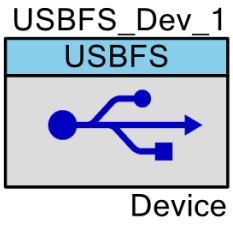
Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

USBFS Device (USBFS_DEV_PDL)

1.0



Features

- USB full-speed device framework
- One bidirectional control endpoint and eight unidirectional data endpoints
- Support for interrupt, control, bulk, and isochronous transfer types
- Crystal-less operation, and integrated USB termination and pull-up resistors
- USB Configurator tool (helps construct descriptor)
- Run-time support for descriptor set selection
- HID, Audio, CDC classes support

General Description

The USBFS_DEV_PDL Component provides a full-speed USB 2.0 Chapter 9 specification compliant device framework. It uses the cy_usbfs low-level driver from the Peripheral Driver Library (PDL) to provide an interface to the USB block hardware and cy_usb_dev middleware that decodes and dispatches requests from the USB host. Additionally, middleware provides support for Audio, CDC and HID classes to simplify USB device implementation. Finally, a stand-alone USB Configurator tool is provided in PDL to make it easy to construct the descriptors. You can construct a generic USB Device, as well as an Audio-, CDC- or HID-based device.

Refer to the USB-IF device class documentation for additional information on descriptors (http://www.usb.org/developers/docs/devclass_docs/).

The USBFS_DEV_PDL Component is a graphical configuration entity built on top of the cy_usbfs driver available in the PDL. It allows schematic-based connections and hardware configuration as defined by the Component Configure dialog. Also, it provides menu item to launch to USB Configurator.

When to Use a USBFS_DEV_PDL Component

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

Definitions

- USB – Universal Serial Bus
- USBFS – USB device supporting the full speed (FS) transfer mode
- CDC – Communication Device Class
- Descriptor – Data structure with a defined format used by USB devices to report their attributes to a USB host
- DMA – Direct Memory Access
- Endpoint – A uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device
- HID – Human Interface Device
- ISR – Interrupt Service Routine
- LPM – Link Power Management
- SOF – Start of Frame
- USB-IF – Universal Serial Bus Implementers Forum

Quick Start

1. Drag a USBFS Device Component from the Component Catalog *Cypress/Communications/USB/USBFS (Device)* folder onto your schematic (placed instance takes the name USBFS_Dev_1).
2. Double-click the dropped Component to open the Configure dialog. You can leave the default configuration; this is enough for the USB Full-Speed device to operate.
3. Right-click the Component in the schematic and select Launch USB Configurator from the context menu to configure USB device descriptors. See Construct USB Descriptors for more information.

   Note USB device descriptors are mandatory for operation.
4. Open the Design-Wide Resources Pin Editor, and assign the Dp and Dm pins for your design.
5. Open the Design-Wide Resources Clock Editor and configure the USB clock (Clk_HF3) to be 48MHz +/- 0.25%. Use guidance provided in the Clock Configuration section.
6. Open the Design-Wide Resources Interrupts Editor, and assign interrupt priorities to High, Medium and Low interrupts. Use guidance provided in the Interrupts Configuration section.

7. Build the project to verify the correctness of your design. This will add the required PDL modules to the Workspace Explorer, generate configuration data, and allocate context for the USBFS_Dev_1 instance.
8. In the *main.c* file, implement USB interrupt handlers, initialize and enable the USB peripheral and start the application:

```
/* Implement Low ISR for USBFS_Dev_1 */
void USBFS_Dev_1_LowIsr(void)
{
    /* Call interrupt processing */
    Cy_USBFS_Dev_Drv_Interrupt(USBFS_Dev_1_HW,
                               Cy_USBFS_Dev_Drv_GetInterruptCauseLo(USBFS_Dev_1_HW),
                               &USBFS_Dev_1_drvContext);
}

/* Implement Medium ISR for USBFS_Dev_1 */
void USBFS_Dev_1_MediumIsr(void)
{
    /* Call interrupt processing */
    Cy_USBFS_Dev_Drv_Interrupt(USBFS_Dev_1_HW,
                               Cy_USBFS_Dev_Drv_GetInterruptCauseMed(USBFS_Dev_1_HW),
                               &USBFS_Dev_1_drvContext);
}

/* Implement High ISR for USBFS_Dev_1 */
void USBFS_Dev_1_HighIsr(void)
{
    /* Call interrupt processing */
    Cy_USBFS_Dev_Drv_Interrupt(USBFS_Dev_1_HW,
                               Cy_USBFS_Dev_Drv_GetInterruptCauseHi(USBFS_Dev_1_HW),
                               &USBFS_Dev_1_drvContext);
}

/* Initialize USB Device 0 (middleware and driver) */
(void) Cy_USB_Dev_Init(USBFS_Dev_1_HW,
                       &USBFS_Dev_1_drvConfig,
                       &USBFS_Dev_1_drvContext,
                       &USBFS_Dev_1_devices[0U],
                       &USBFS_Dev_1_devConfig,
                       &USBFS_Dev_1_devContext);

/*
* Call class initialization functions to add support Audio, CDC or HID classes.
* Example:
  (void) Cy_USB_Dev_HID_Init(&USBFS_Dev_1_hidConfig,
                             &USBFS_Dev_1_hidContext,
                             &USBFS_Dev_1_devContext);
*/

/* Initialize and enable USB interrupts */
(void) Cy_SysInt_Init(&USBFS_Dev_1_LowPriorityInterrupt_cfg,    USBFS_Dev_1_LowIsr);
(void) Cy_SysInt_Init(&USBFS_Dev_1_MediumPriorityInterrupt_cfg, USBFS_Dev_1_MediumIsr);
(void) Cy_SysInt_Init(&USBFS_Dev_1_HighPriorityInterrupt_cfg,   USBFS_Dev_1_HighIsr);

NVIC_EnableIRQ(USBFS_Dev_1_LowPriorityInterrupt_cfg.intrSrc);
NVIC_EnableIRQ(USBFS_Dev_1_MediumPriorityInterrupt_cfg.intrSrc);
NVIC_EnableIRQ(USBFS_Dev_1_HighPriorityInterrupt_cfg.intrSrc);

/* Enable global interrupts */
```

```
__enable_irq();

/* Enable pull-up on D+ line (USB device appears on the bus).
* This function waits until device enumeration is completed.
*/
Cy_USB_Dev_Connect(true, CY_USB_DEV_WAIT_FOREVER, &USBFS_Dev_1_devContext);

/* USB Device is ready for operation after Connect function returns:
* - Enable OUT endpoint to start receiving data from USB Host: Cy_USB_Dev_StartReadEp
* - Load IN endpoint to send data to USB Host: Cy_USB_Dev_WriteEpNonBlocking
*/
```

9. Build and program the device. Observe device is connected to operating system and enumerated successfully.

Note The USB Device requires drivers to operate with your operating system, so you have provide drivers for custom device or use HID or CDC[1] class of device, which does not require drivers.

Input/Output Connections

This section describes the various input and output connections for the USBFS_DEV_PDL Component. An asterisk (*) in the following list indicates that it may not be shown on the Component symbol for the conditions listed in the description of that I/O.

Terminal Name	I/O Type	Description
sof*	Output	The start-of-frame (sof) output allows the device to identify the start of the frame and synchronize internal endpoint clocks to the host. This output is visible if the Enable SOF Output parameter is enabled.

1 Only Windows 10 provides CDC class support for virtual COM port without inf file.

Internal Pins Configuration

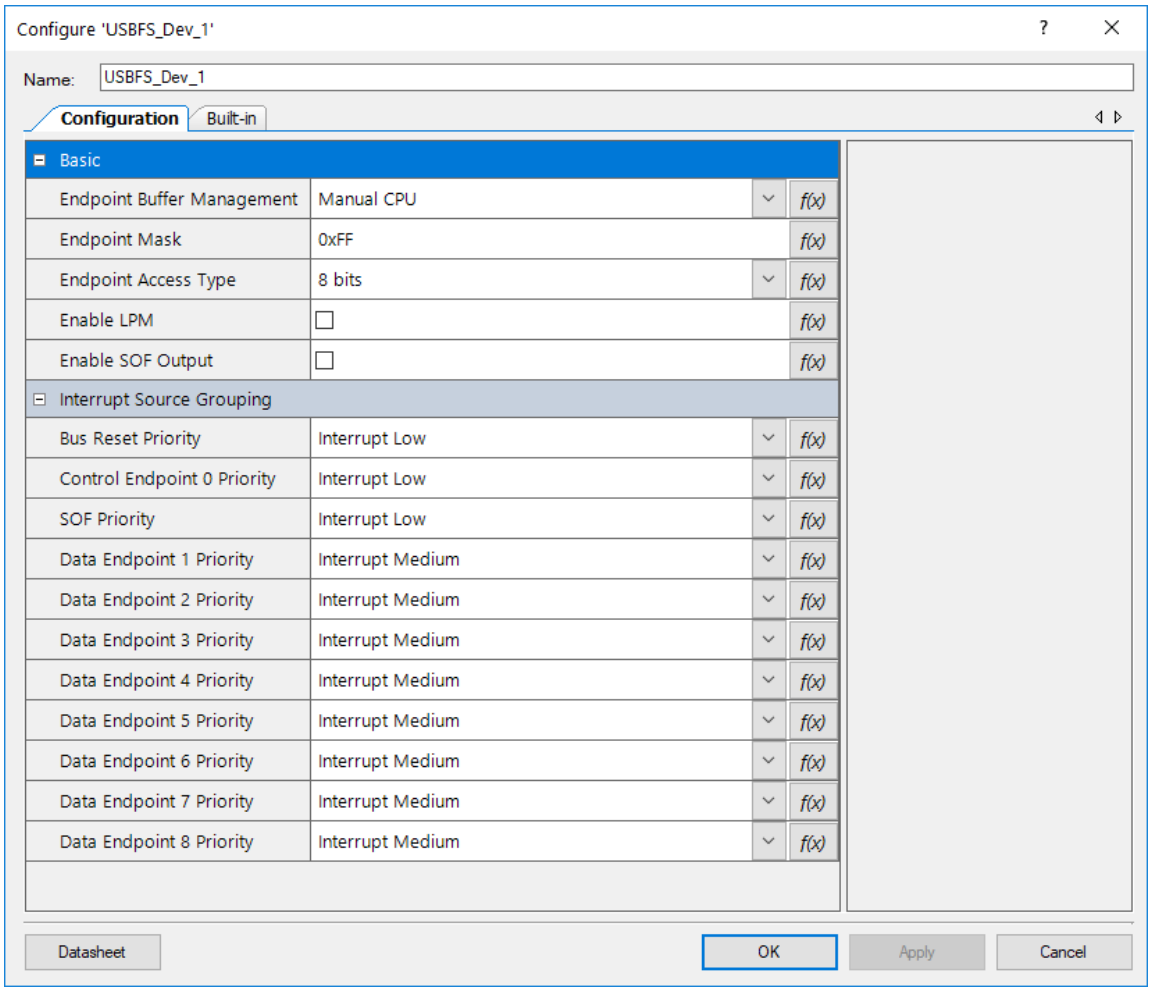
The USB pins are buried inside the Component: USBFS_Dev_1_dp and USBFS_Dev_1_dm. These pins are buried because they use dedicated connections and are not routable as general purpose pins. For more information, refer to the I/O System section in the device Technical Reference Manual (TRM).

Component Parameters

The USBFS_DEV_PDL Component Configure dialog allows you to edit the configuration parameters for the Component instance.

Configuration Tab

This tab contains the Component parameters used in the general peripheral initialization settings.



Parameter Name	Description
Endpoint Buffer Management	The USBFS hardware has dedicated buffer for data endpoint, which has 512 bytes. This buffer is accessed and utilized differently depending on Endpoint Buffer Management parameter: • Manual CPU (Mode 1): The hardware buffer is divided between endpoints so each endpoint has a space in the buffer to store data. *The sum of endpoints buffers is limited by hardware buffer size (512 bytes).* The CPU handles data transfer between SRAM endpoint buffer and the USBFS hardware buffer. • Manual DMA (Mode 2): The hardware buffer is divided between endpoints so each endpoint has a space in the buffer to store data. *The sum of endpoints buffers is limited by hardware buffer size (512 bytes).* The CPU triggers DMA request to transfer data between SRAM endpoint buffer and the USBFS hardware buffer. • Automatic DMA (Mode 3): the USBFS hardware buffer acts as a FIFO and SRAM buffer must be provided to be used instead of endpoint hardware buffer. The endpoint buffer is assigned to the DMA channel, which is controlled by the hardware. When transfer starts, it issues DMA requests to transfer data between endpoint buffer and hardware buffer until transfer completion.
Endpoint Mask	8-bit mask that defines which endpoints are utilized by the USB Device (bit 0 – endpoint 1, bit 1 – endpoint 2 and so on up to endpoint 8). The value of the mask is defined by USB Device descriptors. **Note** Mask must enable endpoints that is enabled in the USB Device descriptors. Otherwise the compilation error is generated.
Endpoint Access Type	Defines how the USBFS hardware buffer is accessed – 8-bit or 16-bit. The 16-bits access improves the performance. **Note** The extra byte is allocated in the USBFS hardware buffer for each odd sized endpoint when Endpoint Buffer Management mode is Manual CPU or Manual DMA.
Enable SOF Output	Exposes the Start-of-Frame (SOF) output on the USBFS symbol. The SOF output allows the device to identify the start of the frame and synchronize application events with it. The host issues SOF packets at a nominal rate of once every 1 ms for a full-speed-bus. The host stops sending SOF packets for more than 3 ms to suspend the USB device.
Arbiter Priority	Priority of the Arbiter. This interrupt source triggers when one of the following conditions for any endpoint are met: IN endpoint buffer full, endpoint DMA grant, endpoint buffer overflow, endpoint buffer underflow, endpoint error in transaction, endpoint DMA terminated. It is available only when Endpoint buffer Management uses DMA.
Bus Reset Priority	Priority of the Bus Reset interrupt. This interrupt source triggers when an USB bus reset event occurs.
Control Endpoint 0 Priority	Priority of the Control Endpoint 0 interrupt. This interrupt source triggers whenever the host tries to communicate over the control endpoint.
LPM Priority	Priority of the LPM interrupt. This interrupt source triggers when an LPM entry USB extension packet is received.
SOF Priority	Priority of the SOF interrupt. This interrupt source triggers when a start of frame is received. To enable this interrupt source, the Enable SOF interrupt option has to be checked.
Data Endpoint 1-8 Priority	Priority of the Data Endpoints interrupt (Each endpoint has its own parameter). These interrupt sources trigger whenever the host completes communication over the corresponding data endpoint. They are available only when an endpoint is utilized by the device.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the Component using software.

By default, PSoC Creator assigns the instance name USBFS_Dev_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.

This Component uses the cy_usbfs driver and cy_usb_dev middleware from the PDL. The driver is copied into the “pdl\drivers\peripheral\usbfs\” directory and the middleware is copied into the “pdl\middleware\usb_dev\” directory of the application project after a successful build.

Refer to the PDL documentation for a detailed description of the complete API. To access this document, right-click on the Component symbol on the schematic and choose the “**Open Driver PDL Documentation…**” or “**Open Middleware PDL Documentation…**” option in the drop-down menu.

The Component generates the configuration structures and base address described in the Global Variables and Preprocessor Macros sections. Pass the generated data structures and the base address to the associated cy_usb_dev middleware function in the application initialization code to configure the peripheral. Once the peripheral is initialized, the application code can perform run-time changes by referencing the provided base address in the driver API functions.

Global Variables

The USBFS_DEV_PDL Component populates the following peripheral initialization data structure(s). The generated code is placed in C source and header files that are named after the instance of the Component (e.g., *USBFS_Dev_1.c*). Each variable is also prefixed with the instance name of the Component.

cy_stc_usbfs_dev_drv_config_t const USBFS_Dev_1_drvConfig

The instance-specific driver configuration structure. The pointer to this structure should be passed to Cy_USB_Dev_Init function (that is a part of USBFS_Dev_1_Start function) to initialize driver with Component GUI selected settings.

cy_stc_usbfs_dev_drv_context_t USBFS_Dev_1_drvContext

The instance-specific driver context structure. It is used for internal configuration and data keeping for the USBFS Device driver. Do not modify anything in this structure.

cy_stc_usb_dev_context_t USBFS_ Dev_1_devContext

The instance-specific middleware context structure. It is used for internal configuration and data keeping for the USB Device middleware. Do not modify anything in this structure.

cy_stc_usb_dev_audio_context_t USBFS_Dev_1_audioContext

The instance-specific Audio middleware context structure. It is used for internal configuration and data keeping for the USB Device Audio Class. Do not modify anything in this structure.

cy_stc_usb_dev_cdc_context_t USBFS_Dev_1_cdcContext

The instance-specific CDC middleware context structure. It is used for internal configuration and data keeping for the USB Device CDC Class. Do not modify anything in this structure.

cy_stc_usb_dev_hid_context_t USBFS_Dev_1_hidContext

The instance-specific HID middleware context structure. It is used for internal configuration and data keeping for the USB Device HID Class. Do not modify anything in this structure

The USB Configurator generates following configuration structures for the middleware (find declarations in the *USBFS_Dev_1_cfg.h)*.

cy_stc_usbfs_dev_config_t const USBFS_Dev_1_devConfig

The instance-specific middlware device configuration structure. The pointer to this structure should be passed to Cy_USB_Dev_Init function to initialize middleware.

cy_stc_usb_dev_cdc_config_t const USBFS_Dev_1_cdcConfig

The instance-specific CDC middleware configuration structure. The pointer to this structure should be passed to Cy_USB_Dev_CDC_Init function to initialize CDC class support in the middleware.

cy_stc_usb_dev_hid_context_t const USBFS_Dev_1_hidConfig

The instance-specific HID middleware configuration structure. The pointer to this structure should be passed to Cy_USB_Dev_HID_Init function to initialize HID class support in the middleware

Preprocessor Macros

The USBFS_DEV_PDL Component generates the following preprocessor macro(s). Note that each macro is prefixed with the instance name of the Component (e.g. “USBFS_Dev_1”).

#define USBFS_Dev_1_HW ((USBFS_Type *) USBFS_Dev_1_USBFS__HW)

The pointer to the base address of the USBFS instance.

Data in RAM

The generated driver configuration structure may be placed in flash memory (const) or RAM. The former is the more memory-efficient choice if you do not wish to modify the configuration data at run-time. Under the **Built-In** tab of the Configure dialog, set the parameter Config Data in Flash to make your selection. The default option is to place the data in flash.

Interrupt Service Routine

The interrupt service routine (ISR) is mandatory for the USBFS_DEV_PDL Component operation; therefore, an Interrupt Components are placed inside it. The cy_usbfs driver function Cy_USBFS_Dev_Drv_Interrupt implements the ISR functionality. You must call the Cy_USBFS_Dev_Drv_Interrupt function inside the ISR and enable the interrupt controller to trigger the corresponding interrupt. Refer to the code example in the Quick Start section.

Code Examples and Application Notes

This section lists the projects that demonstrate the use of the Component.

Code Examples

PSoC Creator provides access to code examples in the Find Code Example dialog. For Component-specific examples, open the dialog from the Component Catalog or an instance of the Component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Code Example" topic in the PSoC Creator Help for more information.

There are also numerous code examples that include schematics and example code available online at the Cypress Code Examples web page.

Application Notes

Cypress provides a number of application notes describing how PSoC can be integrated into your design. You can access the Cypress Application Notes search web page at www.cypress.com/appnotes.

Functional Description

Clock Configuration

The USBFS_DEV_PDL Component has stringent clock requirements. The following conditions must be met for a valid USB Device design:

- The USB clock (Clk_HF3) must be 48 MHz.
- The accuracy of the USB clock must be within +/-0.25%.

To meet these conditions, navigate to the Design-Wide Resources Clock Editor. You may also drag and drop a USBFS_DEV_PDL Component from the Component Catalog and then click on the error in the Notice List window. This will open the System Clock Editor.

Perform the following changes to use the internal PSoC clocks to drive the USB. Alternatively, use an external clock that satisfies the above conditions.

- Configure IMO or ECO to be source to the PLL.

  Note If IMO is used as source for PLL set the “Trim with” parameter in the IMO box as USB to meet accuracy requirements.

- Configure PLL output frequency 48MHz or frequency which multiple of 48.

- Configure Clk_HF3 source to be PLL and provide USB Clock 48 MHz.

  Note The Clk_HF3 has a integer divider so input clock can be a multiple of 48

- Make sure that USB clock meets accuracy requirements +/-0.25%.

Once the clock configuration meets the requirements, no warnings or errors should appear in the Notice List window.

For more information about USB clock configuration, refer to the USB Device mode chapter in the device TRM.

Interrupts Configuration

The USB block hardware provides three interrupts: Low, Medium and High and 13 interrupt sources which are mapped to them. The Component configuration dialog allows you to assign each interrupt source to trigger one of the interrupts. This allows you to assign different priorities for interrupt sources handling. The Component does not configure the interrupt's priority. The interrupt priority is configured in the interrupt controller (NVIC) and this is application specific. To do that, open the Design-Wide Resources Interrupts Editor and configure the interrupt priority for the utilized core. **By default, all Component interrupts have the same priority**.

The recommended (default) interrupt source mapping is:

1. Interrupt Low: Control Endpoint 0, Bus Reset and SOF interrupt sources
2. Interrupt Medium: Data Endpoint 1-8 interrupt sources.
3. Interrupt High: the LMP and Arbiter interrupt sources.

However, the final configuration must be defined by the application.

Typically, the interrupt priorities are assigned accordingly to interrupt names: High interrupt has the greater priority than Low and Medium interrupts, the Medium interrupt has the greater priority than Low interrupt and the Low interrupt has the lowest priority among the USBFS interrupts. For example, configure ARM CM4 priorities: High Interrupt – Priority 1, Medium Interrupt – Priority 2, Low Interrupt – Priority 3.

Use this list of interrupt priority requirements for proper operation:

1. Manual DMA mode (Mode 2): the Arbiter interrupt source must be assigned to the interrupt with a priority that is higher than the interrupt triggered by Data Endpoint 1-8 interrupt sources.
2. Automatic DMA mode (Mode 3): follow the same the rule as above.

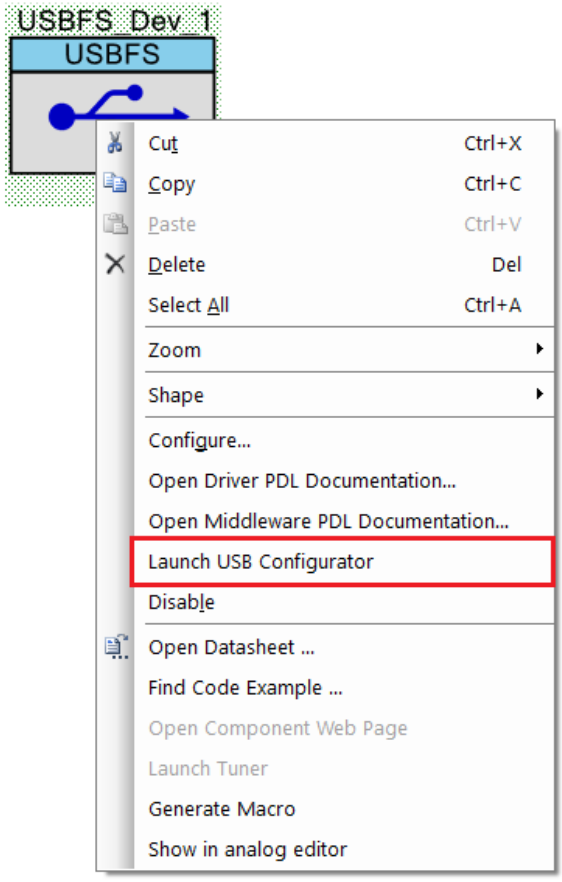
Low Power Mode

The USBFS_DEV_PDL Component does not provide callback functions to handle low power mode transition. This is the user's responsibility to implement low power callbacks using the cy_usbfs driver API and application processing to archive the desired behavior. The cy_usbfs

supports USB suspend, resume, and remote wakeup functionality. Refer to the driver documentation Low Power Mode section for more information.

Construct USB Descriptors

The USB descriptors are mandatory for USB Device operation. The PDL provides a USB Configurator to configure USB Device descriptors and then generates source and header files. These files keep device descriptors and helper structures required for USB Device operation. To run the configurator, right-click the Component in the schematic and select Launch USB Configurator from the context menu.



The USB Configurator provides the interface templates for Audio and CDC Classes and a number of HID Report Descriptors. Use these templates to simplify the process of USB Device descriptors construction. Refer to the USB Configurator User Guide for the list of supported descriptors.

Detailed information about USB Descriptors is provided by the USB Specification.

Detailed descriptions about Audio, CDC and HID class is provided by the USB Implementers Forum (USB-IF) Class Documentation.

Note After Component instance name is changed you must run USB Configurator and press “Save” button to pass new instance name. Otherwise, the project will generate configuration files using old instance name what results compilation error.

Industry Standards

USB specification

The USBFS_DEV_PDL Component is implemented on top of the USB fixed-function block. It supports full-speed communication (12 Mbps) and is designed to be compliant with the USB Specification Revision 2.0. For details about the USB specification, see the USB Implementers Forum website.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Refer to PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6) for information on MISRA compliance and deviations for files generated by PSoC Creator.

The USBFS_DEV_PDL Component has the following specific deviations:

MISRA-C: 2004 Rule	Rule Class [2]	Rule Description	Description of Deviation(s)
10.1	R	The value of an expression of integer type shall not be implicitly converted to a different underlying type if: a) it is not a conversion to a wider integer type of the same signedness, or b) the expression is complex, or c) the expression is not constant and is a function argument, or d) the expression is not constant and is a return expression.	The numeric constants used for initialization of unsigned integer values miss “U” what causes integer type conversion. This operation is safe because all constants are positive numbers.
13.2	A	Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.	The macro CY_USBFS_DEV_DRV_ALLOC_ENDPOINT_BUFFER does not test against zero construction (size & 0x1U). The result of operation is 0 or 1 therefore it is safe to use it as boolean in the condition operation.

2 Required / Advisory

This Component uses firmware driver cy_usbfs and cy_usb_dev middleware from PDL module. Refer to the PDL documentation for information on their MISRA compliance and specific deviations.

This Component has the following embedded Components: Clock, Pins, Interrupt and DMA. Refer to the corresponding Component datasheets for information on their MISRA compliance and specific deviations.

Registers

Refer to the Universal Serial Bus (USB) Device Mode Registers section in the device TRM.

Resources

The USBFS_DEV_PDL Component uses a single USB block configured for Device operation.

DC and AC Electrical Characteristics

Characterization data for PSoC 6 devices is available in the applicable device datasheet. These documents are located on the Cypress website http://www.cypress.com/products/32-bit-arm-cortex-m4-psoc-6.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.0.a	Updated the datasheet for the MISRA section.	Added specific deviations.
1.0	Initial Version	

© Cypress Semiconductor Corporation, 2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical Components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical Component is any Component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.