USB Vendor Commands with PSoC® 3 and PSoC 5

AN56377

Authors: Brad Haber, Hridya Valsaraju
Associated Project: Yes
Associated Part Family: CY8CXXXX
Software Version: PSoC® Creator™
Associated Application Notes: None

Abstract

The USB interface on PSoC® 3 / PSoC 5 can be used to control PSoC 3 and PSoC 5 based products or monitors fast changing variables, both from a host computer. The example project discussed in this application note implements these functions using USB vendor requests through the USBFS component of PSoC 3 / PSoC 5. This provides the ability to easily log and display data obtained by manipulating the other components in a design.

Introduction

The USB specification describes three types of requests to communicate to a device: Standard, Class, and Vendor Specific. All devices must support standard requests, and may optionally support Class or Vendor requests. This application note uses the Vendor class requests to communicate with and exercise PSoC 3 / PSoC 5. For additional information on USB requests, refer to the USB specification available from the USB-IF.

Cypress offers two other components that are used in this example. Unlike a class device, a vendor specific device requires the developer to provide a device driver. This example uses a generic USB driver, *cyusb.sys*, and a .NET managed DLL, *CyUSB.DLL* for writing host applications. You can find both the driver and *.DLL* files included in SuiteUSB 3.4 which can be downloaded for free from the Cypress website www.cypress.com.

This example uses the following items:

- PSoC® Creator™
- PSoC Programmer
- PSoC Development Kit (CY8CKIT-001)
- Cypress SuiteUSB 3.4

Configure Components using PSoC Creator

This section discusses the implementation of two vendor commands that are defined to perform the following actions:

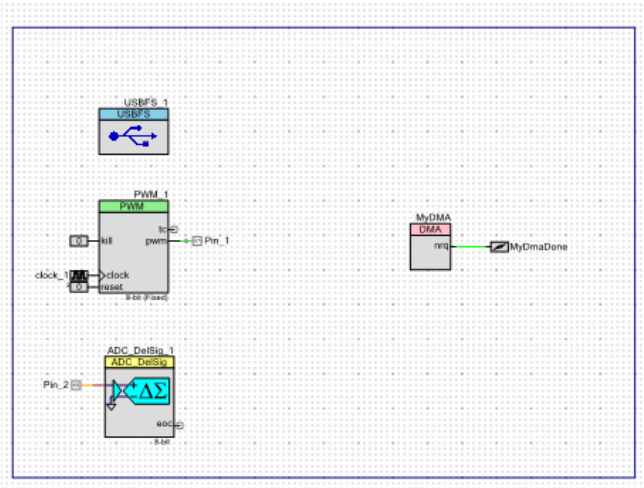
1. Change the duty cycle of a PWM wave to a value input by the user on a host computer GUI application.
2. Initiate streaming data to the host computer from any PSoC 3 / PSoC 5 register whose address is selected by the user through the GUI. This implementation helps to monitor any variable at high speed during run time. This is done by using the host computer connected to a PSoC 3 based product.

PSoC Creator is used to configure the PSoC 3 or PSoC 5 device with the components used in this example. A total of nine components are used to build this project. Create a new project in PSoC Creator named “PSoC 3 USB Control” (or any other name you prefer). Add the following components to the Top Schematic:

- USBFS
- PWM
- ADC
- Digital Port to use with PWM
- Analog Port to use with ADC
- DMA
- Interrupt to use with DMA
- Control register to use with DMA
- Logic Low to use with PWM

After adding these components your schematic must be similar to Figure 1.

Figure 1. Schematic View after Adding Required Components



- The **USBFS** component is used for communications between PSoC 3 / PSoC 5 and the host computer.
- The **DMA** component is used to dump data into the USB Data Register from the user specified register.
- The **PWM** component provides you a configurable duty cycle.
- The **ADC** is used for easy demonstration of ‘stream from register’ functionality.
- The **Interrupt** is triggered when each DMA transaction is completed.
- The **Control register** is used to trigger the DMA component.
- The **Analog port** pin feeds input from a potentiometer on the development board.
- The **Clock** is used for the PWM component.
- The **Digital port** is used to connect to the output of the PWM.

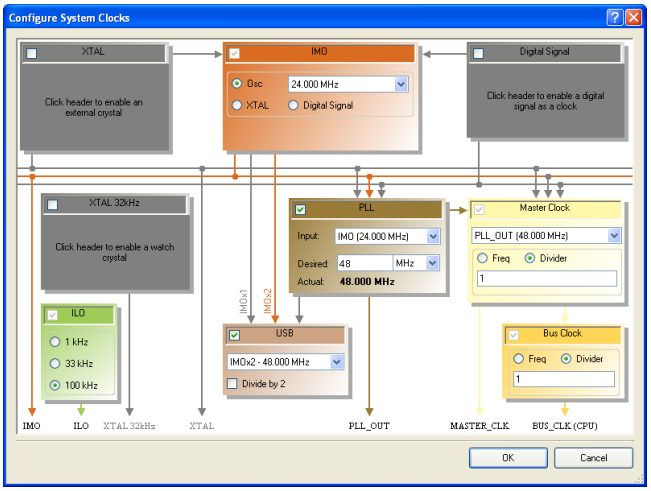
Configuring the PSoC Components

Prior to configuring the individual components, configure the clock settings for this example. The clocks are configured selecting the **.cydwr* file in the Solution Explorer window of the project. After the *.cydwr* file is opened, select the 'Clocks' tab and then the 'Edit clock' icon. This opens the Clock Configuration window. Configure the clocks to the following settings:

- IMO
  - 24.000 MHz
- PLL
  - IMO (48.000 MHz)
  - Desired (48 MHz)
- USB
  - Clock Enable block checked
  - IMOx2 – 48.000 MHz
- Master Clock
  - PLL Out (48.000 MHz)
  - Divider 1
- ILO: 100 kHz

The clock configuration window must be similar to Figure 2.

Figure 2. Clock Configuration Window

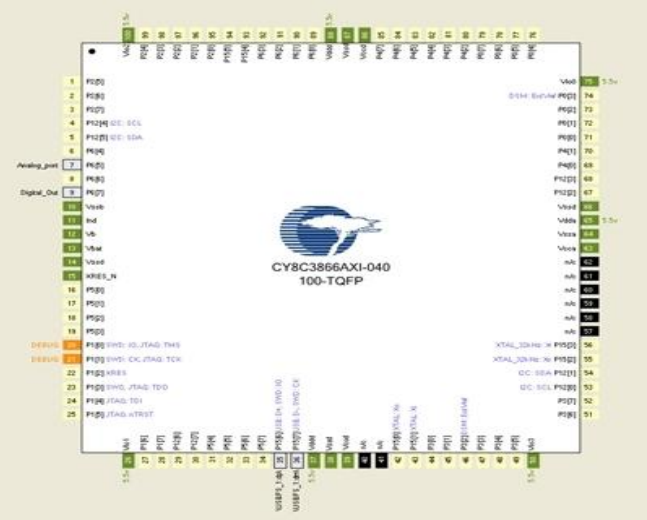


Now configure the individual components that were previously added to this design. To configure the components in PSoC Creator, first select the component on the schematic, then right click and select "Configure" from the available options.

PSoC 3 Pin Assignments

In the .cywdr PSoC 3 configuration window, route the digital and analog ports to specific pins. This is not necessary, because PSoC Creator does this automatically when the application is built. However, this shows that the USBFS pins are automatically routed and only the port pins remain unassigned. For this example, the digital port is routed to P6[7] and the analog port to P6[5], as shown in Figure 3.

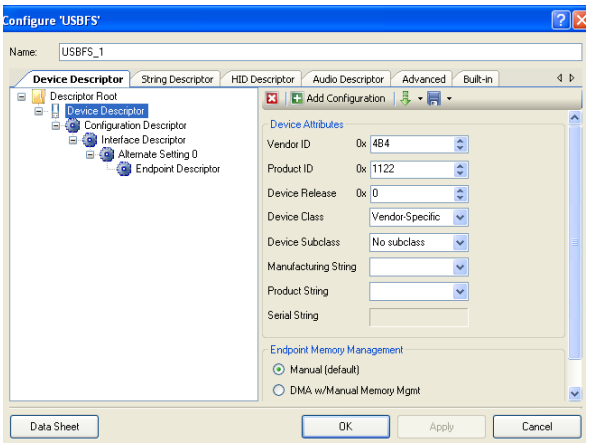
Figure 3. Pin Assignments



Configuring the USBFS Component

In this example, the USB component is configured to report itself to the host computer as a vendor specific peripheral device. PSoC Creator generates most of what is necessary to communicate over USB, simplifying what you need to do for this design. After selecting “Configure” (as discussed in the previous section), a new window opens. This window enables you to customize the USB component descriptor configuration for this application. Select 'Device Descriptor' to begin customizing the device attributes.

Figure 4. USBFS Configuration Window



- Vendor ID
  - For this application, leave the Vendor ID to the default value of 0x4b4. This Vendor ID is assigned to Cypress by the USB Implementers Forum. For your product, you must obtain your own unique Vendor ID from the USB Implementers Forum.
- Product ID
  - The entry for this field is determined by the vendors to suit their own product identification scheme. For this example, the default value of 0x8051 is changed to a new Product ID value of 0x1122.

The Vendor ID and Product ID fields are reported to the host computer when PSoC 3 is connected to the USB. These fields are used to ensure that the correct driver is loaded and that the host computer application used recognizes this example.

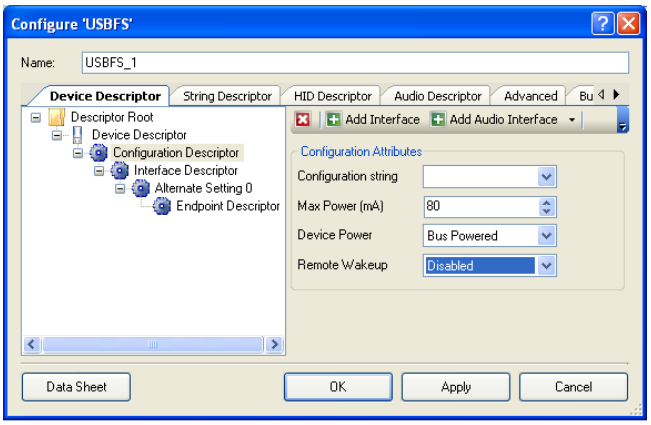
- Device Release
  - The entry for this field is determined by the vendors to suit their own product identification scheme. A typical use of this entry is a zero based value to identify revisions of a product. As this is the first product for this application note, this value is set to 0.
- Device Class
  - This field enables you to define the USB peripheral as either one of the approved USB Class devices or as a unique Vendor Specific device. In this application note, the USB interface is defined as a vendor specific device. Use the drop down menu to select “Vendor-Specific”.
- Device Subclass
  - The USB peripheral is defined as a Vendor specific device, so this field must be left as “No subclass”.
- Manufacturing String
  - You can enter your company's name in this field. For this application note, “Cypress” is entered.
- Product String
  - You can enter the Product name in this field. For this application note, “PSoC 3 USB Control” is entered.
- Serial Number String
  - You can enter the Product serial number in this field. For this application note, this field is left blank.
- Endpoint Memory Management
  - Select “Manual” radio button.

Press the **Apply** button to save the configuration settings defined earlier. The Device Descriptor tab must be similar to Figure 4 on page 3. Now select the Configuration Descriptor to continue describing the USB interface.

- Configuration String
  - This field can be left blank, because there is only one configuration. This is described in the Device Descriptor strings.
- Max Power
  - A bus powered USB interface is defined. The USB specification defines a device as either a low power device requiring a maximum of one 100 mA unit of power, or as a high power device requiring five units of power. In this application, a full speed USB interface is defined and it fits the low power definition. This device can draw a maximum of 100 mA. Your product may require less than one unit load and you can declare the actual maximum mA value used.
- Device Power
  - A USB peripheral is defined as being either bus or self powered. For this application note, the peripheral is declared as a bus powered device. Additional USB compliance requirements are needed for a self powered device. However, these requirements are beyond the scope of this application note. For additional information on self-powered device USB compliance requirements, refer to the USBFS component data sheet and the application note AN15813 “Monitoring the EZ-USB FX2LP™ VBUS” at www.cypress.com.
- Remote Wakeup
  - The USB Control application need not wake a host computer from a stand-by condition. This field is left as 'Disabled'.

Press the Apply button to save the configuration settings defined earlier. The Configuration Descriptor tab must be similar to Figure 5.

Figure 5. USBFS Configuration Window

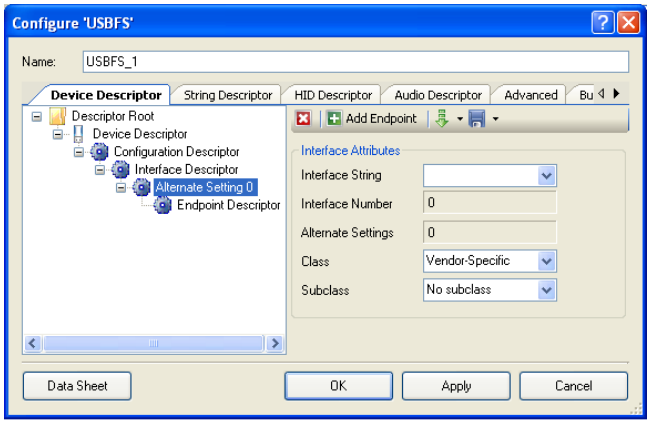


This example has only one interface, so there is nothing to configure in the Interface Descriptor section. Select Alternate Setting 0 to continue defining the USB interface.

- Interface String
  - This field can be left blank, because only one interface is described for the USB interface.
- Interface Number
  - This field is filled in by PSoC Creator based on the previous entries made in the ‘Configure USBFS’ window. .
- Alternate Settings
  - This field is filled in by PSoC Creator based on the previous entries made in the ‘Configure USBFS’ window.Class
  - Use the drop down menu and select “Vendor-Specific”.
- Subclass
  - Because the USB device we create belongs to a vendor specific class, the Subclass field can be left in its default setting of “No subclass”.

Press the **Apply** button to save the settings defined earlier. The Alternate Setting 0 tab must be similar to Figure 6.

Figure 6. Alternate Setting 0 Window

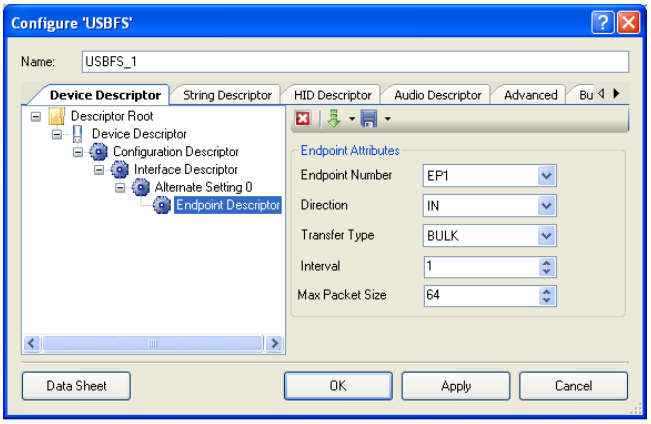


This section defines the endpoint that this example uses to send data to the host computer. Click on Endpoint Descriptor to begin.

- Endpoint Number
  - Select an endpoint from the drop down menu. For this example, “EP1” is selected.
- Direction
  - USB is a host centric interface. All transfers are considered from the host computer perspective. Hence, we define the direction of the endpoint as “IN” because we stream data from the PSoC3 USB device into the host computer.
- Transfer Type
  - USB offers four types of transfers: Control, Bulk, Isochronous, and Interrupt. Bulk transaction types are characterized by the ability to guarantee error-free delivery of data between the host and a function. This is done by error detection and retry. Bulk transfers are used for this example.
- Interval
  - For a full speed bulk IN endpoint, such as that defined for this application, the Interval field does not apply. It may be safely left with its default value. Additional information on this field is available in Table 9-13 of the USB 2.0 specification.

The Endpoint Descriptor window is similar to Figure 7.

Figure 7. Endpoint Configuration Window



Configuring the PWM Component

After opening the configuration window as described earlier, select the following settings for the PWM component. Leave the other settings with their default settings:

- Resolution: 8-bit
- PWM Mode: One output
- Period: 255
- CMP Value 1: 127
- CMP Type 1: Less
- Kill Mode: Disabled (Advanced tab)
- Interrupts: None (Advanced tab)

The note next to the Period field can be safely ignored at this point.

Configuring the DMA Component

After opening the configuration window as described earlier, select the following settings for the DMA component:

- hw_request_enabled: false
- hw_termination_enabled: false
- num_tds: 1

Configuring the ADC Component

After opening the configuration window as described earlier, select the required ADC settings. For example, in this project, the following settings are used.

- Power: High power
- Conversion: Continuous
- Resolution: 8
- Conversion Rate: 10000
- Input Range: Vssa to Vdda (Single Ended)
- Input Buffer Gain: 1
- Reference: Internal Reference

Configuring the Digital Port Component

After opening the configuration window as described earlier, select the following settings for the Digital Port component:

- Set the port direction to PortDirection_Output.
- Set the access mode to PortAccessMode_HW
- Set Pin Mode (Pins) to CMOS

The clock placed next to the PWM is used with its default setting of 24 MHz.

Now use the wiring tool to connect the components as shown in Figure 1 on page 2.

Note that the USBFS component is a software component, so no wiring tool connections are necessary for it. We are not ready to build the project to generate the APIs. Select 'Build' from the top menu and then 'Build USBControl'.

Customizing PSoC Creator Generated Source Code Files

PSoC Creator automatically generates the source code file needed for this example project. You only need to add the custom code to the appropriate files. After the project is built, you begin to add the code to finish the project in the *main.c* and *USBFS_1_vnd.c* files.

Customizing USBFS_1_vnd.c

Place your custom vendor specific code in this file. As mentioned earlier, this section describes the vendor commands that enable PSoC 3 to carry out specific tasks.

Three vendor commands are defined: one to change the duty cycle of the PWM, one to start a streaming read for a register, and the last vendor command stops the data streaming. Each vendor request is defined under a case statement. If there is no data to be returned to the host in the control transfer, then the firmware calls the USBFS_1_InitNoDataControlTransfer () function before the 'break' statement. This provides an acknowledgement to the host for the control transfer. If there is a data stage in the control transfer, then the firmware calls either USBFS_1_InitControlWrite() or USBFS_1_InitControlRad() as necessary.

Here are some vendor specific declarations:

```
/***************************************
*********************************
* Vendor Specific Declarations
***************************************
*********************************/
/* `#START VENDOR_SPECIFIC_DECLARATIONS`
Place your declaration here */
#include "device.h"
#include "CyDmac.h"
#define StartStream 0xA1
#define ChangeDC 0xB1
#define StopStream 0xE1

extern int STARTbit;

extern uint8 MYTD;
extern T_USBFS_1_TD CurrentTD;

extern int MyChannel;

extern uint8 FINISHED;

uint8 pDC, CmpVal;

uint16 RegAddHi = 0x4000;
uint16 RegAddLo;
uint8
USBFS_1_InitNoDataControlTransfer(void);
/* `#END` */
```

Now add the custom vendor command code to the USBFS_1_HandleVendorRqst() function:

```
uint8 USBFS_1_HandleVendorRqst(void)
{
uint8 requestHandled = USBFS_1_FALSE;

if ((CY_GET_REG8(USBFS_1_bmRequestType) &
USBFS_1_RQST_DIR_MASK) ==
USBFS_1_RQST_DIR_D2H)
{
/* Control Read */
```

```
switch (CY_GET_REG8(USBFS_1_bRequest))
{
       case
       USBFS_1_GET_EXTENDED_CONFIG_DESCRIPTO
       R:
       #if
       defined(USBFS_1_ENABLE_MSOS_STRING)
                            currentTD.pData =
       &USBFS_1_MSOS_CONFIGURATION_DESCR[0u]
       ;
                            currentTD.count =
       USBFS_1_MSOS_CONFIGURATION_DESCR[0u];
                            requestHandled  =
       USBFS_1_InitControlRead();
                        #endif /* End
       USBFS_1_ENABLE_MSOS_STRING */
                 break;
        default:
                 break;
         }
    }

    /* `#START VENDOR_SPECIFIC_CODE` Place
your vendor specific request here */

switch (CY_GET_REG8(USBFS_1_bRequest))
{
case StartStream:

RegAddHi = CY_GET_REG16(USBFS_1_wValue); //
Read the upper 16 bits from the wValue field
of setup packet

RegAddLo = CY_GET_REG16(USBFS_1_wIndex);
//Read the lower 16 bits from the wIndex
field of setup packet

#if (defined(__C51__)) /* PSoC3
CyDmaTdSetAddress(MYTD, RegAddLo,
CYDEV_USB_ARB_RW1_DR);
#else/* PSoC5

MyChannel = MyDMA_DmaInitialize(1, 0,
RegAddHi, HI16(USBFS_1_ARB_RW1_DR));

CyDmaTdSetAddress(MYTD,
RegAddLo,LO16(CYDEV_USB_ARB_RW1_DR));
/* Associate the TD with the channel. */
CyDmaChSetInitialTd(MyChannel, MYTD);
//enable the channel
CyDmaChEnable(MyChannel, 1);

#endif
requestHandled =
USBFS_1_InitNoDataControlTransfer();
STARTbit =1;
break;

case ChangeDC:
pDC = CY_GET_REG8(USBFS_1_wValueLo); //Read
the desired dutycycle value from wValue
field
CmpVal = pDC*0.01*255;
PWM_1_WriteCompare(CmpVal);
requestHandled =
USBFS_1_InitNoDataControlTransfer();
break;

case StopStream:

STARTbit=0;                          //stop
streaming
requestHandled =
USBFS_1_InitNoDataControlTransfer();

break;
       }

/* `#END` */

return(requestHandled);
}
```

Customizing isr_1.c

The interrupt MyDMADone gets asserted when each DMA transfer is over. When each packet of 64 bytes is filled in the EP1 buffer by DMA, this interrupt is triggered and the variable FINISHED is set to “1”.
When each packet of 64 bytes is filled in the EP1 buffer by DMA, this interrupt is triggered and the variable FINISHED is set to “1”.

```
/*****************************************
**********************************
*  Place your includes, defines and code
here
*****************************************
**********************************/
/* `#START MyDmaDone_intc` */
extern uint8 FINISHED;
/* `#END` */
/*****************************************
**********************************
* Function Name: MyDmaDone_Interrupt
*****************************************
**********************************
* Summary:
*   The default Interrupt Service Routine
for MyDmaDone.
*
*   Add custom code between the coments to
keep the next version of this file
*   from over writting your code.
*
* Parameters:
*
* Return:
*   void.
*
*****************************************
**********************************/
CY_ISR(MyDmaDone_Interrupt)
{
    /*  Place your Interrupt code here. */
    /* `#START MyDmaDone_Interrupt` */
       FINISHED=1;
    /* `#END` */
```

Customizing main.c

Customize the *main.c* file by adding the code needed to implement this design. Code is added to load the IN endpoint with data specified from the user-defined register, the definitions of the various registers used are present in the PSoC 3 / PSoC 5 Register TRM. A flow chart for the loading of IN endpoint is present in Figure 26.4 of the PSoC 3 TRM. The customized *main.c* file is shown in the following code:

```
#include <device.h>

uint8 MyChannel;
uint8 MYTD;
int STARTbit = 0;

uint8 FINISHED = 0 ;

extern uint16 RegAddHi;
extern T_USBFS_1_EP_CTL_BLOCK USBFS_1_EP[];

void main()
{

CYGlobalIntEnable; /*Enable Global
Interrupts*/

PWM_1_Start();

ADC_DelSig_1_Start();
ADC_DelSig_1_StartConvert();
ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_WA
IT_FOR_RESULT);

USBFS_1_Start(0, USBFS_1_3V_OPERATION);

while(!USBFS_1_bGetConfiguration());

MyDmaDone_Start();
#if (defined(__C51__))
MyChannel = MyDMA_DmaInitialize(1,
                                0,
                                0,
                                0);
#else
/* PSoC 5 */
MyChannel = MyDMA_DmaInitialize(1, 0,
RegAddHi, HI16(CYDEV_USB_ARB_RW1_DR));
#endif

MYTD = CyDmaTdAllocate();  /* Get a TD*/

CyDmaTdSetConfiguration(MYTD,
                        64,
                       DMA_INVALID_TD,
                       MyDMA__TD_TERMOUT_EN
);
/* Set TD to transfer 64 bytes */

/* Associate the TD with the channel. */
CyDmaChSetInitialTd(MyChannel, MYTD);

/* Enable the channel */
CyDmaChEnable(MyChannel, 1);

while(1)
{
if(STARTbit == 1)
{

/*We are implementing the transfer in
register level since the endpoint data
register is configured as the DMA
destination. We directly use the APIs (which
are simpler to use) if we leave out the DMA
in between. Refer to the PSoC3 Register TRM
for a detailed description of these
registers.*/

/* load the write address pointer of the
endpoint into the USBFS_ARB_RW1_WA registers
*/

CY_SET_REG8(&USBFS_1_ARB_RW1_WA_PTR[0],
USBFS_1_EP[1].buffOffset & 0xFFu);

CY_SET_REG8(&USBFS_1_ARB_RW1_WA_MSB_PTR[0],
(USBFS_1_EP[1].buffOffset >> 8u));

/* Request DMA action. */
CyDmaChSetRequest(MyChannel, CPU_REQ);

/* Set the count and data toggle */
CY_SET_REG8(&USBFS_1_SIE_EP1_CNT0_PTR[0],
(0x40 >> 8u) | (USBFS_1_EP[1].epToggle)); /*
count is 64 */

CY_SET_REG8(&USBFS_1_SIE_EP1_CNT1_PTR[0],
0x40 & 0xFFu);

/* load the read address pointer of the
endpoint into the USBFS_ARB_RW1_RA registers
*/
CY_SET_REG8(&USBFS_1_ARB_RW1_RA_PTR[0],
USBFS_1_EP[1].buffOffset & 0xFFu);
CY_SET_REG8(&USBFS_1_ARB_RW1_RA_MSB_PTR[0],
(USBFS_1_EP[1].buffOffset >> 8u));

USBFS_1_EP[1].apiEpState =
USBFS_1_NO_EVENT_PENDING;

/* Write the Mode register */
CY_SET_REG8(&USBFS_1_SIE_EP1_CR0_PTR[0],
USBFS_1_EP[1].epMode);

while(!FINISHED); /* Wait for DMA done
interrupt*/
FINISHED=0;

}
}
}

/* [] END OF FILE */
```

After modifying the source files, rebuild the project. You are now ready to program the custom firmware into PSoC 3. Pressing the 'Program' icon in PSoC Creator, loads the firmware through the PSoC Programmer utility.

A message is displayed in the bottom status bar of PSoC Creator stating that the PSoC 3 device is successfully programmed.

Configure the CY8CKIT-001

The PSoC 3 development kit has a variable resistor that connects to the analog port defined earlier. Configure the development board using the following steps:

- Wire connection from VR to P6[5]
- Set J8 to VBUS

Creating a GUI using Cypress Suite USB 3.4

The Cypress Suite USB 3.4 is a powerful, yet easy to use set of USB Development tools that enables you to create .NET managed C# applications to communicate with PSoC 3 and other Cypress USB products. It includes *CyUSB.dll* which is a managed .NET class library. It provides a number of classes and methods which are used by you for easy communication with a USB device. A robust API simplifies communications between the host and peripheral device.

For this application, a GUI is created to capture the streamed data from the register. This is shown in Figure 8. A commented source code for this GUI is available as an attachment to this application note.

Figure 8 Cypress Suite USB 3.4 GUI for PSoC 3 USB Control

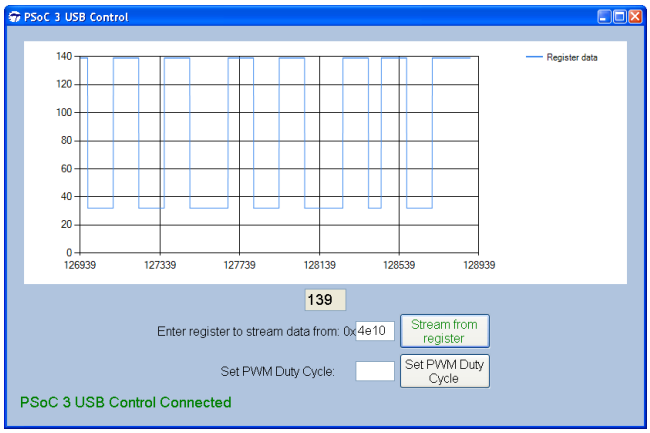
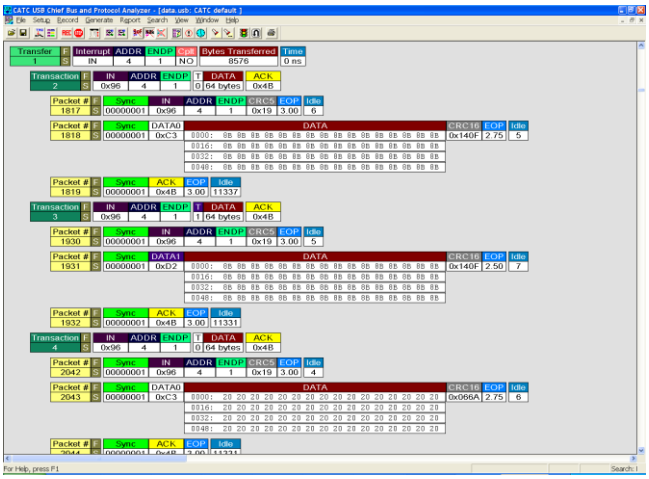


Figure 9 shows a CATC USB trace of the data transferred after pressing "Stream from register" in the GUI shown in Figure 9. This example uses a 2 Hz square wave input to the ADC.

Figure 9. CATC USB Trace of Data Transferred from PSoC 3 to the Host Computer



Create INF to bind cyusb.sys to PSoC 3

As the USB device created here of a vendor specific class, the cyusb.sys driver is used with it which is a generic USB driver.

To bind this window compatible driver to your device, an INF file is has to be specified which tells Windows about the services and the driver required.

The following code is an example for an INF file that provides the necessary information to Windows to allow the device to bind to *CYUSB.sys* and enable communications:

```
[Version]
Signature="$CHICAGO$"
Class=USB
ClassGUID= {36FC9E60-C465-11CF-8056-
444553540000}
provider=%CYPRESS%
; ---Uncomment and complete below to support
WHQL submission---;
;CatalogFile=
;DriverVer=mm/dd/yyyy,x.y.v.z
;-----------------------------------------

[Manufacturer]
%MfgName%=Cypress
```

[Cypress]

```
%VID_04b4&PID_1122.DeviceDesc%=CYUSB,
USB\VID_04b4&PID_1122
; This entry contains the VID/PID in the
; PSoC 3 firmware
; to do
; replace 04b4 (x2) with your unique VID
; replace 1122 (x2) with your unique PID
[DestinationDirs]
CYUSB.Files = 10, System32\Drivers

[CYUSB.Files]
CYUSB.SYS

[CYUSB.NT]
CopyFiles=CYUSB.Files
```

```
AddReg=CYUSB.AddReg

[CYUSB.NT.HW]
AddReg=CYUSB.AddReg.Guid

[CYUSB.NT.Services]
Addservice = CYUSB, 0x00000002,
CYUSB.AddService

[CYUSB.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,CyUsb.sys

[CYUSB.AddReg.Guid]
HKR,,DriverGUID,,%CYUSB.GUID%

;=====================================

[CYUSB.AddService]
DisplayName = %CYUSB.SvcDesc%
ServiceType = 1  ; SERVICE_KERNEL_DRIVER
StartType = 3    ; SERVICE_DEMAND_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL
ServiceBinary =
%10%\System32\Drivers\CYUSB.SYS
LoadOrderGroup = Base

[Strings]
PROVIDER="Cypress"
MFGNAME="Cypress Semiconductor"
CYUSB_INSTALL="Cypress Installation Disk"

VID_04b4&PID_1122.DeviceDesc="Cypresss PSoC
3 USB Control example project."
; VID/PID combination programmed in PSoC 3
; firmware
; To do
; replace "Cypress...." string with your own
; meaningful text
; replace the XXXX with your unique VID
; replace the YYYY with your unique PID

CYUSB.SvcDesc="Cypress USB Device"
CYUSB.GUID="{AE18AA60-7F6A-11d4-97DD-
00010229B959}"
; Replace GUID with your own unique GUID
```

Summary

This application note illustrates the creation of a simple project that takes advantage of the versatility of PSoC 3 combined with other Cypress utilities. This project demonstrates a design to control a PSoC 3-based device from a host computer and to monitor variables during run time.

About the Authors

Name: Hridya Valsaraju

Title: Applications Engineer

Contact: hridya.v@cypress.com

Name: Brad Haber

Title: Applications Engineer Sr

Contact: brad.haber@cypress.com

Related Application Notes and Example Projects:

Application Notes:

1. USB Peripheral Basics - AN57294
2. PSoC® 3 / PSoC 5 USB HID Fundamentals - AN57473
3. PSoC® 3 / PSoC 5 USB HID Intermediate - AN58726
4. USB BULK Loopback with PSoC® 3 - AN56718

Example Projects:

1. USBUART in PSoC® 3 / PSoC 5
2. Isochronous Transfers in PSoC® 3 / PSoC 5
3. Interrupt Loopback with PSoC® 3 / PSoC 5

Document History

Document Title: USB Vendor Commands with PSoC® 3 and PSoC 5

Document Number: 001-56377

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2811300	BHA/HRID	11/18/2009	New application note
*A	3010626	HRID	08/19/2010	Updated with PSoC 5
*B	3092283	HRID	11/23/2010	The Code Font size has been changed in page 6, 7, 8,9,10 and 11.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/