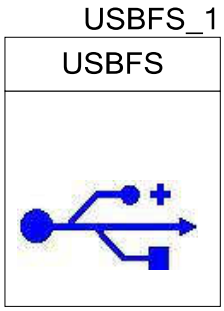
Full Speed USB (USBFS)
1.10

Features



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

General Description

The USBFS component provides a USB full speed Chapter 9 compliant device framework. The component provides a low level driver for the control endpoint that decodes and dispatches requests from the USB host. Additionally, this component provides a USBFS customizer to enable easy descriptor construction.

You have the option of constructing an HID based device or a generic USB Device. You select HID (and switching between HID and generic) by setting the Configuration/Interface descriptors.

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 examples provided. It is your responsibility to design applications that conform to the USB spec.

USB Compliance for Self Powered Devices

If the device that you are creating will be self-powered, you must connect a GPIO pin to VBUS through a resistive network and write firmware to monitor the status of the GPIO. 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.

The device responds to GET_STATUS requests based on the status set with the USBFS_SetPowerStatus() function. To set the correct status, USBFS_SetPowerStatus() should 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.

PRELIMINARY

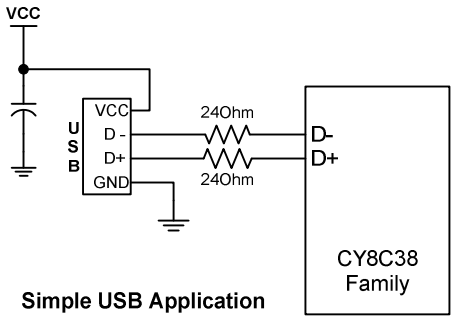
When to use a USBFS

Use the USBFS component when you want to provide your application with a USB 2.0 compliant device interface.

Input/Output Connections

There are no input/output connections for this component in a design. You need to configure the USBFS pin connections using the Design-Wide Resources Pin Editor.

The following diagram shows a simple USB application with the D+ and D- pins from the PSoC device.



Simple USB Application

Component Parameters

Drag a USBFS component onto your design and double-click it to open the Configure USBFS dialog.

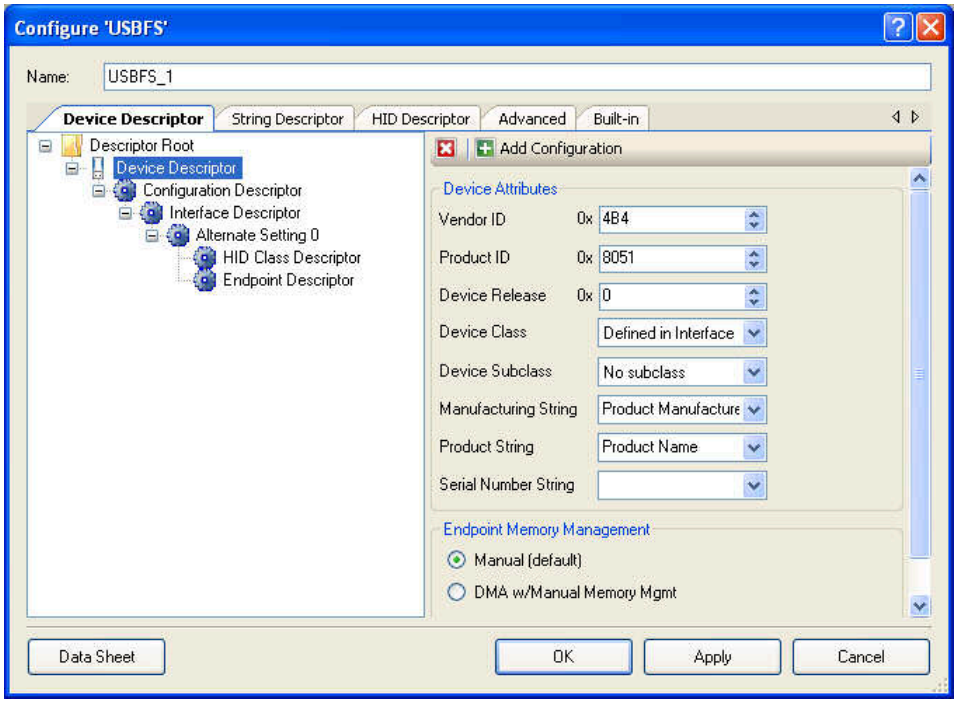
The USBFS Component uses a form driven USBFS dialog to define the USB descriptors for the application. From the descriptors, the customizer personalizes the component.

The component is driven by information generated by the USBFS customizer. This customizer facilitates the construction of the USB descriptors and integrates the information generated into the driver firmware used for device enumeration. The USBFS component does not function without first running the wizard and selecting the appropriate attributes to describe your device. The code generator takes your device information and generated all of the needed USB Descriptors.

The Configure USBFS dialog contains the following tabs and settings:

Device Descriptor Tab

Device Descriptor



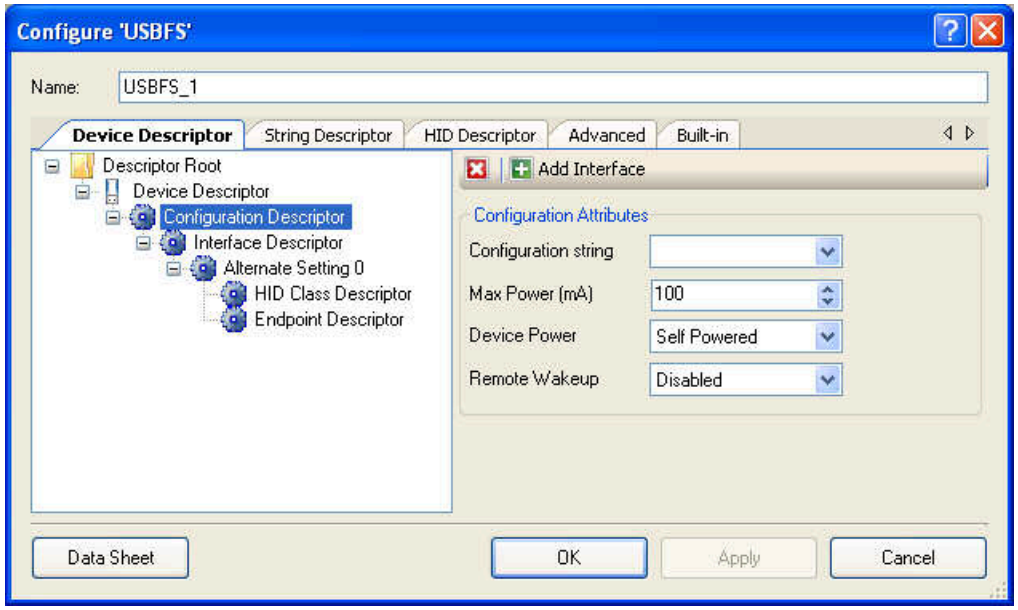
- Vendor ID – Your Company USB Vendor ID (obtained from USB-IF)
- Product ID – Your Specific Product ID
- Device Release – Your Specific Device Release (Device ID)
- Device Class – Device Class is defined in Interface Descriptor or it is Vendor Specific
- Device Subclass – Dependent upon Device Class
- Manufacturing String – Manufacturer Specific Description String
- Product String – Product specific Description String
- Serial Number String

Endpoint Memory Management

Some applications can benefit from using Direct Memory Access (DMA) to move data into and out of the endpoint memory buffers.

- Manual (default) – Select this option to use LoadInEP/ReadOutEP to load and unload the endpoint buffers.
- DMA w/Manual Memory Management – Select this option to expose the DMA interface registers, in addition to the LoadInEP/ReadOutEP functions.

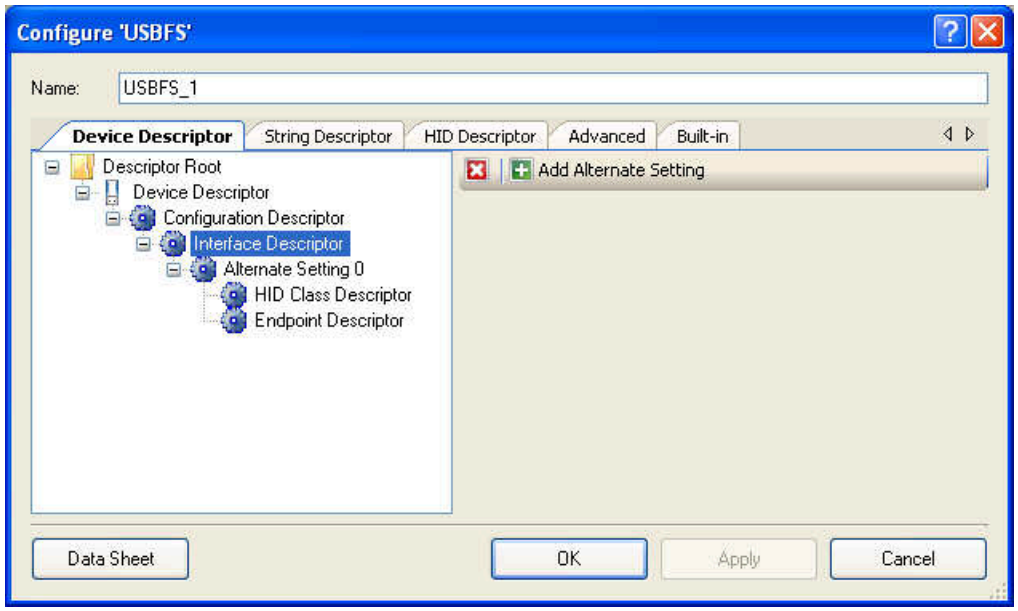
Configuration Descriptor



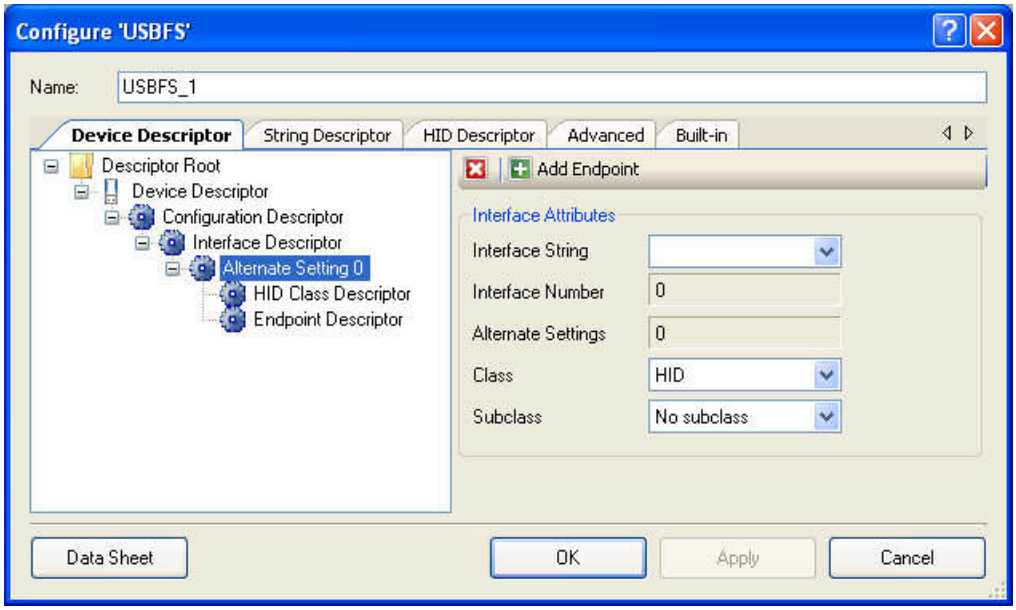
- Configuration string
- Max Power (mA)
- Device Power – Bus Powered or Self Powered Device
- Remote Wakeup – Enabled or Disabled

Interface Descriptor

This level is used to add and delete Interface Alternate Settings. The interfaces are configured in the Alternate Setting.

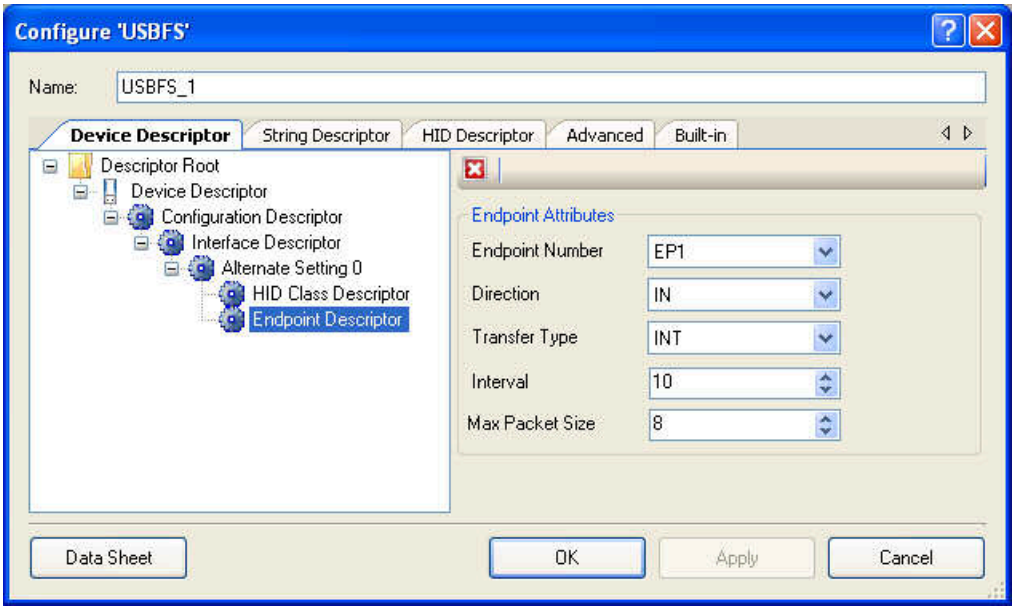


Interface Descriptor—Alternate Settings



- Interface String
- Interface Number—The interface number is computed by the customizer
- Alternate Settings—The alternate setting is computed by the customizer
- Class – HID, Vendor Specific or Undefined
- Subclass – Dependent upon the selected class

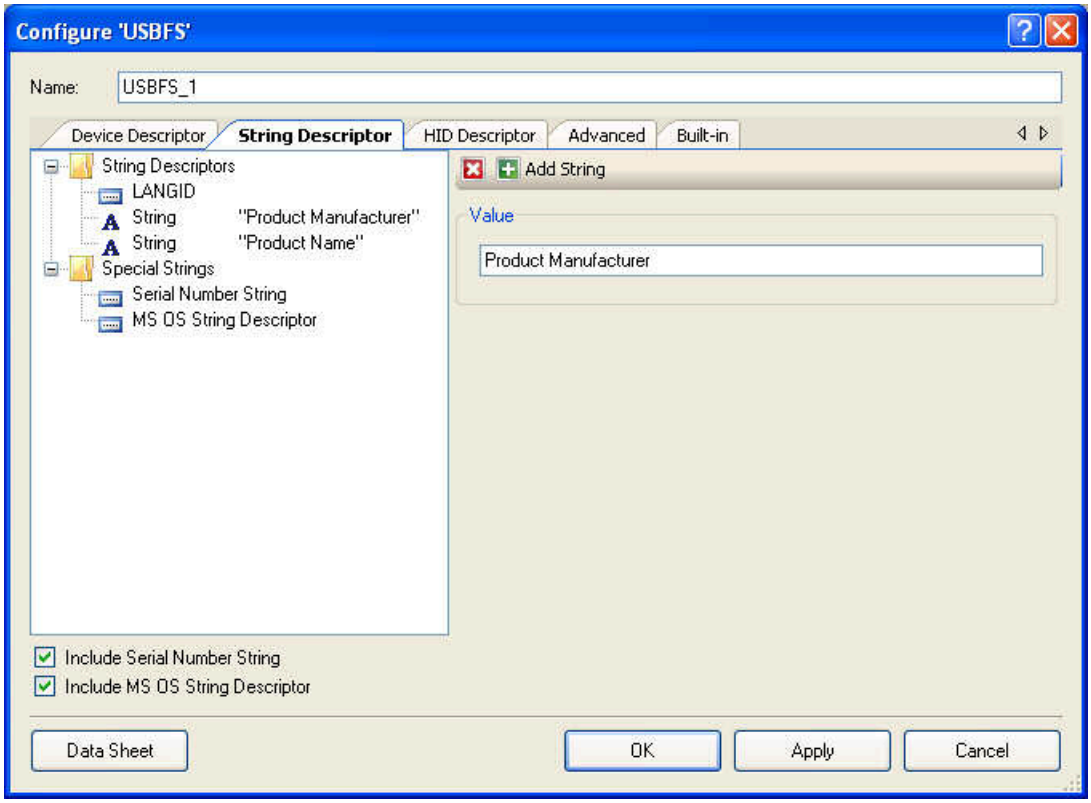
Endpoint Descriptor



- Endpoint Number
- Direction – Input or Output
- Transfer Type – Control, Interrupt, Bulk, or Isochronous Data transfers
- Interval (ms) – Polling interval specific to this endpoint
- Max Packet Size(bytes)
- Double Buffer – Enabled or Disabled

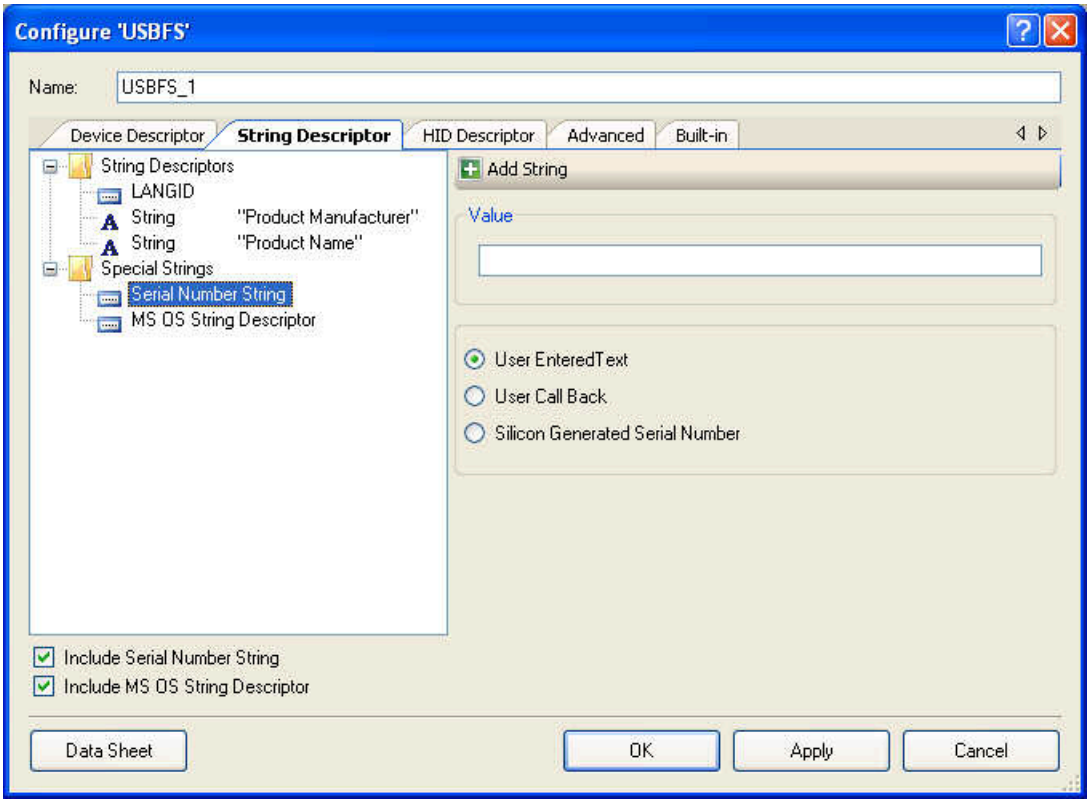
String Descriptor Tab

String Descriptors

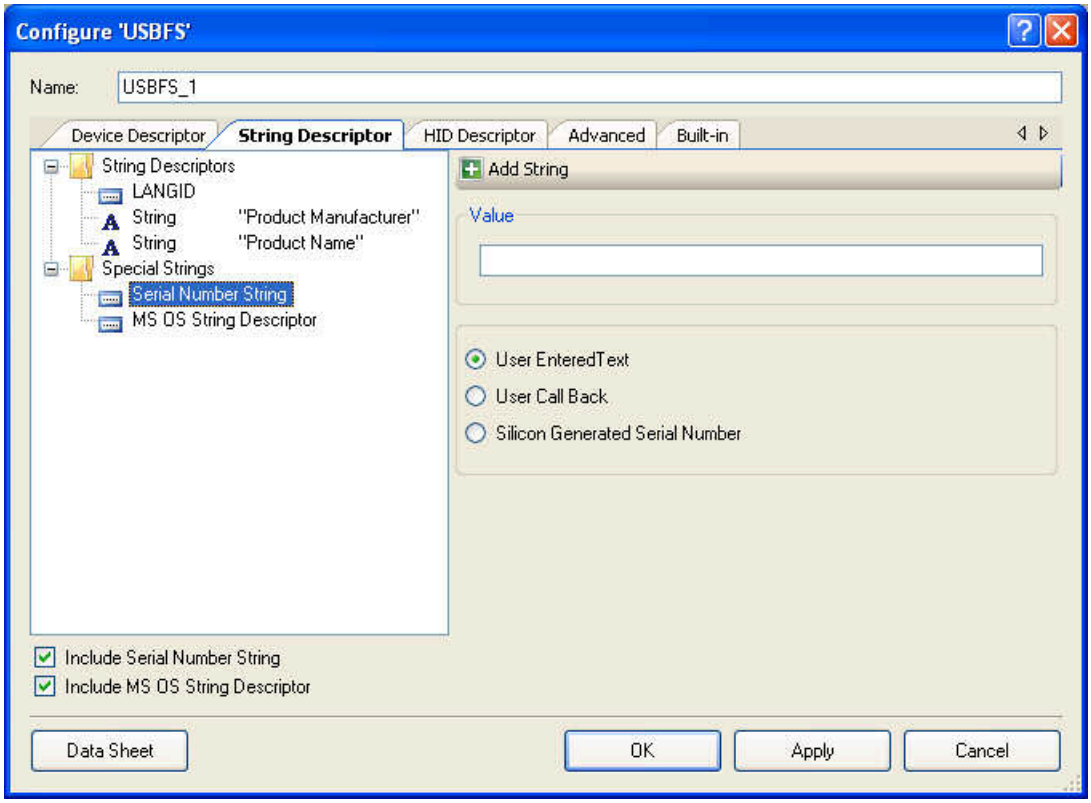


- LANGID -- Select Language
- String -- Value

Serial Number String

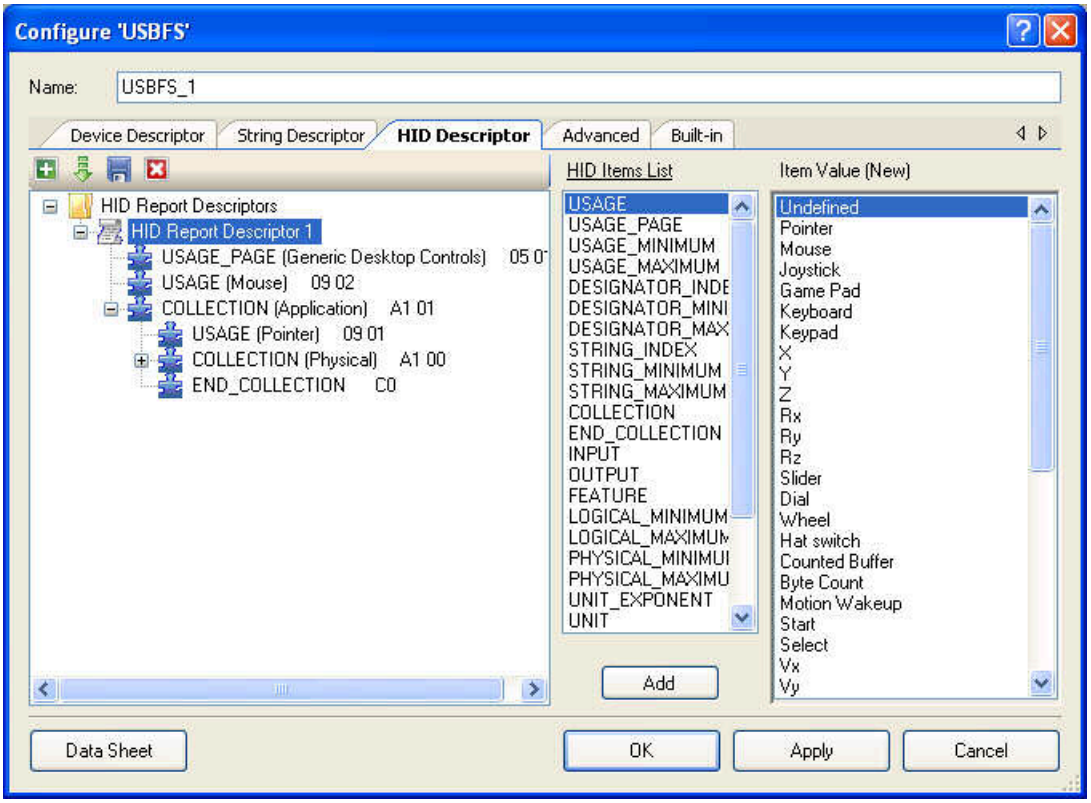


MS OS String Descriptor



HID Descriptor Tab

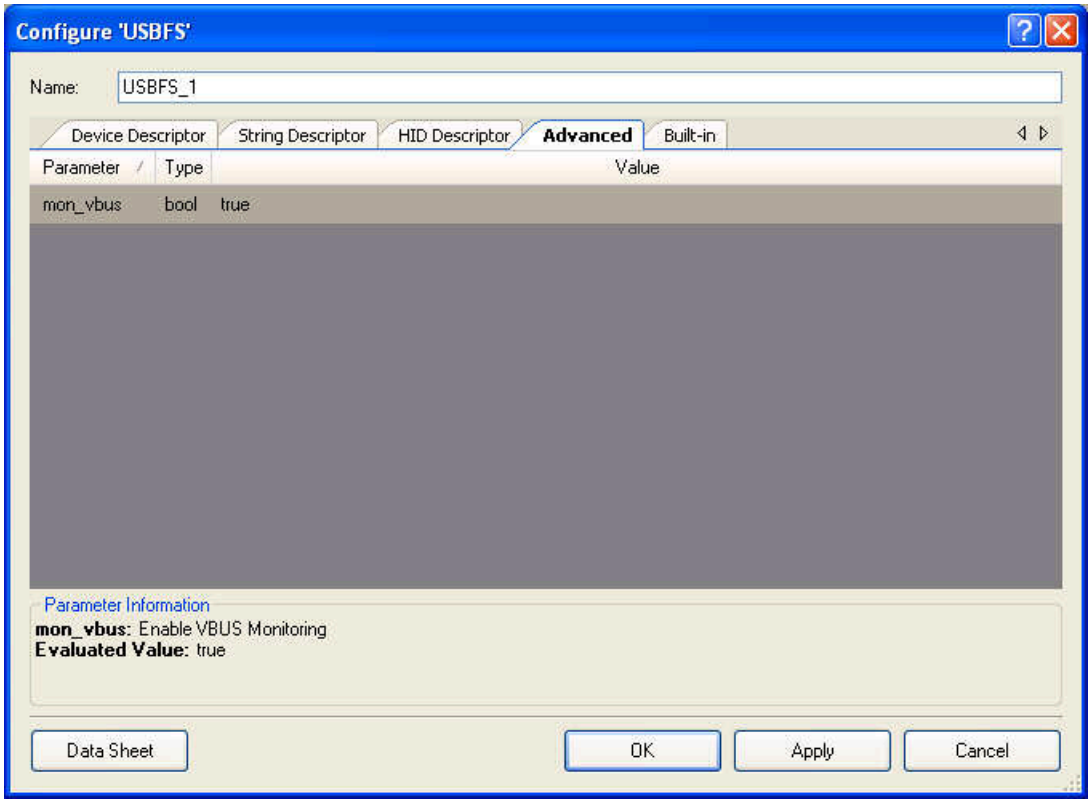
HID Descriptors



- HID Items List
- Item Value

Advanced Tab

Enable VBUS Monitoring



The mon_vbus parameter adds a single VBUS monitor pin to the design. This pin must be connected to VBUS and must be assigned in the pin editor.

Placement

USB is implemented as a fixed function block.

Resources

The USBFS uses the USB fixed function block.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "USBFS_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "USBFS".

Basic USBFS Device API

Function	Description
USBFS_Start	Activate the component for use with the device and specific voltage mode.
USBFS_Stop	Disable component.
USBFS_bCheckActivity	Checks and clears the USB bus activity flag. Returns 1 if the USB was active since the last check, otherwise returns 0.
USBFS_bGetConfiguration	Returns the currently assigned configuration. Returns 0 if the device is not configured.
USBFS_bGetInterfaceSetting	Returns the current alternate setting for the specified interface.
USBFS_bGetEPState	Returns the current state of the specified USBFS endpoint.
USBFS_bGetEPAckState	Identifies whether ACK was set by returning a non-zero value.
USBFS_wGetEPCount	Returns the current byte count from the specified USBFS endpoint.
USBFS_LoadInEP	Loads and enables the specified USBFS endpoint for an IN transfer.
USBFS_bReadOutEP	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.
USB_EnableOutEP	Enables the specified USB endpoint to accept OUT transfers
USBFS_DisableOutEP	Disables the specified USB endpoint to NAK OUT transfers
USBFS_SetPowerStatus	Sets the device to self powered or bus powered
USBFS_Force	Forces a J, K, or SE0 State on the USB D+/D- pins. Normally used for remote wakeup.

Human Interface Device (HID) Class Support

Function	Description
USBFS_UpdateHIDTimer	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.
USBFS_bGetProtocol	Returns the protocol for the specified interface

void USBFS_Start (uint8 bDevice, uint8 bMode)

Description: Performs all required initialization for USBFS Component.

Parameters: (uint8) bDevice: Contains the device number from the desired device descriptor set entered with the USBFS customizer.

(uint8) bMode: The operating voltage. 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 and their associated values are given in the following table.

Power Setting	Notes
USBFS_3V_OPERATION	Disable voltage regulator and pass-thru Vcc for pull-up
USBFS_5V_OPERATION	Enable voltage regulator and use regulator for pull-up

Return Value: None

Side Effects: None

void USBFS_Stop (void)

Description: Performs all necessary shutdown task required for the USBFS Component.

Parameters: None

Return Value: None

Side Effects: None

uint8 USBFS_bCheckActivity(void)

Description: Returns the activity status of the bus. Clears the status hardware to provide fresh activity status on the next call of this routine.

Parameters: None

Return Value: (uint8) cystatus: Standard API return values.

Return Value	Description
1	If bus activity was detected since the last call to this function
0	If bus activity not was detected since the last call to this function

Side Effects: None

uint8 USBFS_bGetConfiguration(void)

Description: Gets the current configuration of the USB device.

Parameters: None

Return Value: uint8: Returns the currently assigned configuration. Returns 0 if the device is not configured.

Side Effects: None

uint8 USBFS_ bGetInterfaceSetting(uint8 bInterfaceNumber)

Description: Gets the current alternate setting for the specified interface.

Parameters: uint8 bInterfaceNumber: Interface number

Return Value: uint8: Returns the current alternate setting for the specified interface.

Side Effects: None

uint8 USBFS_bGetEPState(uint8 bEPNumber)

Description: Returns the state of the requested endpoint.

Parameters: (uint8) bEPNumber: The endpoint number.

Return Value: (uint8) Returns the current state of the specified USBFS endpoint. Symbolic names provided, and their associated values are given in the following table. Use these constants whenever the you write code to change the state of the endpoints such as ISR code to handle data sent or received.

Return Value	Description
USBFS_NO_EVENT_PENDING	Indicates that the endpoint is awaiting SIE action
USBFS_EVENT_PENDING	Indicates that the endpoint is awaiting CPU action
USBFS_NO_EVENT_ALLOWED	Indicates that the endpoint is locked from access
USBFS_IN_BUFFER_FULL	The IN endpoint is loaded and the mode is set to ACK IN
USBFS_IN_BUFFER_EMPTY	An IN transaction occurred and more data can be loaded
USBFS_OUT_BUFFER_EMPTY	The OUT endpoint is set to ACK OUT and is waiting for data
USBFS_OUT_BUFFER_FULL	An OUT transaction has occurred and data can be read

Side Effects: None

uint8 USBFS_bGetEPAckState(uint8 bEPNumber)

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.

Parameters: (uint8) bEPNumber: Contains the endpoint number.

Return Value: (uint8): If an ACKed transaction occurred then this function returns a non-zero value. Otherwise a zero is returned.

Side Effects: None

uint16 USBFS_wGetEPCount(uint8 bEPNumber)

Description: Returns the transfer count for the requested endpoint. The value from the count registers includes 2 counts for the two byte checksum of the packet. This function subtracts the two counts.

Parameters: (uint8) bEPNumber: Contains the endpoint number.

Return Value: (uint8): Returns the current byte count from the specified USBFS endpoint.

Side Effects: None

void USBFS_LoadInEP(uint8 bEPNumber, uint8 *pData, uint16 wLength)

Description: Loads and enables the specified USB endpoint for an IN interrupt or bulk transfer.

Parameters: (uint8) bEPNumber: Contains the endpoint number.

(uint8) *pData: A pointer to a data array from which the data for the endpoint space is loaded.

(uint16) wLength: The number of bytes to transfer from the array and then send as a result of an IN request. Valid values are between 0 and 256.

Return Value: None

Side Effects: None

Return Value: None

Side Effects: None

uint8 USB_bReadOutEP(uint8 bEPNumber, uint8 *pData, uint16 wLength)

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 or the number of bytes requested by the wCount parameter.

Parameters: (uint8) bEPNumber: Contains the endpoint number.

(uint8) *pData: A pointer to a data array from which the data for the endpoint space is loaded.

(uint16) wLength: The number of bytes to transfer from the array and then send as a result of an IN request. Valid values are between 0 and 256. The function moves fewer than the requested number of bytes if the host sends fewer bytes than requested.

Return Value: None

Side Effects: None

void USBFS_EnableOutEP(uint8 bEPNumber)

Description: Enables the specified endpoint for OUT bulk or interrupt transfers. Do not call this function for IN endpoints.

Parameters: (uint8) bEPNumber: Contains the endpoint number.

Return Value: None

Side Effects: None

void USBFS_DisableOutEP(uint8 bEPNumber)

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

Parameters: (uint8) bEPNumber: Contains the endpoint number.

Return Value: None

Side Effects: None

void USBFS_SetPowerStatus(uint8 bPowerStaus)

Description: Sets the current power status. 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.

Parameters: (uint8) bPowerStaus: Contains the desired power status, one for self powered or zero for bus powered. Symbolic names are provided and their associated values are given here:

Power Status	Description
USBFS_DEVICE_STATUS_BUS_POWERED	Set the device to bus powered.
USBFS_DEVICE_STATUS_SELF_POWERED	Set the device to self powered.

Return Value: None

Side Effects: None

void USBFS_Force(uint8 bState)

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

Parameters: (uint8) bState: A byte indicating which of the four bus states to enable. Symbolic names provided, and their associated values are listed here:

State	Description
USBFS_FORCE_SE0	Force a Single Ended 0 onto the D+/D- lines
USBFS_FORCE_J	Force a J State onto the D+/D- lines
USBFS_FORCE_K	Force a K State onto the D+/D- lines
USBFS_FORCE_NONE	Return bus to SIE control

Return Value: None

Side Effects: None

uint8 USBFS_UpdateHIDTimer(uint8 bInterface)

Description: Updates the HID Report idle timer and returns the status. Reloads the timer if it expires.

Parameters: (uint8) bInterface: Contains the interface number.

Return Value: (uint8): Returns the state of the HID timer. Symbolic names are provided and their associated values are given here:

Return Value	Notes
USBFS_IDLE_TIMER_EXPIRED	The timer expired.
USBFS_IDLE_TIMER_RUNNING	The timer is running.
USBFS_IDLE_TIMER_IDEFINITE	Returned if the report is sent when data or state changes.

Side Effects: None

uint8 USBFS_bGetProtocol(uint8 bInterface)

Description: Returns the hid protocol value for the selected interface.

Parameters: (uint8) bInterface: Contains the interface number.

Return Value: (uint8): Returns the protocol value.

Side Effects: None

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the USBFS component. This example assumes the component has been placed in a design with the default name "USBFS_1."

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

The following C code shows you how to use the USBFS component 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 the how the USBFS customizer configures the component.

```
#include <device.h>

uint8 abMouseData[3] = {0,0,0};
uint8 i = 0;
void main()
{
    CYGlobalIntEnable;                          //Enable Global Interrupts
    USBFS_1_Start(0, USBFS_1_3V_OPERATION);     //Start USBFS Operation/device 0
                                                //and with 3V operation
    while(!USBFS_1_bGetConfiguration());        //Wait for Device to enumerate
```

```
        //Enumeration is completed load endpoint 1. Do not toggle the first time
        USBFS_1_LoadInEP(1, abMouseData, 3);

        while(1)
        {
          while(!USBFS_1_bGetEPAckState(1));  //Wait for ACK before loading data
          //ACK has occurred, load the endpoint and toggle the data bit
          USBFS_1_LoadInEP(1, abMouseData, 3);

          if(i==128)                          //When our count hits 128
            abMouseData[1] = 0x05;            //Start moving the mouse to the right
          else if(i==255)                     //When our counts hits 255
            abMouseData[1] = 0xFB;            //Start moving the mouse to the left
          i++;
        }
    }
```

USBFS Setup Corresponding to the Example Code

Someone needs to walk through the basic steps of placing and configuring the USBFS component here.

- 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.

Descriptor	Data
USB component 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

Descriptor	Data
Serial number string	No string
Configuration descriptor	Configuration
Configuration attributes	
Configuration string	No string
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

USB Standard Device Requests

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

Standard Device Request	USB Component 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 Component 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 Component will add support to this request to enable Isochronous transfers with repeating frame patterns.	9.4.11

HID Class Request

Class Request	USBFS Component 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

USB Suspend, Resume, and Remote Wakeup

The USBFS Component supports USB Suspend, Resume, and Remote Wakeup. Since these features are tightly coupled into the user application, the USBFS Component provides a set of API functions.

USFS Activity Monitoring

The USBFS_bCheckActivity API function provides 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 the USBFS_Suspend API function and the USBFS_bCheckActivity API to detect USB activity. This function disables the USBFS block, but maintains the current USB address (in the USBCR register). The device uses the sleep feature to reduce power consumption.

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 calls the USBFS_Resume API function. This function enables the USBFS SIE and Transceiver, bringing

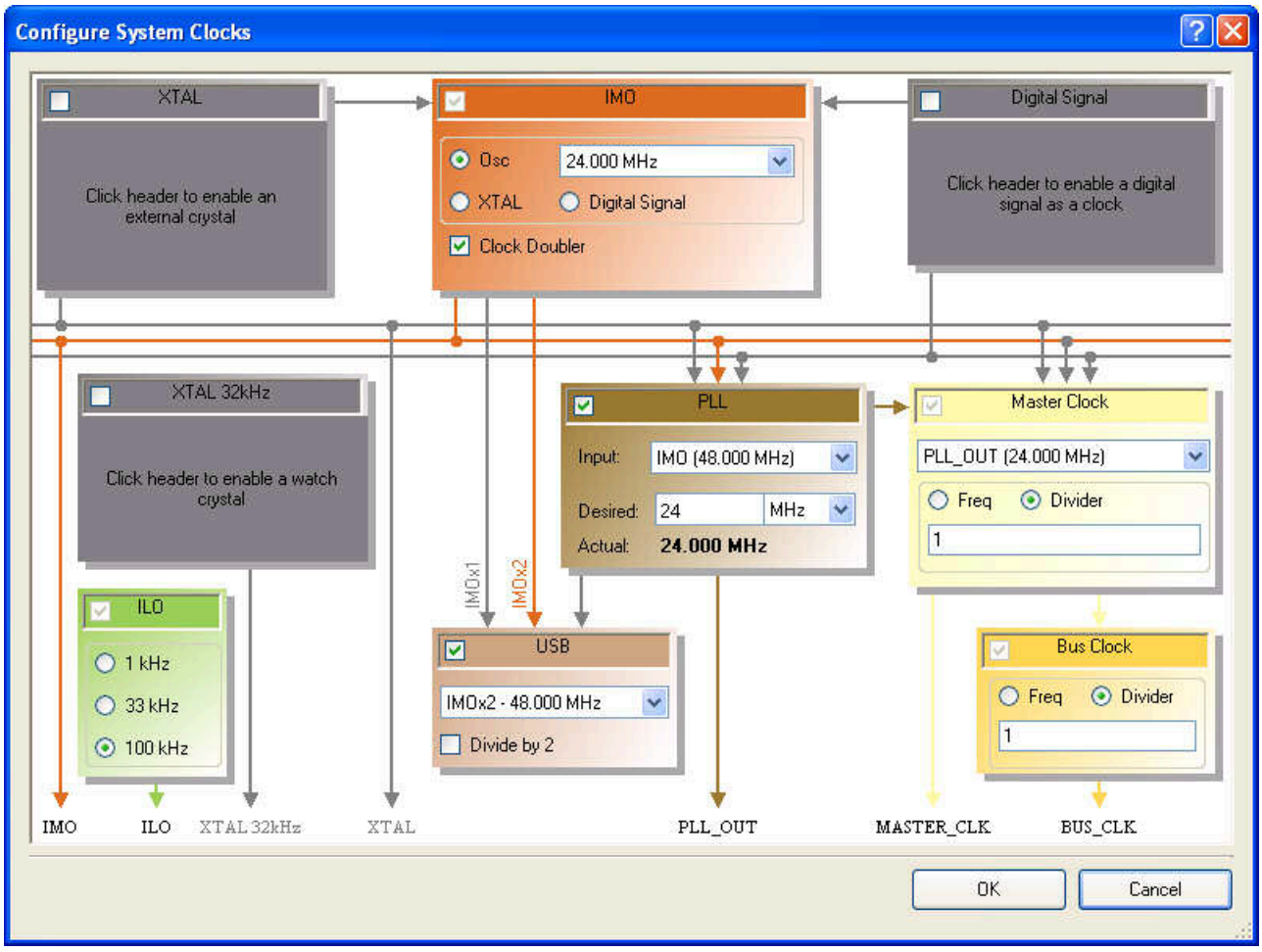
them out of power down mode. It does not change the USB address field of the USBCR register, maintaining the USB address previously assigned by the host.

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.

Clock Settings

The USB hardware block requires a 48 MHz clock to be configured through the PSoC Creator Design-Wide Resource Editor. You must also set the ILO clock to 100 kHz.



DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data.

5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	Ω	
Maximum Clock Rate	---		TBD	MHz	

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.10.b	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
1.10.a	Moved local parameters to formal parameter list.	To address a defect that existed in PSoC Creator v1.0 Beta 4.1 and earlier, the component was updated so that it could continue to be used in newer versions of the tool. This component used local parameters, which are not exposed to the user, to do background calculations on user input. These parameters have been changed to formal parameters which are visible, but un-editable. There are no functional changes to the component but the affected parameters are now visible in the “expression view” of the customizer dialog.

PRELIMINARY