. Valid values are between 0 and 256. The function moves less than that if the number of bytes sent by the host are less requested.

Return Value:

Returns the number of bytes sent by the host to the USB device. This could be more or less than the number of bytes requested.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions. Currently only the IDX_PP and the CUR_PP page pointer registers are modified.

USBFS_EnableOutEP and USBFS_EnableOutISOCEP

Description:

Enables the specified endpoint for OUT Bulk or Interrupt transfers (..._EnableOutEP) and Isochronous transfers (..._EnableOutISOCEP). Do not call these functions for IN endpoints.

C Prototype:

```
void  USBFS_EnableOutEP(BYTE bEPNumber)
void  USBFS_EnableOutISOCEP(BYTE bEPNumber)
```

Assembly:

```
MOV   A, 1
lcall  USBFS_EnableOutEP
```

Parameters:

Register A contains the endpoint number.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions. Currently only the IDX_PP page pointer register is modified.

USBFS_DisableOutEP

Description:

Disables the specified USBFS OUT endpoint. Do not call this function for IN endpoints.

C Prototype:

```
void  USBFS_DisableOutEP(BYTE bEPNumber)
```

Assembly:

```
MOV    A, 1                             ; Select endpoint 1
lcall  USBFS_DisableOutEP
```

Parameters:

Register A contains the endpoint number.

Return Value:

None.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions.

USBFS_Force

Description:

Forces a USB J, K, or SE0 state on the D+/D- lines. This function gives the necessary mechanism for a USB device application to perform USB Remote Wakeup functionality. For more information, refer to the USB 2.0 Specification for details on Suspend and Resume functionality.

C Prototype:

```
void  USBFS_Force(BYTE bState)
```

Assembly:

```
mov    A, USB_FORCE_K
lcall  USBFS_Force
```

Parameters:

bState is byte indicating which among four bus states to enable. Symbolic names given in C and assembly, and their associated values are listed here:.

State	Value	Description
USB_FORCE_SE0	0xC0	Force a Single Ended 0 onto the D+/D- lines
USB_FORCE_J	0xA0	Force a J State onto the D+/D- lines
USB_FORCE_K	0x80	Force a K State onto the D+/D- lines
USB_FORCE_NONE	0x00	Return bus to SIE control

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions.

USBFS_UpdateHIDTimer

Description:

Updates the HID Report Idle timer and returns the expiry status. Reloads the timer if it expires.

C Prototype:

```
BYTE  USBFS_UpdateHIDTimer(BYTE bInterface)
```

Assembly:

```
MOV   A, 1                             ; Select interface 1
lcall  USBFS_UpdateHIDTimer
```

Parameters:

Register A contains the interface number.

Return Value:

The state of the HID timer is returned in A. Symbolic names are given in C and assembly, and their associated values are given here:.

State	Value	Description
USB_IDLE_TIMER_EXPIRED	0x01	The timer expired.
USB_IDLE_TIMER_RUNNING	0x02	The timer is running.
USB_IDLE_TIMER_IDEFINITE	0x00	Returned if the report is sent when data or state changes.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions.

USBFS_bGetProtocol

Description:

Returns the hid protocol value for the selected interface.

C Prototype:

```
BYTE  USBFS_bGetProtocol(BYTE bInterface)
```

Assembly:

```
MOV   A, 1                             ; Select interface 1
lcall  USBFS_bGetProtocol
```

Parameters:

bInterface contains the interface number.

Return Value:

Register A contains the protocol value.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions.

USBFS_SetPowerStatus

Description:

Sets the current power status. Set the power status to one for self powered or zero for bus powered. The device will reply to USB GET_STATUS requests based on this value. This allows the device to properly report its status for USB Chapter 9 compliance. Devices may change their power source from self powered to bus powered at any time and report their current power source as part of the device status. You should call this function any time your device changes from self powered to bus powered or vice versa, and set the status appropriately.

C Prototype:

```
void USBFS_SetPowerStatus(BYTE bPowerStaus);
```

Assembly:

```
MOV   A, 1                             ; Select self powered
lcall  USBFS_SetPowerStatus
```

Parameters:

bPowerStatus contains the desired power status, one for self powered or zero for bus powered. Symbolic names are given in C and assembly, and their associated values are given here:

State	Value	Description
USB_DEVICE_STATUS_BUS_POWERED	0x00	Set the device to bus powered.
USB_DEVICE_STATUS_SELF_POWERED	0x01	Set the device to self powered.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. This is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the responsibility of the calling function to preserve the values across calls to fastcall16 functions.

Sample Firmware Source Code

The following C code shows you how to use the USBFS User Module in a simple HID application. Once connected to a PC host, the device enumerates as a 3 button mouse. When the code is run the mouse cursor zigzags from right to left. This code illustrates how the USBFS Setup Wizard configures the user module:

```
//----------------------------------------------------------------------------
// C main line
//----------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules


BYTE abMouseData[3] = {0,0,0};
BYTE i = 0;

void main (void)
{
     M8C_EnableGInt; //Enable Global Interrupts
     USBFS_Start(0, USB_5V_OPERATION); //Start USBFS Operation using device 0 at 5V
while(!USBFS_bGetConfiguration()); //Wait for Device to enumerate
     USBFS_LoadInEP(1, abMouseData, 3, USB_NO_TOGGLE); //Begin initial USB transfers

     while(1)
     {
        if(USBFS_bGetEPAckState(1) != 0) //Check and see if ACK has occured
        {
            USBFS_LoadInEP(1, abMouseData, 3, USB_TOGGLE); //Load EP1 with mouse data
            if(i==128)
 abMouseData[1] = 0x05;  //Start moving the mouse to the right
            else if(i==255)
 abMouseData[1] = 0xFB; //Start moving the mouse to the left
            i++;
        }
     }
}
```

Note For host reset tolerant USB code, the following line should be added to USBFSINT.asm in the user code area of the USB Reset interrupt:

```
mov [_busReset], 0x01 //set USB bus reset flag
```

The following Assembly code shows you how to use the USBFS User Module in a simple HID application. Once connected to a PC host the device enumerates as a 3 button mouse. When the code is run the mouse cursor zigzags from right to left. This code illustrates how the USBFS Setup Wizard configures the user module:

```
;------------------------------------------------------------------------------
; Assembly main line
;------------------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

area bss(RAM) // inform assembler that variables follow
abMouseData: blk 3 // USBFS data variable
i: blk 1 // count variable
area text(ROM,REL) // inform assembler that program code follows

_main:
          OR F,1
          ; Start USBFS Operation using device 0
PUSH X
          MOV X,3
          MOV A,0
          LCALL USBFS_Start
          POP X
          ; Wait for Device to enumerate
.no_device:
PUSH X
          LCALL USBFS_bGetConfiguration
        POP X
          CMP A,0
          JZ .no_device
          ; Enumeration is completed load endpoint 1. Do not toggle the first time
; USBFS_LoadInEP(1, abMouseData, 3, USB_NO_TOGGLE);
PUSH X
          MOV A,0
          PUSH A
          MOV A,0
          PUSH A
          MOV A,3
          PUSH A
          MOV A,0
          PUSH A
          MOV A,71
          PUSH A
          MOV A,1
          PUSH A
          LCALL USBFS_LoadInEP
          ADD SP,250
          POP X

_Endless_Loop:
```

```
          PUSH X
          MOV A,1
          LCALL USBFS_bGetEPAckState
          POP X
          CMP A,0
          JZ _Endless_Loop
          ; ACK has occurred, load the endpoint and toggle the data bit
; USBFS_LoadInEP(1, abMouseData, 3, USB_TOGGLE);
PUSH X
          MOV A,1
          PUSH A
          MOV A,0
          PUSH A
          MOV A,3
          PUSH A
          MOV A,0
          PUSH A
          MOV A,71
          PUSH A
          MOV A,1
          PUSH A
          LCALL USBFS_LoadInEP
          ADD SP,250
          POP X

          ; When our count hits 128
CMP [i],128
          JNZ _Move_Left
          JMP _Move_Right

_Move_Right:
          ; Start moving the mouse to the right
MOV [abMouseData+1],5
          JMP _Increment_i

_Move_Left:
          ; When our counts hits 255
; Start moving the mouse to the left
CMP [i], 255
          JNZ _Increment_i
          MOV [abMouseData+1],251

_Increment_i:
          INC [i]
          JMP _Endless_Loop
          .terminate:
          jmp .terminate
```

USBFS Setup Corresponding to the Example Code

1. Create a new project with a base part supported by the USBFS User Module (such as CY8C24894).
2. In the Device Editor, click Protocols and add the USBFS User Module.
3. Right click the USBFS icon and select the USB Setup Wizard.
4. Select the Human Interface Device (HID) radio button. Optional step: rename the user module from USBFS_1 to USBFS to match sample code.
5. Right-click the USBFS User Module icon in the Device Editor to open the USBFS Setup Wizard.
   - Click the Import HID Report Template operation and make the name Import HID Report Template italics to show that it is a label.
   - Select the 3 button mouse template.
   - Click the Apply operation on the right side of the template.
   - Select the Add String operation to add Manufacturer and Product strings.
   - Edit the device attributes: Vendor ID, Product ID, and select strings.
   - Edit the interface attributes: select HID for the Class field.
   - Edit the HID class descriptor: select the 3 button mouse for the HID Report field.
   - Click **OK** to save the USB descriptor information.
6. Generate the Application.
7. Copy the Sample code and paste it in the main.c.
8. Do a **Rebuild all**.

Descriptor	Data
USB User Module descriptor root	Device name
Device descriptor	Device
Device attributes	
Vendor ID	Use company VID
Product ID	Use product PID
Device release (bcdDevice)	0000
Device class	Defined in interface descriptor
Subclass	No subclass
Manufacturer string	My company
Product string	My mouse
Serial number string	No string
Configuration descriptor	Configuration
Configuration attributes	
Configuration string	No string

Descriptor	Data
Max power	100
Device power	Bus powered
Remote wakeup	Disabled
Interface descriptor	Interface
Interface attributes	
Interface string	No string
Class	HID
Subclass	No subclass
HID class descriptor	
Descriptor type	Report
Country code	Not supported
HID report	3-button mouse
Endpoint descriptor	ENDPOINT_NAME
Endpoint attributes	
Endpoint number	1
Direction	IN
Transfer type	INT
Interval	10
Max packet size	8
String/LANGID	
String descriptors	USBFS
LANGID	
String	My company
String	My mouse
Descriptor	
HID report descriptor root	USBFS
HID report descriptor	USBFS

Further Information

USB Standard Device Requests

This section describes the requests supported by the USBFS User Module. If a request is not supported the USBFS User Module normally responds with a STALL, indicating a request error.

Standard Device Request	USB User Module Support Description	USB 2.0 Spec Section
CLEAR_FEATURE	Device:	9.4.1
	Interface: not supported.	
	Endpoint	
GET_CONFIGURATION	Returns the current device configuration value.	9.4.2
GET_DESCRIPTOR	Returns the specified descriptor.	9.4.3
GET_INTERFACE	Returns the selected alternate interface setting for the specified interface.	9.4.4
GET_STATUS	Device:	9.4.5
	Interface:	
	Endpoint:	
SET_ADDRESS	Sets the device address for all future device accesses.	9.4.6
SET_CONFIGURATION	Sets the device configuration.	9.4.7
SET_DESCRIPTOR	This optional request is not supported.	9.4.8
SET_FEATURE	Device: DEVICE_REMOTE_WAKEUP support is selected by the bRemoteWakeUp user module parameter. TEST_MODE is not supported.	9.4.9
	Interface: Not supported.	
	Endpoint: The specified Endpoint is halted.	
SET_INTERFACE	Not supported.	9.4.10
SYNCH_FRAME	Not supported. Future implementations of the user module will add support to this request to enable Isochronous transfers with repeating frame patterns.	9.4.11

HID Class Request

Class Request	USBFS User Module Support Description	Device Class Definition for HID - Section
GET_REPORT	Allows the host to receive a report by way of the Control pipe.	7.2.1
GET_IDLE	Reads the current idle rate for a particular Input report.	7.2.3
GET_PROTOCOL	Reads which protocol is currently active (either the boot or the report protocol).	7.2.5
SET_REPORT	Allows the host to send a report to the device, possibly setting the state of input, output, or feature controls.	7.2.2
SET_IDLE	Silences a particular report on the Interrupt In pipe until a new event occurs or the specified amount of time passes.	7.2.4
SET_PROTOCOL	Switches between the boot protocol and the report protocol (or vice versa).	7.2.6

USBFS Setup Wizard

This section details all the USBFS descriptors given by the USBFS User Module. The descriptions include the descriptor format and how user module parameters map into the descriptor data.

The USBFS Setup Wizard is a tool given by Cypress to assist engineers in the designing of USB devices. The setup wizard displays the device descriptor tree; when expanded the following folders that are part of the standard USB descriptor definitions appear:

- Device attributes
- Configuration descriptor
- Interface descriptor
- HID Class descriptor
- Endpoint descriptor
- String/LANGID
- HID Descriptor

To access the setup wizard, right click the USB User Module icon in the device editor and click the USB Setup Wizard... menu item.

When the device descriptor tree is fully expanded, you see all the setup wizard options. The left side displays the name of the descriptor, the center displays the data, and the left displays the operation available for a particular descriptor. In some instances, a descriptor has a pull down menu that presents available options.

Descriptor	Data		Operations
USBFS User Module descriptor root	“Device Name”		Add device
Device descriptor	DEVICE_1		Remove/Add configuration
Device attributes			
Vendor ID	FFFF		
Product ID	FFFF		
Device release (bcdDevice)	0000		
Device class	Undefined	pull down	
Subclass	No subclass	pull down	
Protocol	None	pull down	
Manufacturer string	No string	pull down	
Product string	No string	pull down	
Serial number string	No string	pull down	
Configuration descriptor	CONFIG_NAME		Remove/Add interface
Configuration attributes			
Configuration string	No string	pull down	
Max power	100		
Device power	Bus powered	pull down	
Remote wakeup	Disabled	pull down	
Interface descriptor	INTERFACE_NAME		Remove/Add endpoint
Interface attributes			
Interface string	No string	pull down	
Class	Vendor specific	pull down	
Subclass	No subclass	pull down	
HID class Descriptor			
Descriptor type	Report	pull down	
Country code	Not supported	pull down	
HID report	None	pull down	
Endpoint descriptor	ENDPOINT_NAME		Remove
Endpoint attributes			
Endpoint number	0		
Direction	IN	pull down	

Descriptor	Data		Operations
Transfer type	CNTRL	pull down	
Interval	10		
Max packet size	8		
String/LANGID			
String descriptors	Device name		Add string
LANGID		pull down	
String	Selected string name		Remove
Descriptor			
HID Descriptor	Device name		Import HID Report Template

Understanding the USB Setup Wizard

The USB Setup Wizard window is a table that presents three major areas for programming. The first area is the Descriptor USBFS User Module, the second is the String/LANGID, and the third is the Descriptor HID report. Use the two buttons below the table perform the selected command.

The first section presents the Descriptor. The second section presents the String/LANGID; when a string ID is required, this area is used to input that string. To add a string for a USB device, click on the **Add String** operation. The software adds a row and prompts you to Edit your string here. Type the new string then click **Save/Generate**. Once the string is saved, it is available for use in the Descriptor section from the pull down menus. If you close without saving, the string is lost.

The third area presents the HID Report Descriptor Root. From here you add or import an HID Report for the selected device.

USB User Module Descriptor Root

The first column displays folders to expand and collapse. For the purpose of this discussion, you must fully expand the tree that all options are visible. The setup wizard permits the entering of data into the middle Data column; if there is a pull down menu, use it to select a different option. If there is no pull down menu, but there is data, use the cursor to highlight and select the data, then overwrite that data with another value or text option. All the values must meet the USB 2.0 Chapter 9 Specifications.

The first folder displayed at the top is the *USB User Module Descriptor Root*. It has the user module name in the Data column (this is the user module name given to it by the software. This user module is the one placed in the Interconnect View. The Add Device operation on the right hand column adds another USB device complete with all the different fields required for describing it. The new USB device descriptor is listed at the bottom after the Endpoint Descriptor. Click **OK** to save. If you do not save the newly added device, it is not available for use.

Device Descriptor has DEVICE_NUMBER as the Data; it may be removed or a configuration added. All the information about a particular USB device may be entered by over writing the existing data or by using a pull down menu.

When the input of data is complete, either by using the pull down menus or by typing alphanumeric text in the appropriate spots, click **OK** to save.

USB Suspend, Resume, and Remote Wakeup

The USB Suspend, Resume, and Remote Wakeup features are tightly coupled into the user application. You should write firmware to lower power consumption appropriately for your device.

USFS Activity Monitoring

The USBFS_bCheckActivity API function gives a means to check if any USB bus activity occurred. The application uses the function to determine if the conditions to enter USB Suspend were met.

USBFS Suspend

Once the conditions to enter USB suspend are met, the application takes appropriate steps to reduce current consumption to meet the suspend current requirements. To put the USB SIE and transceiver into power down mode, the application calls M8C_Sleep macro and the USBFS_bCheckActivity API to detect USB activity. The sleep macro disables the USBFS block, but maintains the current USB address (in the USBCR register).

USBFS Resume

While the device is suspended, it periodically checks to determine if the conditions to leave the suspended state were met. One way to check resume conditions is to use the sleep timer to periodically wake the device. If the resume conditions were met, the application exits the sleep loop, which enables the USBFS SIE and Transceiver, bringing them out of power down mode. It does not change the USB address field of the USBCR register, maintaining the USB address previously assigned by the host.

USBFS Remote Wakeup

If the device supports remote wakeup, the application is able to determine if the host enabled remote wakeup with the USBFS_bRWUEnabled API function. When the device is suspended and it determines the conditions to initiate a remote wakeup are met, the application uses the USBFS_Force API function to force the appropriate J and K states onto the USB Bus, signaling a remote wakeup.

Creating Vendor Specific Device Requests and Overriding Existing Requests

The USBFS User Module supports vendor specific device requests by providing a dispatch routine for handling setup packet requests. You can also write your own routines that override any of the supplied standard and class specific routines, or enable unsupported request types.

Processing of USBFS Device Requests

All control transfers, including vendor specific and overridden device requests, are composed of:

- A setup stage where request information is moved from host to device.
- A data stage consisting of zero or more data transactions with data send in the direction specified in the setup stage.
- A status stage that concludes the transfer.

In the USBFS User Module, all control transfers are handled by the Endpoint 0 Interrupt Service Routine (USBFS_EP0_ISR).

The Endpoint 0 Interrupt Service Routine transfers control of all setup packets to the dispatch routine, which routes the request to the appropriate handler based upon the bmRequestType field. The handler initializes specific user module data structures and transfers control back to the Endpoint 0 ISR. A handler for vendor specific or override device request is given by the application. The user module handles the data and status stages of the transfer without any involvement of the user application. Upon completion of the transfer, the user module updates a completion status block. The status block is monitored by the application to determine if the vendor specific device request is complete.

All setup packets enter the USBFS_EP0_ISR, which routes the setup packet to the USBFS_bmRequestType_Dispatch routine. From here all the standard device requests as well as the vendor specific device requests are dispatched. The device request handlers must prepare the application to receive data for control writes or prepare the data for transmission to the host for control reads. For no-data control transfers, the handler extracts information from the setup packet itself.

The USBFS User Module processes the data and status stages exactly the same way for all requests. For data stages, the data is copied to or from the control endpoint buffer (registers EP0DATA0-EP0DATA7) depending upon the direction of the transaction.

Vendor Specific Device Request Dispatch Routines

Depending upon the application requirements, the USBFS User Module dispatches up to eight types of vendor specific device requests based upon the bmRequestType field of the setup packet. Refer to section 9.3 of the USB 2.0 specification for a discussion of USB device requests and the bmRequestType field. The eight types of vendor specific device requests the USBFS User Module dispatches are listed in the table Vendor Specific Request Dispatch Routine Names.

Table 3. Vendor Specific Request Dispatch Routine Names

Direction	Recipient	Dispatch Routine Entry Point	Enable Flag
Host to Device (Control Write)	Device	USB_DT_h2d_vnd_dev_Dispatch	USB_CB_h2d_vnd_dev
	Interface	USB_DT_h2d_vnd_ifc_Dispatch	USB_CB_h2d_vnd_ifc
	Endpoint	USB_DT_h2d_vnd_ep_Dispatch	USB_CB_h2d_vnd_ep
	Other	USB_DT_h2d_vnd_oth_Dispatch	USB_CB_h2d_vnd_oth
Device to Host (Control Read)	Device	USB_DT_d2h_vnd_dev_Dispatch	USB_CB_d2h_vnd_dev
	Interface	USB_DT_d2h_vnd_ifc_Dispatch	USB_CB_d2h_vnd_ifc
	Endpoint	USB_DT_d2h_vnd_ep_Dispatch	USB_CB_d2h_vnd_ep
	Other	USB_DT_d2h_vnd_oth_Dispatch	USB_CB_d2h_vnd_oth

Follow these steps for an application to give an assembly language dispatch routine for the vendor specific device request.

1. In the *USBFS.inc* file, enable support for the vendor specific dispatch routine. Find the dispatch routine enable flag and set EQU to 1.
2. Write an appropriately named assembly language routine to handle the device request. Use the entry points listed in the table above.

Override Existing Request Routines

To override a standard or class specific device request, or enable an unsupported device request, you must do the following:

1. In the *USBFS.inc* file, redefine the specific device request as USB_APP_SUPPLIED.
2. Write an appropriately named assembly language function to handle the device request. The name of the assembly language function is APP_ plus the device name.

For example, to override the supplied HID class Set Report request, USB_CB_SRC_h2d_cls_ifc_09, enable the routine with these changes to *USBFS.inc*:

```
;@PSoC_UserCode_BODY_1@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.

; Enable an override of the HID class Set Report request.
USB_CB_SRC_h2d_cls_ifc_09: EQU USB_APP_SUPPLIED

  ;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```

Then, write an assembly language routine named APP_USB_CB_SRC_h2d_cls_ifc_09. Device request names are derived from the USB bmRequestType and bRequest values (USB spec**ification** Table 9-2).

This code is a stub for the assembly routine for the previous example:

```
export APP_USB_CB_SRC_h2d_cls_ifc_09
APP_USB_CB_SRC_h2d_cls_ifc_09:

;Add your code here.

; Long jump to the appropriate return entry point for your application.
LJMP USBFS_InitControlWrite
```

Dispatch and Override Routine Requirements.

At a minimum, the dispatch or override routine must return control back to the Endpoint 0 ISR by a LJMP to one of the Endpoint 0 ISR Return Points listed in the following table. The routine may destroy the A and X registers, but the Stack Pointer (SP) and any other relevant context must be restored prior to returning control to the ISR.

Table 4. Endpoint 0 ISR Return Points

Return Entry Point	Required Data Items	Description
USBFS_Not_Supported	Use this return point when the request is not supported. It STALLs the request.	
	Data Items: None	
USBFS_InitControlRead	This return point is used to initiate a Control Read transfer.	
	USBFS_DataSource (BYTE)	The data source is RAM or ROM (USBFS_DS_RAM or USBFS_DS_ROM). This is necessary since different instructions are used to move the data from the source ROMX or MOV.
	USBFS_TransferSize (WORD)	The number of data bytes to transfer.
	USBFS_DataPtr (WORD)	RAM or ROM address of the data.
	USBFS_StatusBlockPtr (WORD) optional	Address of a status block allocated with the USBFS_XFER_STATUS_BLOCK macro.
USBFS_InitControlWrite	This return point is used to initiate a Control Write transfer.	
	USBFS_DataSource (BYTE)	USBFS_DS_RAM (the destination for control writes must RAM).
	USBFS_TransferSize (WORD)	Size of the application buffer to receive the data
	USBFS_DataPtr (WORD)	RAM address of the application buffer to receive the data
	USBFS_StatusBlockPtr (WORD) optional	Address of a status block allocated with the USBFS_XFER_STATUS_BLOCK macro.
USB_InitNoDataControlTransfer	This return point is used to initiate a No Data Control transfer.	
	USBFS_StatusBlockPtr (WORD) optional	Address of a status block allocated with the USBFS_XFER_STATUS_BLOCK macro.

Status Completion Block

The status completion block contains two data items, a one byte completion status code and a two byte transfer length. The “main” application monitors the completion status to determine how to proceed. Completion status codes are found in the following table. The transfer length is the actual number of data bytes transferred.

Table 5. USBFS Transfer Completion Codes

Completion Code	Description
USB_XFER_IDLE (0x00)	USB_XFER_IDLE indicates that the associated data buffer does not have valid data and the application should not use the buffer. The actual data transfer takes place while the completion code is USB_XFER_IDLE, although it does not indicate a transfer is in progress.
USB_XFER_STATUS_ACK (0x01)	USB_XFER_STATUS_ACK indicates the control transfer status stage completed successfully. At this time, the application uses the associated data buffer and its contents.
USB_XFER_PREMATURE (0x02)	USB_XFER_PREMATURE indicates that the control transfer was interrupted by the SETUP of a subsequent control transfer. For control writes, the contents of the associated data buffer contains the data up to the premature completion.
USB_XFER_ERROR (0x03)	USB_XFER_ERROR indicates that the expected status stage token was not received.

Customizing the HID Class Report Storage Area

If you enable optional HID class support, the Setup Wizard creates a fixed-size report storage area for data reports from the HID class device. It creates separate report areas for IN, OUT, and FEATURE reports. This area is sufficient for the case where no Report ID item tags are present in the Report descriptor and therefore only one Input, Output, and Feature report structure exists. If you want better control over the report storage size or want to support multiple report IDs, you will need to do the following:

1. Use the wizard to specify your device description, endpoints, and HID reports then generate the application.
2. Disable the wizard defined report storage area in *USB_descr.asm*.
3. Copy the wizard created code that defines the report storage area.
4. Paste it into the protected user code area in *USB_descr.asm* or a separate assembly language file.
5. Customize the code to define the report storage area.

Specify Your Device and Generate Application

Use the USB setup wizard to specify your device description, endpoints, and HID reports. Click the **Generate Application** button in PSoC Designer.

Disable the Wizard Defined Report Storage Area

In the *USB_descr.asm* file, disable the wizard defined storage area by uncommenting the WIZARD_DEFINED_REPORT_STORAGE line in the custom code area as shown.

```
WIZARD: equ 1
WIZARD_DEFINED_REPORT_STORAGE: equ 1
   ;--------------------------------------------------
;@PSoC_UserCode_BODY_1@ (Do not change this line.)
   ;--------------------------------------------------
   ; Insert your custom code below this banner
   ;--------------------------------------------------
   ; Redefine the WIZARD equate to 0 below by
; uncommenting the WIZARD: equ 0 line
; to allow your custom descriptor to take effect
   ;--------------------------------------------------

; WIZARD: equ 0
   WIZARD_DEFINED_REPORT_STORAGE: equ 0
   ;--------------------------------------------------
   ; Insert your custom code above this banner
   ;--------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```

Copy the Wizard Created Code

Find this code in *USB_descr.asm*.

```
;----------------------------------------------------------------------
; HID IN Report Transfer Descriptor Table for ()
;----------------------------------------------------------------------
IF WIZARD_DEFINED_REPORT_STORAGE
AREA  UserModules     (ROM,REL,CON)
.LITERAL
USB_D0_C1_I0_IN_RPTS:
TD_START_TABLE 1                              ; Only 1 Transfer Descriptor
  TD_ENTRY       USB_DS_RAM, USB_HID_RPT_3_IN_RPT_SIZE, USB_INTERFACE_0_IN_RPT_DATA,
NULL_PTR
.ENDLITERAL
ENDIF ; WIZARD_DEFINED_REPORT_STORAGE
```

There are three sections, one each for the IN, OUT, and FEATURE reports. Copy all three sections.

Paste the Code Into the Protected User Code Area

You can paste the code into the protected user code area of *USB_descr.asm* shown or a separate assembly language file.

```
    ;---------------------------------------------------
    ;@PSoC_UserCode_BODY_2@ (Do not change this line.)
    ;---------------------------------------------------
    ; Redefine your descriptor table below. You might
    ; cut and paste code from the WIZARD descriptor
    ; above and then make your changes.
    ;---------------------------------------------------

    ;---------------------------------------------------
    ; Insert your custom code above this banner
    ;---------------------------------------------------
    ;@PSoC_UserCode_END@ (Do not change this line.)
; End of File USB_descr.asm
```

Customize the Code to Define the Report Storage Area

To define the report storage area, you will write your own transfer descriptor table entries. The table contains entries to define storage space for the required data items. Each transfer descriptor entry in the table creates a new Report ID. IDs are numbered consecutively, starting with zero. Report ID 0 is not used; you cannot specify a Report ID of 0, but the transfer descriptor entry specified for the ID 0 will be used in the case that no Report IDs are present in the Report descriptor. For the sake of code efficiency, you should use Report IDs in order starting with ID 1.

Table 6. Transfer Descriptor Table Entries

Table Entry	Required Data Items	Description
TD_START_TABLE	USBFS_NumberOfTableEntries	Number of Report IDs defined. IDs are numbered consecutively from 0. Report ID 0 is not used.
TD_ENTRY		
	USBFS_DataSource	The data source is RAM or ROM (USBFS_DS_RAM or USBFS_DS_ROM).
	USBFS_TransferSize	Size of the data transfer in bytes. The first byte is the Report ID.
	USBFS_DataPtr	RAM or ROM address of the data transfer.
	USBFS_StatusBlockPtr	Address of a status block allocated with the USBFS_XFER_STATUS_BLOCK macro.

The following example sets up the unused Report ID 0, and two other IN reports with different sizes. Note Conditional assembly statements are only necessary if you place the code in the protected user code area of *USB_descr.asm.*

```
;-----------------------------------------------------------------------
; HID IN Report Transfer Descriptor Table for ()
;-----------------------------------------------------------------------
IF WIZARD_DEFINED_REPORT_STORAGE
ELSE

_ID0_RPT_SIZE:   EQU 8       ; 7 data bytes + report ID = 8 bytes (unused)
_SM_RPT_SIZE:    EQU 3       ; 2 data bytes + report ID = 3 bytes
_LG_RPT_SIZE:    EQU 5       ; 4 data bytes + report ID = 5 bytes

AREA data (RAM, REL, CON)

EXPORT _ID0_RPT_PTR
_ID0_RPT_PTR: BLK 8         ; Allocates space for report ID0 (unused)
EXPORT _SM_RPT_PTR
_SM_RPT_PTR:    BLK 3       ; Allocates space for report ID1
EXPORT _LG_RPT_PTR
_LG_RPT_PTR:    BLK 5       ; Allocates space for report ID2

AREA bss (RAM, REL, CON)

EXPORT _SM_RPT_STS_PTR
_SM_RPT_STS_PTR: USBFS_XFER_STATUS_BLOCK
EXPORT _LG_RPT_STS_PTR
_LG_RPT_STS_PTR: USBFS_XFER_STATUS_BLOCK

AREA  UserModules     (ROM,REL,CON)
.LITERAL
EXPORT USB_D0_C1_I0_IN_RPTS:
  TD_START_TABLE 3
  TD_ENTRY  USBFS_DS_RAM, _ID0_RPT_SIZE, _ID0_RPT_PTR, NULL_PTR ; ID0 unused
  TD_ENTRY  USBFS_DS_RAM, _SM_RPT_SIZE, _SM_RPT_PTR, _SM_RPT_STS_PTR ; ID1
  TD_ENTRY  USBFS_DS_RAM, _LG_RPT_SIZE, _LG_RPT_PTR, _LG_RPT_STS_PTR ; ID2
.ENDLITERAL

ENDIF ; WIZARD_DEFINED_REPORT_STORAGE
```

Version History

Version	Originator	Description
1.5	MAXK	Updated USBFS sample code to properly handle PC resets.
1.60	DHA	1. Changed variable area location to allow code sublimation. 2. Removed .Literal/.Endliteral directives around jmp instructions.
1.70	DHA	1. Added wizard help button and help file. 2. Added verification of the writing EP0_CR. 3. Added verification of the SIE MODEs and ACK bit into the EP0 ISR. 4. Updated the constant area definition in the user module.
1.80	DHA	1. Moved the USBFS_USB_EP(X)_BIT_LOOKUP tables from the "AREA UserModules" to the "AREA lit". 2. Added initialization of the USB_Protocol variable to comply with HID specification. 3. Added both the "#pragma fastcall16" and "extern" directives for the "USBFS_bRWUEnabled" API function. 4. Updated Assembly and C Prototype for the USBFS_bGetEPAckState API function in the "Application Programming Interface" section.
2.00	DHA	1. Fixed ‘@INSTANCE_NAME`_bCheckActivity’ API function to prevent missed activity. 2. Added CYRF89235 device support. 3. Corrected description of DisableOutEP API function in the user module datasheet.
2.10	MYKZ	1. Corrected the USBFS_bReadOutEP function to avoid CPU clock change for CY8C20xx6 family. 2. Updated Stop() function to fix problem with USB interrupts disabling.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.