International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 4, Issue 3, March 2015 ISSN 2319 - 4847

DMA& USB INTERFACING FOR KEYBOARD USING CYPRESS PSoC-5

Er. Harisharan Aggarwal#1 **, Er. Yadwinder Singh**#2

#1.HOD (ECE), Guru Kashi University, Talwandi Sabo

#2.(student), Guru Kashi University, Talwandi Sabo

ABSTRACT

The main objective of this paper is to make us understand the role of importance of PSoC-5in our lives and also to know about its various applications. Here we are going to provide a complete report on the functionality and also the implementation of various projects on Psoc-5. Our main application over which we will work on is ADC data buffering using DMA & USB Key HID interfacing with Keyboard.In case of ADC data buffer we will see the data will buffer easily with less losses. Also we will see how we can interface keyboard with USB HID. These are the widel applications on which we will work on. The need of PSoC-5 in this era is incredible. The PSoC-5 has got time management skills which makes easy and error free research areas. PSoC-5 helps systems to retain compatibility, flexibility and is also cost effective. So as to meet the demands of the fast growing technology we need to learn the various aspects and functionality of the PSoC-5which will help to make our work and knowledge more valuable. PSoC-5is used in wide no. of applications. It has got a lot of scope in the future as well in the present. The applications are power management, wireless communication, automotive transportation, centre console, touch screens, button replacement, HVAC, motor control, switches, ultrasonic parking, computer and peripherals like iPod, iPhone, iPad accessories and thermal management. It has got a lot of scope in medical lines like making Blood Pressure monitor, Blood glucose meter, fertility monitor and infusion pump. Our goal is to implement the applications "ADC DATA BUFFERING USING DMA" and "USB HID INTERFACING WITH KEYBOARD". For achieving our goal we are starting here with small experiments. In case of ADC data buffering using DMA through PSoC-5 the data get buffer with less losses and we get the result by the easiest way. In the other application we can interface the USB with Keyboard very easily.

1.INTRODUCTION

PSoC-5 microcontroller is plays a very widel role in many applications now a days. As for a project it covers all the needs on a single chip.It gives a whole new concept of microcontroller as it contains both analog & digital blocks.PSoC-5(Programmable system on chip) [9] is made by Cypress semiconductor & is to be defined as the family of integrated circuits.
Here we are going to do study about the basic principle of Psoc-5, implementation of small projects which will implemented in all the processor modules ie CY8C28, CY8C38 CY8C55.Further we will work on ADC data buffering using DMA and USB HID intermediate with Keyboard as the big applications of Psoc-5.
We will work here on PSoC-5 Designer, PSoC-5 Creator, PSoC-5 Programmer for achieving the results & for further analysing them.PSoC-5(Programmable System on Chip) represents a whole new concept in microcontroller development. In addition to all the standard elements of 8-bit microcontrollers, PSoC-5chips feature digital and analog programmable blocks, which themselves allow implementation of large number of peripherals.Digital blocks consist of smaller programmable blocks that can be configured to allow different development options. Analog blocks are used for development of analog elements, such as analog filters, comparators, instrumentational (non)inverting amplifiers, as well as AD and DA convertors.Number of components that can be devised is primarily a function of the available programmable blocks. Depending on the microcontroller family, PSoC-5chips have 4–16 digital blocks, and 3–12 analog programmable block

2.THEORY

We are focusing here on reading all the desired materials and the datasheet of Psoc-5. We will use the tools like PSoC-5Creator and designer to run different programmes which are necessary to understand the functioning of both the tools.
Firstly we will simply blink the LEDs using the PWM [10] as peripheral on PSoC-5 Creator and Designer using the development Kit. We will also repeat the project by taking different ports period and period width.
Secondly we will be implementing a project which demonstrates a 9-bit Delta Sigma ADC by measuring the voltage of the potentiometer center tap wiper and displaying the result on the LCD on all the processor modules and all the software.These all were the basic projects that we done for knowing the steps for performing the applications.These basic projects that we done helps us for getting our main applications.
After doing all these minor projects we comes to know about the usage of the PSoC-5 we will directly focus on achieving the applications of Psoc-5. In case of ADC data buffering using DMA. DMA [13] controller is uses to handle

the data transfer without CPU intervention. DMA is useful in applications that require ADC data buffering and allows the CPU for doing the simultaneous tasks. Here we will see the basics of 8-bit, 16-bit, and 20-bit Delta Sigma ADC data buffering using DMA with example projects. The 20-bit example project accompanying this application note demonstrates problems with data buffering using DMA. These problems occur when the peripheral spoke width is less than the actual data width. The project describes how to tackle this using multiple DMA channels. The DMA is used to move data from a source to destination without CPU intervention is the basic concept on which we will further work.In our third application i.e. USB HID intermediate with Keyboard, We are using here Psoc-5 full speed USB interface, where we will take the basics of USB HID development and its implementation is depends upon the knowledge by which users get knows that how to incorporate the OUTPUT items to receive information from a host device using the status LEDs on a keyboard taking as an example, we also send the keyboard information as an INPUT to type a predefined string of text into a text editor.

Figure2.1 CY8C55 Family Processor Module[3]

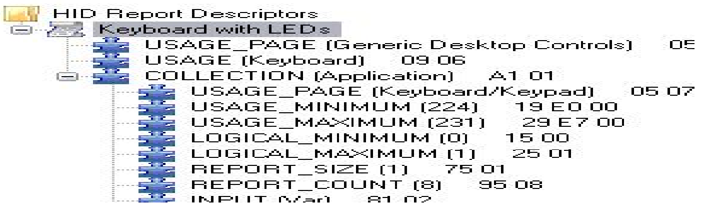
3.METHODEOLOGY

Here we will covers the next level of Human Interface Devices (HID) [11] USB development on PSoC-53 / PSoC-55 discussing OUTPUT items using a keyboard. Weare using here Psoc-5's full speed USB interface, where we will take the basics of USB HID development and its implementation is depends upon the knowledge by which users get knowsthat how to incorporate the OUTPUT items to receive information from a host device using the status LEDs on a keyboard taking as an example, we also send the keyboard information as an INPUT to type a predefined string of text into a text editor.

Human Interface Devices (HID) [12]are the devices which enable the consumers to easily interface with and control their PCs. almost all of the HIDs send information to the host. Keyboards and mice are the most common examples.However, there is often PC have needs to send the information to the peripheral and for receiving the information is from the host that is accomplished with an Output Report.

Output Items in the HID Report Descriptor must be configured properly for receiving this Output Report properlyconfiguring the Output items.The transferring result can take the form of a LED on a keyboard.The Keyboard requires the interaction from the host to provide the information to it. In a keyboard the Input has taken from the keys that the user presses and Output taken from the LEDs on the keyboard by which the information will display (such as checking if the caps lock is enabled or not).

For understanding the Output Items in greater detail, the Report Descriptor will uses for a 104-key keyboard. In Figure 3.1 it will shows HID Report Discriminator in which by expanding the parameters we will able to get output. The chosen organization of the report descriptor follows the required format for a boot interface, which we are discussing in the Boot Interface section. However, the report descriptor can be organized in various ways and still function as a keyboard with LEDs.


```
HID Report Descriptors
  Keyboard with LEDs
    USAGE_PAGE (Generic Desktop Controls)    05
    USAGE (Keyboard)    09 06
    COLLECTION (Application)    A1 01
      USAGE_PAGE (Keyboard/Keypad)    05 07
      USAGE_MINIMUM (224)    19 E0 00
      USAGE_MAXIMUM (231)    29 E7 00
      LOGICAL_MINIMUM (0)    15 00
      LOGICAL_MAXIMUM (1)    25 01
      REPORT_SIZE (1)    75 01
      REPORT_COUNT (8)    95 08
      INPUT (Var)    81 02
```
Figure 3.1 Keyboard Report Descriptor

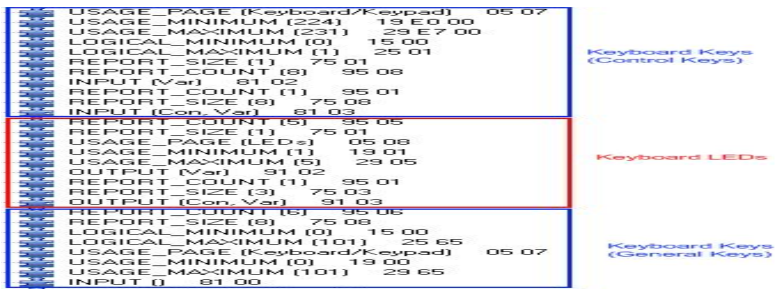
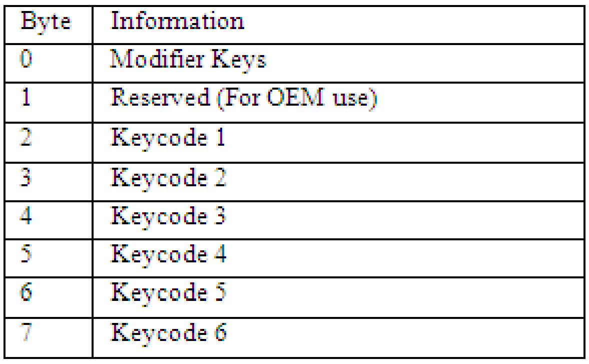

```
USAGE_PAGE (Keyboard/Keypad)    05 07
USAGE_MINIMUM (224)    19 E0 00
USAGE_MAXIMUM (231)    29 E7 00
LOGICAL_MINIMUM (0)    15 00
LOGICAL_MAXIMUM (1)    25 01
REPORT_SIZE (1)    75 01
REPORT_COUNT (8)    95 08
INPUT (Var)    81 02
REPORT_COUNT (1)    95 01
REPORT_SIZE (8)    75 08
INPUT (Con, Var)    81 03
REPORT_COUNT (5)    95 05
REPORT_SIZE (1)    75 01
USAGE_PAGE (LEDs)    05 08
USAGE_MINIMUM (1)    19 01
USAGE_MAXIMUM (5)    29 05
OUTPUT (Var)    91 02
REPORT_COUNT (1)    95 01
REPORT_SIZE (3)    75 03
OUTPUT (Con, Var)    91 03
REPORT_COUNT (6)    95 06
REPORT_SIZE (8)    75 08
LOGICAL_MINIMUM (0)    15 00
LOGICAL_MAXIMUM (101)    25 65
USAGE_PAGE (Keyboard/Keypad)    05 07
USAGE_MINIMUM (0)    19 00
USAGE_MAXIMUM (101)    29 65
INPUT ()    81 00
```

Figure3.2 Sectioned Keyboard Report Descriptor

Bytes are sent to the host as an Input Item and one byte is received from the host as an Output Item. There are eight bytes which are sent to the host and are firstly organized with modifier keys.They are followed by general keys which are shown in Table 3.1A modifier key is uses which is a key and with the help of it the functions of a general key are to be modifies for performing the alternative functions.Modifier keys on the keyboard include the shift, alt, and GUI keys. All other remaining keys are considered as general keys. The format in which this information is transferred to the host is shown in Table

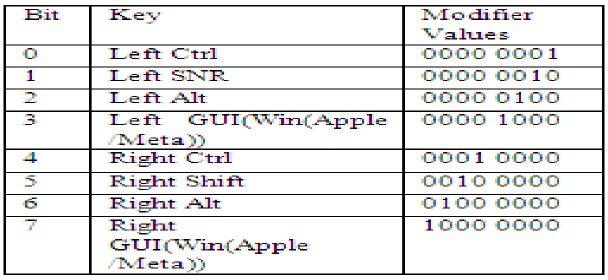
Table3.1. (Keyboard Input Report Table)



Byte	Information
0	Modifier Keys
1	Reserved (For OEM use)
2	Keycode 1
3	Keycode 2
4	Keycode 3
5	Keycode 4
6	Keycode 5
7	Keycode 6

There are only seven bytes of useful information when we sent information is of 8-byte packet. This is because of the reserved byte (Byte 1) shown in Table 3.2 It is intended for OEM [14] use and is not used in most applications. There is to be a reserved key that may be used on a keyboard which contains a non-standard key and performs a function that is specific to that PC. We can see it in laptops and tablet computers.
In most of the keyboards that are purchased in the consumer market, the reserved byte remains a constant value of 0x00 and the remaining bytes in the configuration are the general keys. In a given transaction six available bytes of information indicate that up to six key codes can be sent to the PC and it will enables up to six simultaneous key presses. The order of the key codes in the array does not have any significance.When we compare the newly received report from the previous report the sorting is accomplished then.

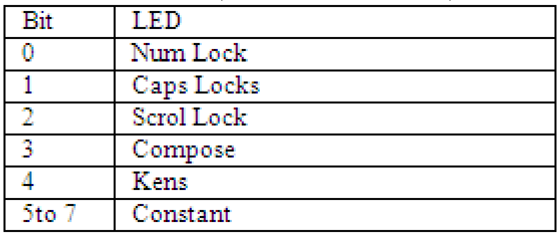
Table 3.2(Modifier Key Index)



Bit	Key	Modifier Values
0	Left Ctrl	0000 0001
1	Left SNR	0000 0010
2	Left Alt	0000 0100
3	Left GUI(Win(Apple /Meta))	0000 1000
4	Right Ctrl	0001 0000
5	Right Shift	0010 0000
6	Right Alt	0100 0000
7	Right GUI(Win(Apple /Meta))	1000 0000

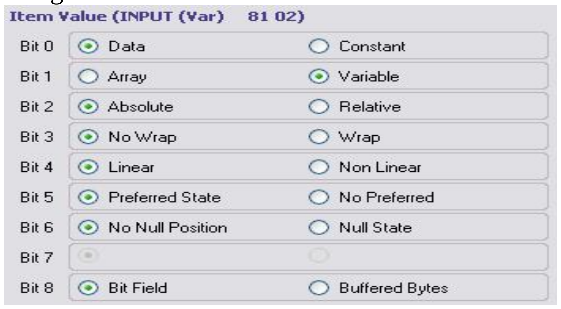
In Table 3.2 We can see that each modifier key has a corresponding bit associated with it.It means in a bit field of information the modifier keys are stored. The HID Usage Tables shows that the Usage values for the modifier keys are in the range from E0-E7. However, it is also to be seen that the usage values are not sent as array data. The modifier keys are to be sent as variable data and it means that each individual bit in the 8-bit value is corresponds to one of the modifier keys. The Usage Minimum/Maximums are then used to link the modifier key information in the bit field to the proper Usage value with the Usage Minimum/Maximum.
The difference between Array and Variable becomes more relevant in this keyboard application.We can differentiate them as follows:
"Array versus Variable: Array means only controls that are currently active are reported such as a button being pressed. Variable means that the data reported is the current state of every control regardless if a button is pressed or not."
According to the USB HID Usage table, the usage values for the LEDs are provided in Table 3.3 LED information is an Absolute Item, which means that the Output Report must include the state of each LED with '0' meaning off and '1' meaning on.The LEDs are not relative items, which means that a '0' represents no change and '1' represents a change in state.

Table 3.3(LED Indicator Index)



Bit	LED
0	Num Lock
1	Caps Locks
2	Scrol Lock
3	Compose
4	Kens
5to 7	Constant

When referencing the Report Descriptor the first Input item is from top down and for the modifier keys (Shift, Alt, and GUI). These keys are configured in a Variable configuration to implement a bit field. The Input Item configuration for Keyboard modifier keys is shown in Figure 3.1

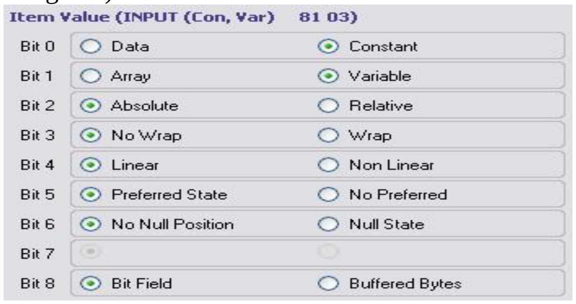


Item Value (INPUT (Var)   81 02)

Bit 0	[✓ Data]	Constant
Bit 1	Array	[✓ Variable]
Bit 2	[✓ Absolute]	Relative
Bit 3	[✓ No Wrap]	Wrap
Bit 4	[✓ Linear]	Non Linear
Bit 5	[✓ Preferred State]	No Preferred
Bit 6	[✓ No Null Position]	Null State
Bit 7		
Bit 8	[✓ Bit Field]	Buffered Bytes

Figure 3.1 Input Item for Keyboard Modifier Keys

The entire byte is padded with zeros and configured as a constant for reserving the second byte in the data structure.By setting bit 0 to a value to '1' we declaring this byte as a constant. (By following figure 3.2 we can see it easily and this is done because each report is byte aligned.)

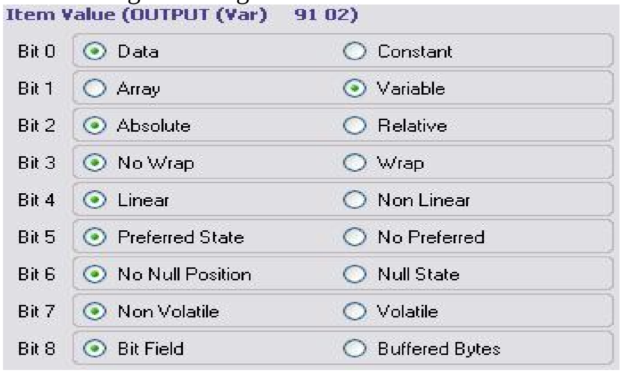


Item Value (INPUT (Con, Var)   81 03)

Bit 0	Data	[✓ Constant]
Bit 1	Array	[✓ Variable]
Bit 2	[✓ Absolute]	Relative
Bit 3	[✓ No Wrap]	Wrap
Bit 4	[✓ Linear]	Non Linear
Bit 5	[✓ Preferred State]	No Preferred
Bit 6	[✓ No Null Position]	Null State
Bit 7		
Bit 8	[✓ Bit Field]	Buffered Bytes

Figure 3.2 Input Item for Reserving Second Byte

The next step in configuring the Descriptor is to configure the Output Item for the LEDs on the keyboard. Here two things are very important and they are:The prefix value is '91' signifying an Output Item with one byte to follow and this is because the Item is an Output Item. We can modified the Bit 7 and it is not the case with any Input Item.It has reason that this value only applies to Output and Feature Items. Bit 7 is set to '0' signifying that the bit is non-volatile which means the device only alters the value with host interaction. It is important to note that bit 1 is set to be variable. For a bit field configuration the Item will again configure.

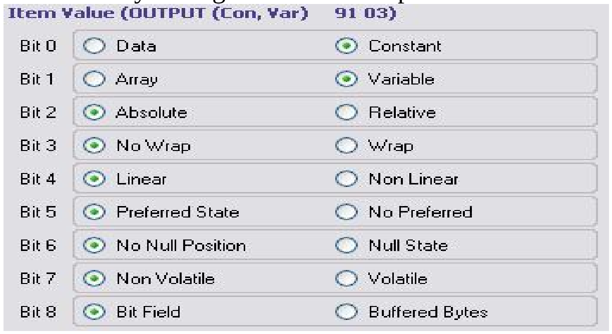


Item Value (OUTPUT (Var)   91 02)

Bit 0	[✓ Data]	Constant
Bit 1	Array	[✓ Variable]
Bit 2	[✓ Absolute]	Relative
Bit 3	[✓ No Wrap]	Wrap
Bit 4	[✓ Linear]	Non Linear
Bit 5	[✓ Preferred State]	No Preferred
Bit 6	[✓ No Null Position]	Null State
Bit 7	[✓ Non Volatile]	Volatile
Bit 8	[✓ Bit Field]	Buffered Bytes

Figure 3.3 Output Item for LEDs

Because all information is byte aligned thus only five bits out of the total eight bits are used for LED information and we can easily see in Figure 3.4, the remaining three bits are reserved by padding the bits with a value of '0' by setting those bits to remain constant. This is done by setting bit 0 in the Output Item to a value of '1'.

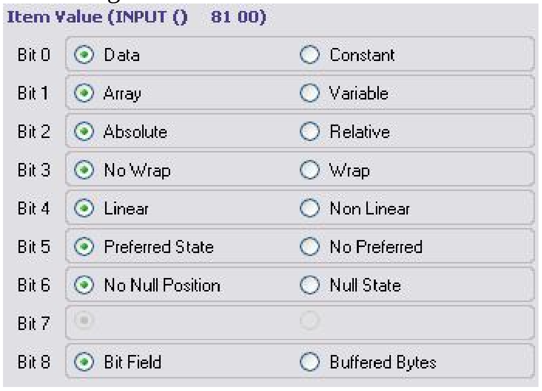


Item Value (OUTPUT (Con, Var)   91 03)

Bit 0	Data	[✓ Constant]
Bit 1	Array	[✓ Variable]
Bit 2	[✓ Absolute]	Relative
Bit 3	[✓ No Wrap]	Wrap
Bit 4	[✓ Linear]	Non Linear
Bit 5	[✓ Preferred State]	No Preferred
Bit 6	[✓ No Null Position]	Null State
Bit 7	[✓ Non Volatile]	Volatile
Bit 8	[✓ Bit Field]	Buffered Bytes

Figure 3.4 Output Item for Padding LEDs

The final Input Item to configure is for the general (non-modifier) keys. The HID specification requires these keys to be configured with an Array and Absolute configuration.


Item Value (INPUT () 81 00)

Bit 0	[✓ Data]	Constant
Bit 1	[✓ Array]	Variable
Bit 2	[✓ Absolute]	Relative
Bit 3	[✓ No Wrap]	Wrap
Bit 4	[✓ Linear]	Non Linear
Bit 5	[✓ Preferred State]	No Preferred
Bit 6	[✓ No Null Position]	Null State
Bit 7		
Bit 8	[✓ Bit Field]	Buffered Bytes

Figure 3.5 Input Item for General Keyboard Button

For its functionality we have to understand the each block of the Report Descriptor and how the information is interpreted, the next step is to study the Report Descriptor in detail .

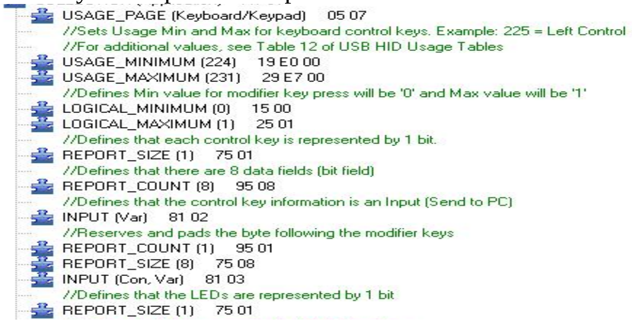

```
USAGE_PAGE (Keyboard/Keypad)    05 07
//Sets Usage Min and Max for keyboard control keys. Example: 225 = Left Control
//For additional values, see Table 12 of USB HID Usage Tables
USAGE_MINIMUM (224)    19 E0 00
USAGE_MAXIMUM (231)    29 E7 00
//Defines Min value for modifier key press will be '0' and Max value will be '1'
LOGICAL_MINIMUM (0)    15 00
LOGICAL_MAXIMUM (1)    25 01
//Defines that each control key is represented by 1 bit.
REPORT_SIZE (1)    75 01
//Defines that there are 8 data fields (bit field)
REPORT_COUNT (8)    95 08
//Defines that the control key information is an Input (Send to PC)
INPUT (Var)    81 02
//Reserves and pads the byte following the modifier keys
REPORT_COUNT (1)    95 01
REPORT_SIZE (8)    75 08
INPUT (Con, Var)    81 03
//Defines that the LEDs are represented by 1 bit
REPORT_SIZE (1)    75 01
```

Figure 3.6 Commented Keyboard Report Descriptor

Table 3.4. (ALT+CTRL+DEL Example)

Step	Modifier Byte	Array Byte
Left Alt Down	00000100	0x00
Right Ctrl Down	00010100	0x00
Delete Down	00010100	0x4C
Delete Up	00010100	0x00
Right Ctrl Up	00000100	0x00
Left Alt Up	00000000	0x00

It is a PC user desires to enter a PC's BIOS upon boot up to edit it. For getting this, the USB HID Specification defines a boot protocol for keyboard. These protocols are predefined and the device is required to conform to the specifications to be a boot device. The boot protocol supports up to eight bytes of information.BIOS can ignores anything over these eight bytes[11]. It is a very important point that in actually BIOS does not read the report descriptor because of the predefined standard, the BIOS have expectations of the information in a certain format. Because of this reason a HID device such as a keyboard can have two interfaces.These interfaces are: one is the boot interfaces and the other is USB aware interfaces. In first kind of interfaces i.e. the boot interface, the requirement of a hard coded Report Descriptor is not necessary here.

This is the project which act as a keyboard and it will type the messages also will displays the status of num lock, caps lock, and scroll lock.

Here we are going to introduce the user to a HID device and it consists of both an Input and Output transfer. The project acts as a keyboard that types messages and displays.For getting our application we have to do following steps:

For this project, we will start by opening PSoC-5Creator and thus create an empty project named 'Project_1_Keyboard'. After the project has been created and PSoC-5Creator is completely loaded, we will place the following components into the schematic entry page (TopDesigncysch).

Character LCD
(1) USBFS
(2) Digital Input Pins
(3) Digital Output Pins

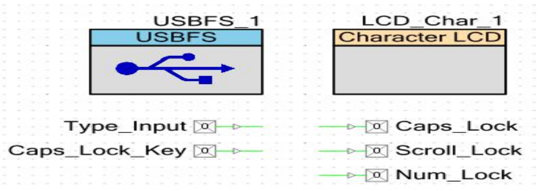
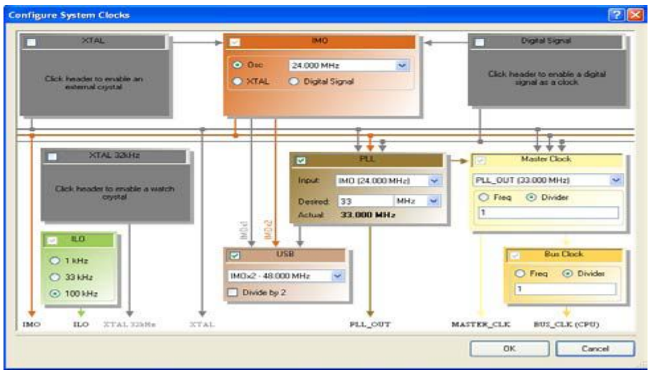


Figure 3.7 PSoC-5Creator Components for Keyboard

These pins required certain changing in settings.Thus for the Input Pins, we will open the pin configuration customizer and be sure to uncheck the ‘HW Connection’ box and thus changing their drive mode to ‘Resistive Pull Down’ which is done by clicking on the ‘General’ tab in the pin customizer menu. For the Digital Output Pins, open the configuration menu, uncheck the ‘HW Connection’ box, and change their drive mode to ‘Strong’.
When the components are placed, we go to the Workspace Explorer window and thus double click on MyFirstKeyboardHID.cydwr. Click on the ‘Clocks’ tab and then double click on one of the clocks to open the GUI clock configuration window.We will make the following changes to the clock as shown in Figure 3.7.
IMO → 24 MHz
ILO → 100 kHz
PLL → Input: IMO, Desired: 48 MHz
Master Clock → PLL Out
USB Clock → IMO x2



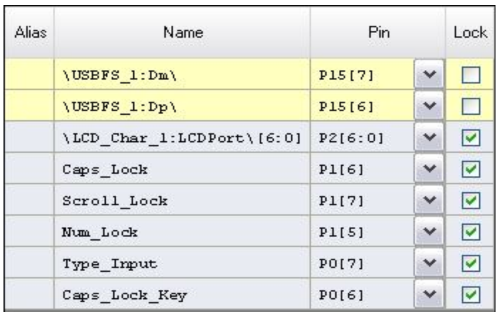
Configure System Clocks

IMO: [✓ Osc] 24.000 MHz; XTAL; Digital Signal
ILO: 1 kHz; 33 kHz; [✓ 100 kHz]
PLL: Input: IMO (24.000 MHz); Desired: 33 MHz; Actual: 33.000 MHz
Master Clock: PLL_OUT (33.000 MHz); Freq [✓ Divider] 1
USB: IMOx2 - 48.000 MHz; Divide by 2
Bus Clock: Freq [✓ Divider] 1

XTAL
Click header to enable an external crystal
Digital Signal
Click header to enable a digital signal as a clock
XTAL 32kHz
Click header to enable a watch crystal
IMOx1
IMOx2
IMO
ILO
XTAL 32kHz
XTAL
PLL_OUT
MASTER_CLK
BUS_CLK (CPU)
OK
Cancel

Figure 3.8 System Clock Configuration Window.

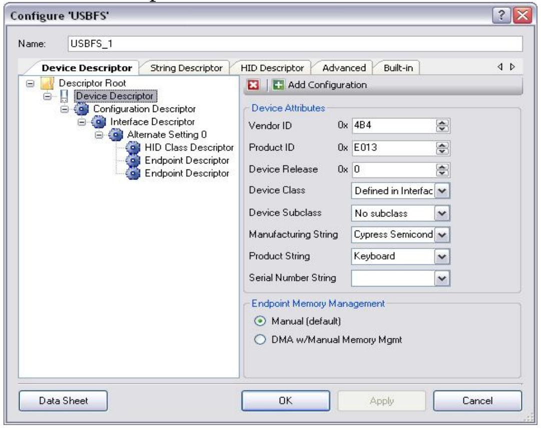
The next step is to configure the pin configuration for the project. For this we have to click on the ‘Pins’ tab located in Project 1 – Keyboardcydwr and change the pins to resemble Figure 3.8. It is important for us to remember that the USB pins are always located at P15[6] and P15[7] whereas the other pins can be moved if desired. The following pinout is configured to work with the project:



Alias	Name	Pin	Lock
	\USBFS_1:Dm\	P15[7]	☐
	\USBFS_1:Dp\	P15[6]	☐
	\LCD_Char_1:LCDPort\[6:0]	P2[6:0]	✓
	Caps_Lock	P1[6]	✓
	Scroll_Lock	P1[7]	✓
	Num_Lock	P1[5]	✓
	Type_Input	P0[7]	✓
	Caps_Lock_Key	P0[6]	✓

Figure 3.9 PSoC-5Creator Pin Configuration (Keyboard)

In next step, the USB component needs to be configured using the USB Wizard. But there is to be addition of an OUT EP. Then we will double click on the USBFS component for opening the Configuration Wizard. Configure the Vender ID (VID) and Product ID (PID) to be 0x4B4 and 0xE013. We will also fill in the Manufacture and Product strings. The strings shown in Figure 3.9 can be used or other strings can be chosen. We can also change the values for the VID and PID. For demonstration purposes any value can be used but if entering production or distributing the example project, our own VID must be assigned from the USB Implementers Forum.



Configure 'USBFS'

Name: USBFS_1

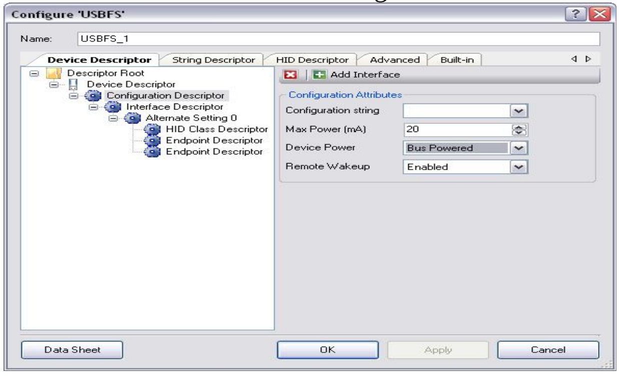
Descriptor Root
- Device Descriptor
  - Configuration Descriptor
    - Interface Descriptor
      - Alternate Setting 0
        - HID Class Descriptor
        - Endpoint Descriptor
        - Endpoint Descriptor

Device Attributes
Vendor ID 0x 4B4
Product ID 0x E013
Device Release 0x 0
Device Class: Defined in Interfac
Device Subclass: No subclass
Manufacturing String: Cypress Semicond
Product String: Keyboard
Serial Number String:

Endpoint Memory Management
[✓ Manual (default)]
DMA w/Manual Memory Mgmt

Figure 3.10 USB Device Descriptor Setup

Our next step is to be the selection of ‘Configuration Descriptor’ and the window changes as shown in Figure below.
Since the project is bus powered, limit the maximum current that can be supplied to the device. It is important to specify a value that is appropriate for the device. Here we are using 20 mA which is more than sufficient.



Configure 'USBFS'

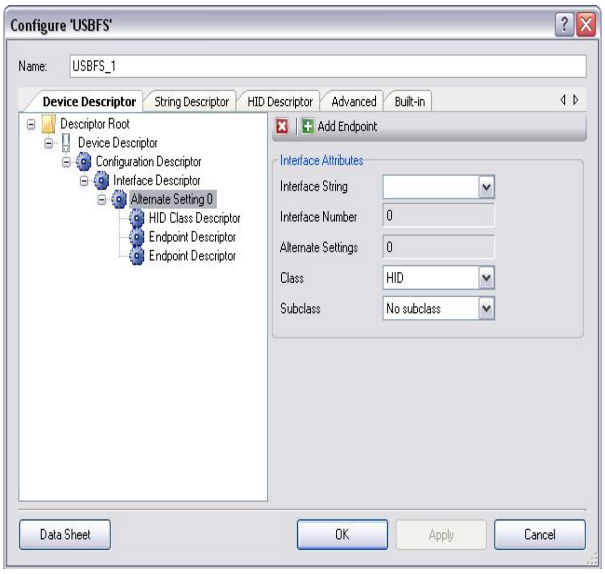
Name: USBFS_1

Descriptor Root
Device Descriptor
Configuration Descriptor
Interface Descriptor
Alternate Setting 0
HID Class Descriptor
Endpoint Descriptor
Endpoint Descriptor

Configuration Attributes
Configuration string:
Max Power (mA): 20
Device Power: Bus Powered
Remote Wakeup: Enabled

Figure 3.11 USB Device Descriptor Setup

Thus configure the interface descriptor. For doing this, we will click on Alternate Setting 0.We have also need to set the Class type to HID. This inform us about the host,”that the attached device is a HID”. It is to be shown in figure below.Here it is necessary to know that the ‘Subclass’ is set for ‘No SubClass’. If a Boot Interface is required, the user must change the subclass to ‘Boot Interface’.



Configure 'USBFS'

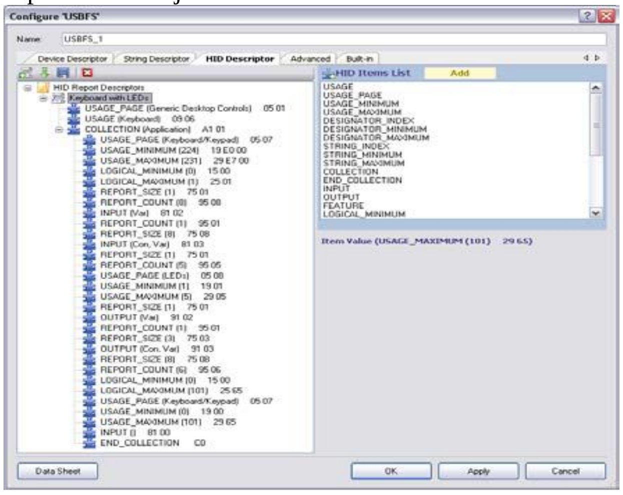
Name: USBFS_1

Descriptor Root
Device Descriptor
Configuration Descriptor
Interface Descriptor
Alternate Setting 0
HID Class Descriptor
Endpoint Descriptor
Endpoint Descriptor

Interface Attributes
Interface String:
Interface Number: 0
Alternate Settings: 0
Class: HID
Subclass: No subclass

Figure 3.12 USB Interface Descriptor Setup

With the interface a HID report descriptor must be associated and for this purpose we have to create the HID Report Descriptor.For doing this we will click on HID descriptor tab in the dialog.Thus we get the following window as shown in Figure 3.12.On this window we will do the require addition that are necessary for our application. After completion of the report descriptor, we will return to the HID Class Descriptor menu as shown in Figure 3.13 and thus set the HID Report to the name of the HID Report that was just created.



Configure 'USBFS'

Name: USBFS_1

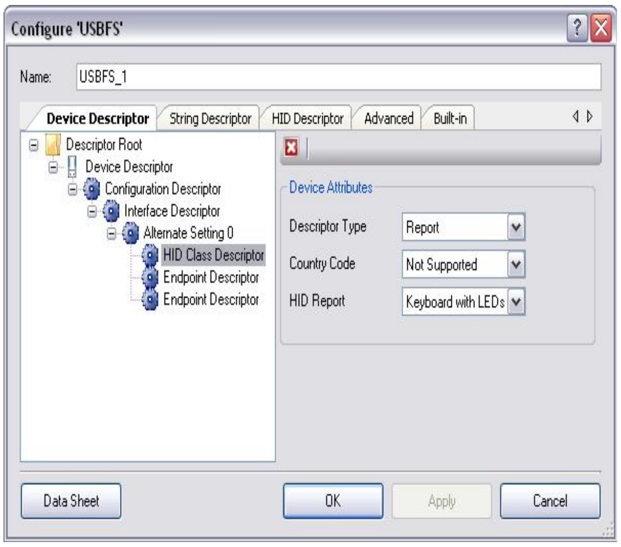
```
HID Report Descriptors
  Keyboard with LEDs
    USAGE_PAGE (Generic Desktop Controls)   05 01
    USAGE (Keyboard)   09 06
    COLLECTION (Application)   A1 01
      USAGE_PAGE (Keyboard/Keypad)   05 07
      USAGE_MINIMUM (224)   19 E0 00
      USAGE_MAXIMUM (231)   29 E7 00
      LOGICAL_MINIMUM (0)   15 00
      LOGICAL_MAXIMUM (1)   25 01
      REPORT_SIZE (1)   75 01
      REPORT_COUNT (8)   95 08
      INPUT (Var)   81 02
      REPORT_COUNT (1)   95 01
      REPORT_SIZE (8)   75 08
      INPUT (Con, Var)   81 03
      REPORT_SIZE (1)   75 01
      REPORT_COUNT (5)   95 05
      USAGE_PAGE (LEDs)   05 08
      USAGE_MINIMUM (1)   19 01
      USAGE_MAXIMUM (5)   29 05
      REPORT_SIZE (1)   75 01
      OUTPUT (Var)   91 02
      REPORT_COUNT (1)   95 01
      REPORT_SIZE (3)   75 03
      OUTPUT (Con, Var)   91 03
      REPORT_SIZE (8)   75 08
      REPORT_COUNT (6)   95 06
      LOGICAL_MINIMUM (0)   15 00
      LOGICAL_MAXIMUM (101)   25 65
      USAGE_PAGE (Keyboard/Keypad)   05 07
      USAGE_MINIMUM (0)   19 00
      USAGE_MAXIMUM (101)   29 65
      INPUT ()   81 00
      END_COLLECTION   C0
```

HID Items List

```
USAGE
USAGE_PAGE
USAGE_MINIMUM
USAGE_MAXIMUM
DESIGNATOR_INDEX
DESIGNATOR_MINIMUM
DESIGNATOR_MAXIMUM
STRING_INDEX
STRING_MINIMUM
STRING_MAXIMUM
COLLECTION
END_COLLECTION
INPUT
OUTPUT
FEATURE
LOGICAL_MINIMUM
```

Item Value (USAGE_MAXIMUM (101)   29 65)

Figure 3.13 USB HID Report Descriptor Setup



Configure 'USBFS'
Name: USBFS_1
Device Descriptor | String Descriptor | HID Descriptor | Advanced | Built-in

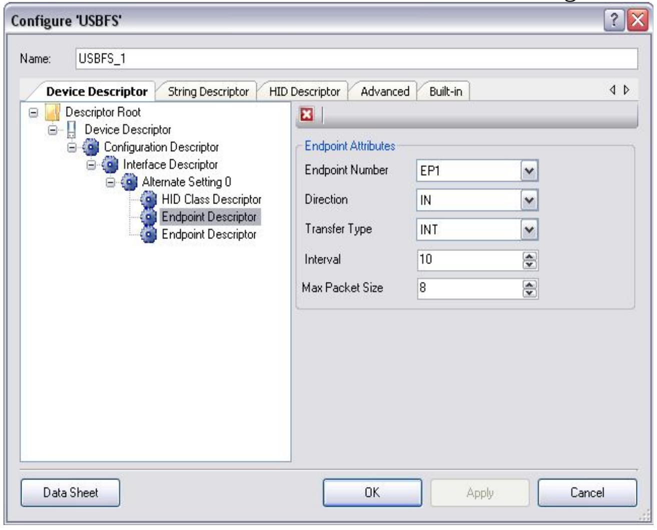
- Descriptor Root
  - Device Descriptor
    - Configuration Descriptor
      - Interface Descriptor
        - Alternate Setting 0
          - HID Class Descriptor
          - Endpoint Descriptor
          - Endpoint Descriptor

Device Attributes
Descriptor Type: Report
Country Code: Not Supported
HID Report: Keyboard with LEDs

Data Sheet | OK | Apply | Cancel

Figure 3.14 USB HID Class Descriptor Setup

Our final step of configuration is that we have to set the Endpoints. Because in this application the information's are to be sends and receives from the host and also an Input and Output endpoint are required. By default, here only one endpoint is listed, hence for add an additional endpoint, we can select the 'Alternative Setting 0' and then click the 'Add Endpoint' button. Further in next step we will select the 'Endpoint Descriptor' and under 'Endpoint Attributes', select the Endpoint number to be EP1, the direction to be 'IN', and the transfer type to be 'INT'. This is shown in Figure 3.14. Select the second Endpoint and select the Endpoint number to be EP2, the direction to be 'OUT', the transfer type to be 'INT', and the 'Max Packet Size' to be '1'. This is shown in Figure3.15.



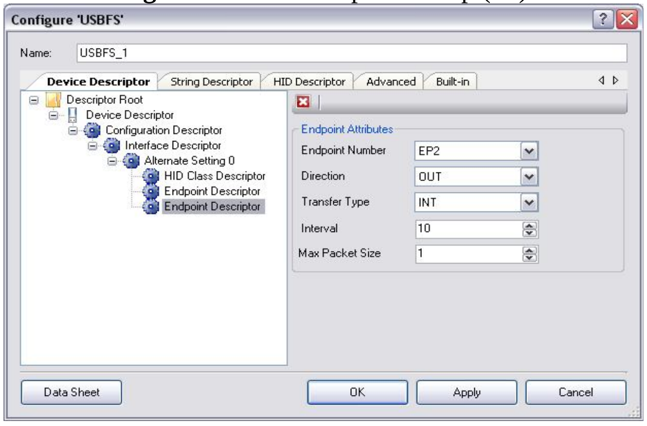
Configure 'USBFS'
Name: USBFS_1
Device Descriptor | String Descriptor | HID Descriptor | Advanced | Built-in

- Descriptor Root
  - Device Descriptor
    - Configuration Descriptor
      - Interface Descriptor
        - Alternate Setting 0
          - HID Class Descriptor
          - Endpoint Descriptor
          - Endpoint Descriptor

Endpoint Attributes
Endpoint Number: EP1
Direction: IN
Transfer Type: INT
Interval: 10
Max Packet Size: 8

Data Sheet | OK | Apply | Cancel

Figure3.15 USB Endpoint Setup (IN)



Configure 'USBFS'
Name: USBFS_1
Device Descriptor | String Descriptor | HID Descriptor | Advanced | Built-in

- Descriptor Root
  - Device Descriptor
    - Configuration Descriptor
      - Interface Descriptor
        - Alternate Setting 0
          - HID Class Descriptor
          - Endpoint Descriptor
          - Endpoint Descriptor

Endpoint Attributes
Endpoint Number: EP2
Direction: OUT
Transfer Type: INT
Interval: 10
Max Packet Size: 1

Data Sheet | OK | Apply | Cancel

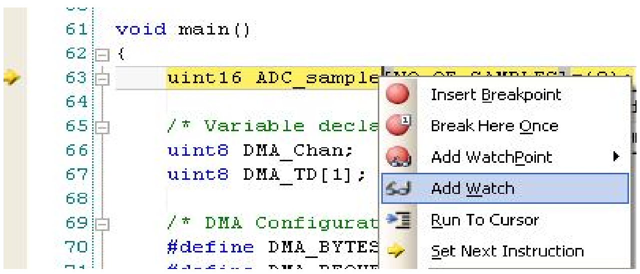
Figure3.16 USB Endpoint Setup (OUT)

The main.cfile will located in the Workspace Explorer and open the file to edit it. Delete the existing code in main.cand place the C code. After the code has been placed in the project, Build/Compile the project and program it into the PSoC-5device.

We will use the jumper wires to make the following connections on the PSoC-5CY8CKIT-001. If we want to change the pin configuration from Figure 4.31,we have to sure to change the wiring configuration accordingly.
SW1 -P0[6] -Caps Lock Key
SW2 -P0[7] -Type Constant String
LED2 -P1[5] -Num Lock LED
LED3 -P1[6] -Caps Lock LED
LED4 -P1[7] -Scroll Lock LED
LCD -P2[6:0] -Status Display
After the device is programmed, we will connect a USB cable from the PC to the DVK. When the device has enumerated, we will press the Caps, Num, and Scroll Lock keys on our keyboard and then observe the changes on LEDs and LCD accordingly. When we Press SW1 on the DVK and then observe the Caps Lock toggle. In next step we will open a text editor on the PC and then press SW2. Observe a string of text transmitted to PC and displayed in the text editor. We are free to edit the string in the main.cfile to type anything. The program by which we will able to obtainour application is to be explain on Appendix.

4 RESULT ANALYSIS

In this chapter we have shown the simulation results obtained by us while working on different applications of the Psoc-5.
ADC Data buffering using DMA
Build and program the chip
Press F5 or click the debug icon to download the program and debug.

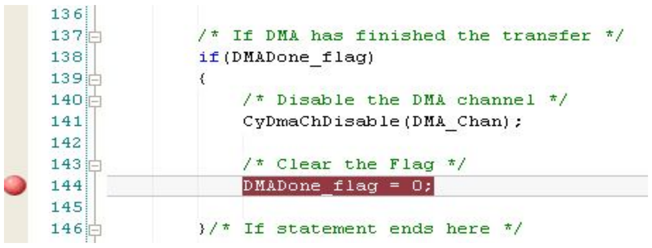


```
void main()
{
    uint16 ADC_sample

    /* Variable decla
    uint8 DMA_Chan;
    uint8 DMA_TD[1];

    /* DMA Configurat
    #define DMA_BYTES
```

Figure 4.1 Add watch timer.

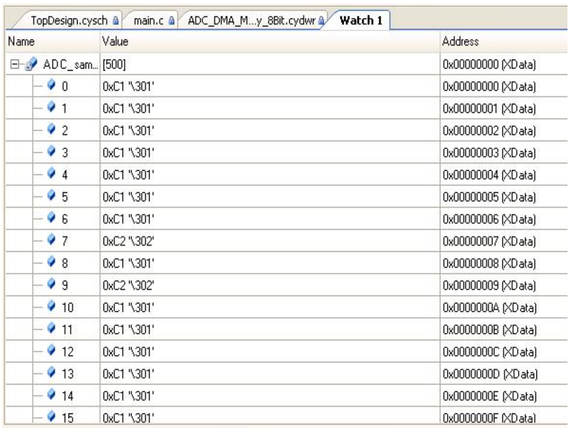


```
        /* If DMA has finished the transfer */
        if(DMADone_flag)
        {
            /* Disable the DMA channel */
            CyDmaChDisable(DMA_Chan);

            /* Clear the Flag */
            DMADone_flag = 0;

        }/* If statement ends here */
```

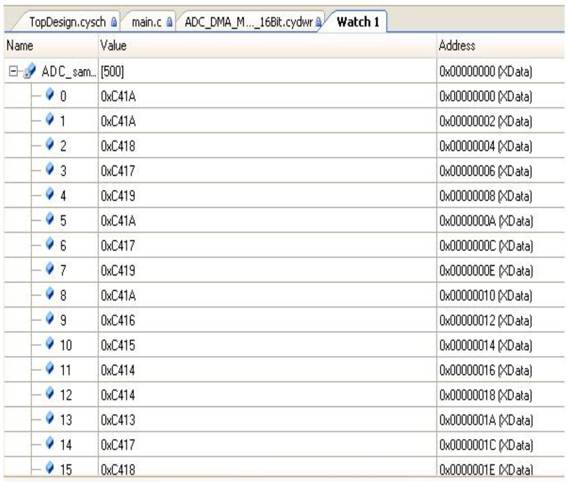
Figure 4.2 including Breakpointput a breakpoint inside in (DMA done flag).
The execution stops at the breakpoint after the DMA transfers the specified number of samples from ADC to memory and the result can be verified by monitoring the "ADCsample" array in the watch window. The outputs for 8 bit, 16 bit,20 bit and 12 bit ADC SAR as follows



Name	Value	Address
ADC_sam...	[500]	0x00000000 (XData)
0	0xC1 '\301'	0x00000000 (XData)
1	0xC1 '\301'	0x00000001 (XData)
2	0xC1 '\301'	0x00000002 (XData)
3	0xC1 '\301'	0x00000003 (XData)
4	0xC1 '\301'	0x00000004 (XData)
5	0xC1 '\301'	0x00000005 (XData)
6	0xC1 '\301'	0x00000006 (XData)
7	0xC2 '\302'	0x00000007 (XData)
8	0xC1 '\301'	0x00000008 (XData)
9	0xC2 '\302'	0x00000009 (XData)
10	0xC1 '\301'	0x0000000A (XData)
11	0xC1 '\301'	0x0000000B (XData)
12	0xC1 '\301'	0x0000000C (XData)
13	0xC1 '\301'	0x0000000D (XData)
14	0xC1 '\301'	0x0000000E (XData)
15	0xC1 '\301'	0x0000000F (XData)

Figure 4.3 Result of 8bit ADC data buffering

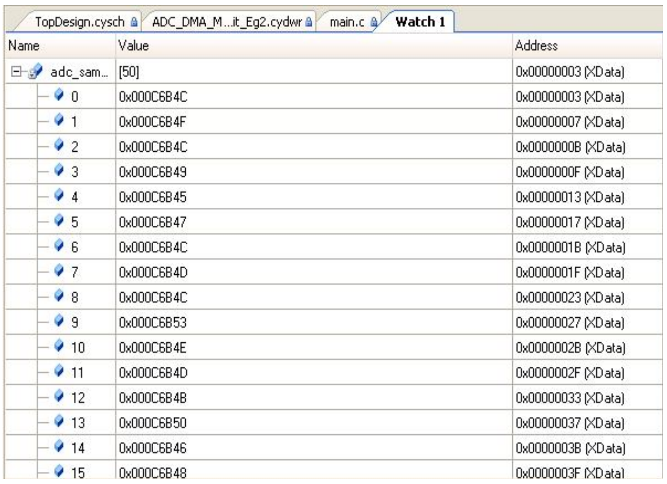
16 bit ADC data buffering



Name	Value	Address
ADC_sam...	[500]	0x00000000 (XData)
0	0xC41A	0x00000000 (XData)
1	0xC41A	0x00000002 (XData)
2	0xC418	0x00000004 (XData)
3	0xC417	0x00000006 (XData)
4	0xC419	0x00000008 (XData)
5	0xC41A	0x0000000A (XData)
6	0xC417	0x0000000C (XData)
7	0xC419	0x0000000E (XData)
8	0xC41A	0x00000010 (XData)
9	0xC416	0x00000012 (XData)
10	0xC415	0x00000014 (XData)
11	0xC414	0x00000016 (XData)
12	0xC414	0x00000018 (XData)
13	0xC413	0x0000001A (XData)
14	0xC417	0x0000001C (XData)
15	0xC418	0x0000001E (XData)

Figure 4.4 Result of 16 bit ADC data buffering

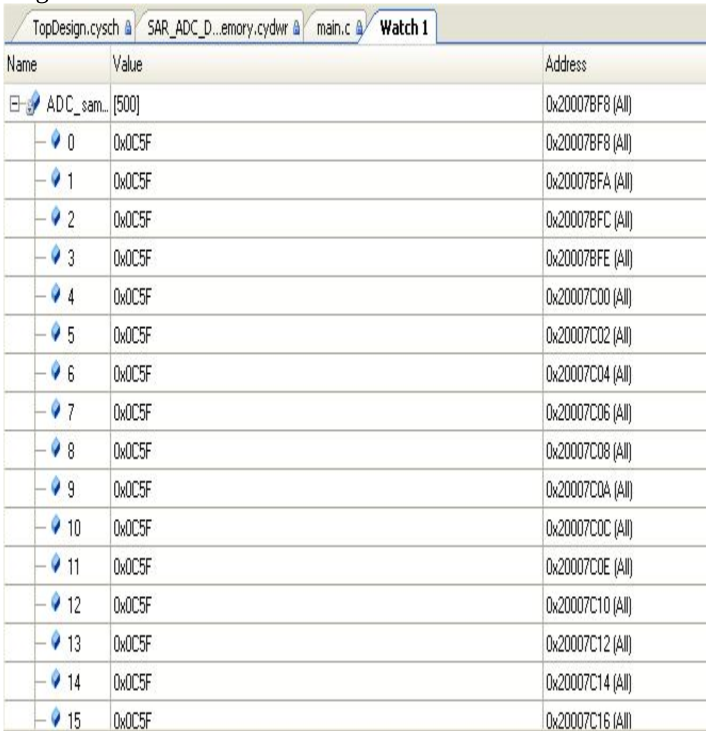
20 bit ADC data buffering



Name	Value	Address
adc_sam...	[50]	0x00000003 (XData)
0	0x000C6B4C	0x00000003 (XData)
1	0x000C6B4F	0x00000007 (XData)
2	0x000C6B4C	0x0000000B (XData)
3	0x000C6B49	0x0000000F (XData)
4	0x000C6B45	0x00000013 (XData)
5	0x000C6B47	0x00000017 (XData)
6	0x000C6B4C	0x0000001B (XData)
7	0x000C6B4D	0x0000001F (XData)
8	0x000C6B4C	0x00000023 (XData)
9	0x000C6B53	0x00000027 (XData)
10	0x000C6B4E	0x0000002B (XData)
11	0x000C6B4D	0x0000002F (XData)
12	0x000C6B4B	0x00000033 (XData)
13	0x000C6B50	0x00000037 (XData)
14	0x000C6B46	0x0000003B (XData)
15	0x000C6B48	0x0000003F (XData)

Figure 4.5 Result of 20 bit ADC data buffering

12 bit SAR ADC data buffering



Name	Value	Address
ADC_sam...	[500]	0x20007BF8 (All)
0	0x0C5F	0x20007BF8 (All)
1	0x0C5F	0x20007BFA (All)
2	0x0C5F	0x20007BFC (All)
3	0x0C5F	0x20007BFE (All)
4	0x0C5F	0x20007C00 (All)
5	0x0C5F	0x20007C02 (All)
6	0x0C5F	0x20007C04 (All)
7	0x0C5F	0x20007C06 (All)
8	0x0C5F	0x20007C08 (All)
9	0x0C5F	0x20007C0A (All)
10	0x0C5F	0x20007C0C (All)
11	0x0C5F	0x20007C0E (All)
12	0x0C5F	0x20007C10 (All)
13	0x0C5F	0x20007C12 (All)
14	0x0C5F	0x20007C14 (All)
15	0x0C5F	0x20007C16 (All)

Figure 4.6 Result of 12 bit SAR ADC data buffering

USB HID INTERFACING
The output for USB HID intermediate with Keyboard is as follows:

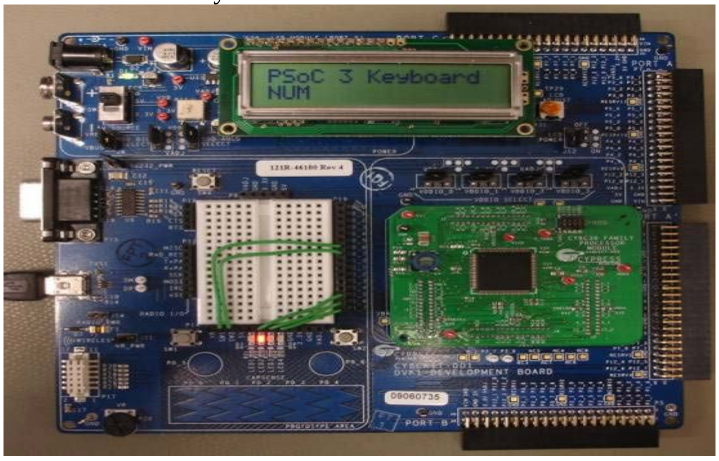


Figure 4.7 Result of USB HID Intermediate with Keyboard

5.APPLICATIONS

These all were the methodology by following the each and every steps from here we firstly implement the small projects and further will implement our main big applications i.e "ADC DATA BUFFERING USING DMA" and third application is "USB HID intermediate with Keyboard".For getting these main applications implementations we will follow the same method from these PSoC-5creator, PSoC-5designer and PSoC-5 Programmer but for these we will firstly study what is data buffering, what is ADC data and what the meaning of DMA, how data will buffer by DMA? In our next application i.e. USB HID intermediate with Keyboard, here we will see how we can intermediate the keyboard with USB HID.We will provide the input from Keyboard by pressing any key on Keyboard like Caps Lock and then output we will see on PSoC-5board by LED glowing. Hence we will comes to know that the USB HID get intermediate with Keyboard. By knowing all these concepts and all these questions we will easily implement our applications

6.CONCLUSION

This dissertation focuses on use of PSoC-5(Programmable System on Chip) for achieving big applications. We designed here ADC data buffering using DMA and USB HID intermediate with Keyboard. In case of ADC data buffering using DMA, we buffers 8 bit, 16 bit, 20 bit and 12 bit SAR ADC data using DMA very easily.USB HID intermediate with Keyboard, we have done burning of the program on Pasco throughPSoC-5/ Miniprog 3 from PC to PSoC-5board and as the output we can see the output on board in the form of light in LED by providing the input from Keyboard. PSoC-5is a less time consuming device.
We started from basics projects .With the help of these basic projects we got enough information to achieve our main applications. Firstly we have done "LED with PWM" project, thus we put some changes on it by changing the connections and as a result we glow the LED on board. We also have done ADC to LCD display and other basic project was Capsense. By doing these projects we got familiar with the working of cypress kit.
This Kit is very attractive and main thing that it is very interesting. On performing the work on kit, we felt more curiosity for doing work on it. By the use of this kit we can also make our own IC.

REFRENCES

[1] Architecture and Programming of PSoC-5microcontroller by Predrag Micakovic.
[2] Designs Guide to the Cypress PSoC-5(Embedded Technology) by Robert Ashby.
[3] CY8CKIT-001-PSOC-5DEVELOPEMENT KIT GUIDEuDoc. # 001-48651 Rev. *D January 5, 2011.
[4] Anu MD,Lakshmi Natarajan , CY8C38xx/CY8C55xx on software version of PSoC-5Creator for PSoC-53 &PSoC-55-"Getting Started with DMA".
[5] Using DMA with high performance peripherals to Maximize the system performance by John Manjine.
[6] Anu MD,Anup Mohan "PSoC-53 &PSoC-55 for ADC data Buffering using DMA".
[7] USB HID intermediate with Keyboard using CY8C38xx, CY8C55xx on PSoC-53/PSoC-55 by Robert Murphy.
[8] Robert Murphy "PSoC-53 &PSoC-55 for USB Fundament
[9] RtrASSoc: an adaptable superscalar reconfigurable system-on-chip. The simulator Silva, J.L. ; Costa, R.M. ; Jorge,G.H.R.System-on-Chip for Real-Time Applications, 2003.

[10] The Method of data exchange between high performance PWM modulator and MCU by Maxim, D. ; Volkov, A.G. ; Makarov, D.V. Micro/Nanotechnologies and Electron Devices (EDM), 2012 IEEE 13th International Conference Publication Year: 2012.

[11] A human-robot interface using vision-based eye gaze estimation system Dong Hyun Yoo ; Jae Heon Kim ; Do Hyung Kim ; Myung Jin Chung Intelligent Robots and Systems, 2002. IEEE/RSJ International ConferenceYear: 2002.

[12] Microcontroller implementation of a voice command recognition system for human-machine interface in embedded systemsBernal-Ruiz, C. ; Garcia-Tapias, F.E. ; Martin-del-Brio, B. ; Bono-Nuez, A. ; Medrano-Marques, N.J Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference in 2005.

[13] DMA controller design using self-clocked methodology Aghdasi, F. ; Bhasin, A.AFRICON, 2004. 7th AFRICON Conference in Africa, 2004.

[14] A Case Study to Track High Value Stillages using RFID for an Automobile OEM and its Supply Chain in the Manufacturing IndustryAghdasi, F. ; Bhasin, A. AFRICON, 2004. 7th AFRICON Conference in Africa