AN2094

PSoC® 1 - Getting Started with GPIO

Author: Meenakshi Sundaram R
Associated Project: Yes
Associated Part Family: CY8C24x23A, CY8C24x94, CY8C21x34, CY8C20x34, CY8C21x23, CY8C21x45, CY8C22x45, CY8C27x43, CY8C28xxx, CY8C29x66, CY7C64215, CYWUSB6953
Software Version: PSoC® Designer™ 5.2 SP1
Related Application Notes: None

If you have a question, or need help with this application note, contact the author at msur@cypress.com

AN2094 discusses general-purpose input and output (GPIO) relevant topics such as GPIO drive modes, shadow registers, and GPIO interrupts to get started with PSoC 1 GPIOs. This document also provides few tips and briefs of the other resources associated with PSoC 1 GPIOs.

Contents

Introduction

General-purpose input and output (GPIO) is a very critical part of any microcontroller unit (MCU) as they form the bridge between the external world and the MCU. The type and nature of this external world bridge depends on the end application. For instance, an ADC requires a GPIO to be an analog pin whereas an I^2C or SPI digital communication block requires the same GPIO to be digital. In order to properly setup this external world bridge, you need to know not only the end application but also the GPIO system of the MCU that is used. PSoC like any other controller has its own GPIO system. This application note discusses the application specific parameters of the GPIO system. Detailed technical overview of the system can be found in the respective device technical reference manual (TRM) under General Purpose I/O chapter of PSoC Core section.

The topics discussed in this application note include:

- **GPIO Drive modes:** Covers usage of each mode and dynamic reconfiguration of the drive modes in firmware with an example.
- **Shadow registers:** Describes the significance and usage of GPIO shadow registers with an example.
- **GPIO Interrupts:** covers GPIO interrupts in PSoC 1 with a simple LED toggle example using interrupts.

This document assumes that the reader is familiar with PSoC Designer IDE.

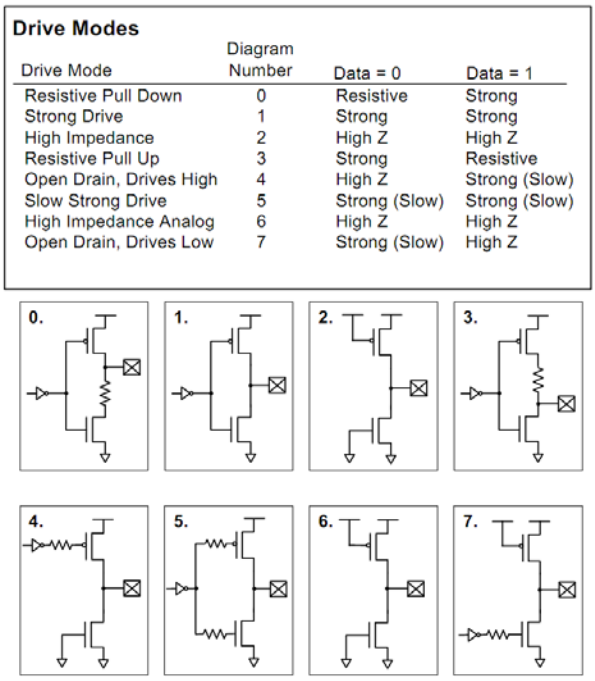
GPIO Drive modes

PSoC 1 offers 8 drive modes, as described in Table 1, in which a GPIO pin can be configured. Figure 1 shows the GPIO cell configuration for each of the drive modes. Figure 2 shows the GPIO cell inside PSoC 1. For complete detail on the GPIO cell structure refer device specific TRM.

Table 1. Drive Mode Details

S.No	Drive mode	Description	Application
1	High-Z	High impedance digital input mode. In this mode, the pin acts as a digital input as there will not be any drives on the pin internally. Writing either '1' or '0' in this mode from PSoC will not have any effect.	Digital input interfacing a Strong drive pin (Pin connected to V_{DD} for '1' and GND for '0')
2	High-Z analog	High impedance analog mode. In this mode, the pin acts as an analog pin. Very similar to Digital High-Z except that in this mode there will not be any digital buffer or Schmidt trigger (as shown in Figure 2) to convert the signal to a '1' or '0' to be readable by PSoC.	Analog input
3	Open drain-high (ODH)	In this mode, writing a '1' drives the pin to V_{DD} while a '0' is high-impedance state.	To provide ODH interface. This mode works in conjunction with a pull-down resistor at the receiving end.
4	Open drain-low (ODL)	In this mode, writing a '0' drives the pin to GND while a '1' is high-impedance state. Also known as Open-Collector mode	To provide ODL or Open collector interface. Example – I^2C pins. This mode works in conjunction with a pull-up resistor at the receiving end.
5	Strong	In this mode, writing a '1' drives the pin to V_{DD} and a '0' drives it to GND.	Digital Output pin.
6	Pull-down	In this mode, writing a '1' drives the pin to V_{DD} and a '0' drives it to GND through a resistor (5.6 K approximately)	As an interface to ODH input or a switch connected to V_{DD}. Can be used as an output to interface LEDs in current sink mode.
7	Pull-up	In this mode, writing a '1' drives the pin to V_{DD} through a resistor (5.6 K approximately) and a '0' drives it to GND. For detail, refer device datasheet.	As an interface to ODL input like tach output from motors or a switch connected to GND. Can be used as an output to interface LEDs in current source mode.
8	Strong Slow	This mode is similar to Strong mode but the slope of the output is slightly controlled so that high harmonics are not present when the output switches.	As a digital output with reduced harmonic effect.

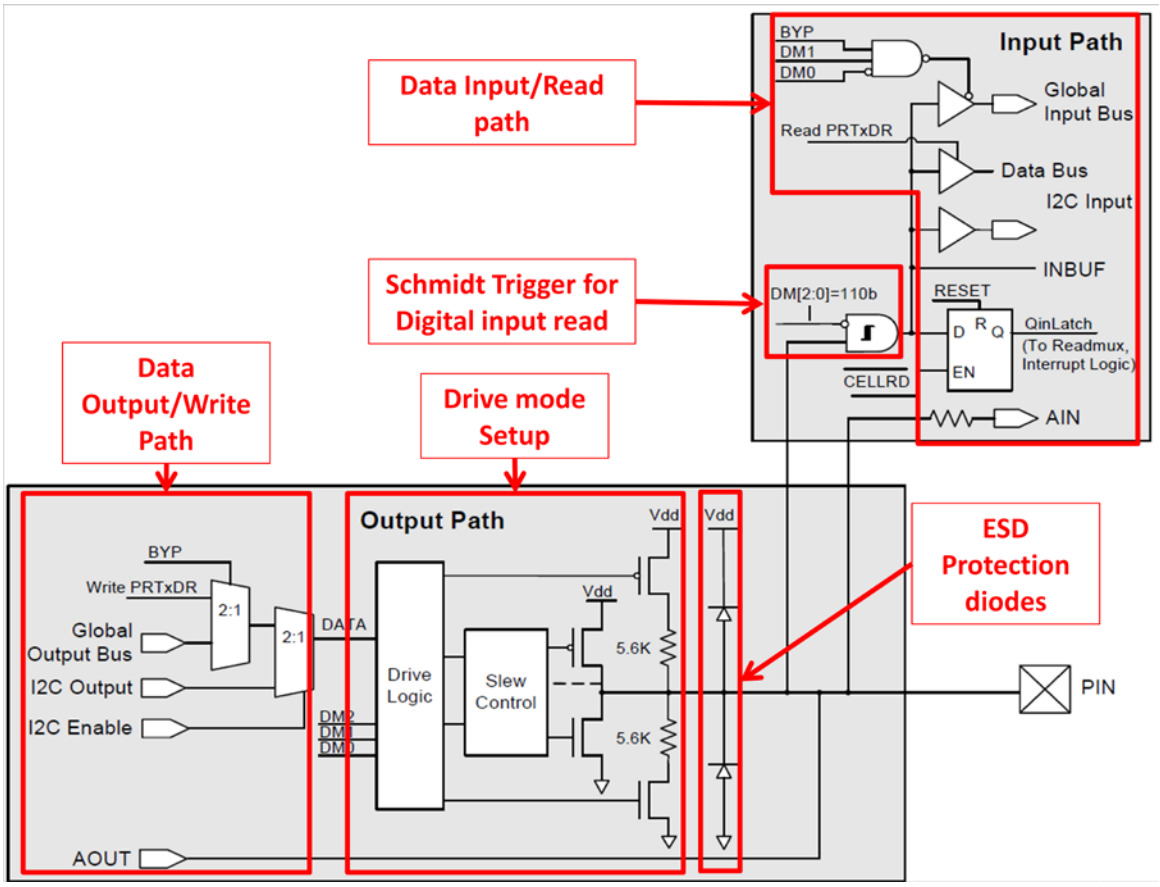
Figure 1. Drive Mode Configuration Details



Drive Modes

Drive Mode	Diagram Number	Data = 0	Data = 1
Resistive Pull Down	0	Resistive	Strong
Strong Drive	1	Strong	Strong
High Impedance	2	High Z	High Z
Resistive Pull Up	3	Strong	Resistive
Open Drain, Drives High	4	High Z	Strong (Slow)
Slow Strong Drive	5	Strong (Slow)	Strong (Slow)
High Impedance Analog	6	High Z	High Z
Open Drain, Drives Low	7	Strong (Slow)	High Z

Figure 2. GPIO Cell Structure in PSoC 1



GPIO configurations can be done in two ways. The first method is defining the configuration as part of initialization in PSoC Designer's device editor. This method is useful when the pin configuration is fixed all the time. The other method is to configure the pin in firmware. This method enables flexibility of configuring the GPIO during runtime.

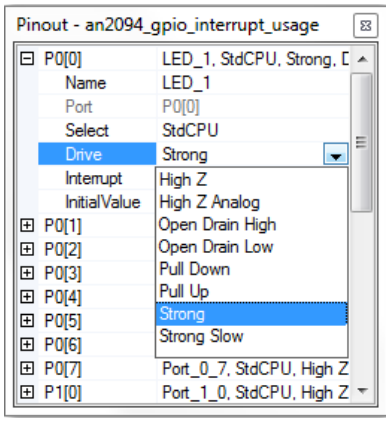
Device Editor Configuration

I/O pins may be configured by using the Pinout View mode of the Device Editor. Inside the Pinout View mode, a table appears in the lower left corner of the PSoC Designer interface. The table is shown in Figure 3.

The various fields shown in Figure 3 are described as below:

1. The "**Name**" field shows the name of the pin. The user can rename the pin to make its purpose more obvious. Renaming the pin generates Macros for the pin, such as Pin data register, Pin mask, and drive mode registers in *PSoCGPIOINT.inc* and *PSoCGPIOINT.h* files are explained later in the section Naming a Pin.

   Figure 3. PSoC Designer Pinout Window ("Drive" List)



2. The "**Port**" field shows the physical mapping of the pin. This field is not editable.

   The "**Select**" field configures some of the special behaviors of pins, which are as follows:

   a. **AnalogInput:** Only Port 0 and Port 2 have additional analog input and analog output options. AnalogInput gets analog signals from the outside world and connects to the analog column input MUX or to PSoC blocks directly. For example, if you use an ADC, you must configure at least one of the pins as AnalogInput to get analog signals from the outside world.

   b. **AnalogOutputBuf:** Only Port 0 has additional analog output options.

   c. **Default:** The global bus is not connected and the drive strength is High-Z Analog.

   d. **StdCPU:** Normal I/O through the port. This is controlled by the CPU[1].

   e. **Global_IN, Global_OUT:** Global inputs and outputs provide capability to route clock and data signals to the digital PSoC blocks. If you configure a pin as a Global_IN (input) or Global_OUT (output), then that pin can talk to the digital blocks. For example, if the Global_IN is selected, then this selection connects that particular pin to the Global_INPUT bus. This bus is then used as an input to the digital PSoC blocks.

   f. **AnalogMuxInput**[2]: Enables connection of the pin to Analog mux bus, which can be routed to various analog blocks inside PSoC.

   g. Apart from the previously mentioned pin types, there are pins that have special features, and are listed. For example, P1[0] and P1[1] have XtalOut and XtalIn, P1[4] has ExtSysClk, P1[5] and P1[7] have I2C_SDA and I2C_SCL, and so on.

3. The "**Drive**" field sets the drive mode of the pin as explained in Table 1 and Figure 1.

4. The "**Interrupt**" field in the Pin out window sets the interrupt type of the pins. Pins may have rising edge, falling edge or both interrupts. These are discussed in the section GPIO Interrupts.

5. The "**Initial Value**" field in the Pin out window sets the initial output value of the pin at startup. This value is imposed by populating the pin's data register during the execution of automatically generated boot code, and can be overridden by the user at runtime.

6. **AnalogMUXBus**[2]: Enables/Disables the connection of pin to AMUX bus in Chip editor. For doing the same in firmware refer section Analog Mux (AMUX) Bus Control Register (MUX_CRx) on Page 13).

Code Level Configuration

Another method, as mentioned earlier, to configure I/O pins is to directly modify the drive mode registers in the firmware[3] using assembly or C language. This method allows you to configure I/O ports dynamically during program execution.

There are three registers for each port that sets the drive mode of every port pin. They are PRTxDM0, PRTxDM1, and PRTxDM2 registers, where 'x' is the port number. One

[1] Only in this mode CPU can control the pin's output state i.e., register write to the port data register will take effect on the pins.

[2] Only available in CY8C21x34, 21x45, 22x45, 24x94, 28xxx family of devices. Used in conjunction with AnalogMuxBus field (available in the above mentioned devices)

[3] If the pin configuration is fixed, then user authored code is not required to configure the pins. PSoC Designer automatically generates startup code to configure the pins according to the settings in the Device Editor.

bit from these three registers each together configures a particular pin. For example, bit0 of PRT0DM0, PRT0DM1, and PRT0DM2 controls the P0[0] drive mode. The configuration details are explained in Table 2.

Table 2. Drive Mode Register Values

PRTxDM2[n]	PRTxDM1[n]	PRTxDM0[n]	Drive Mode
0	0	0	Resistive Pull-down
0	0	1	Strong Drive
0	1	0	High Impedance – Digital
0	1	1	Resistive Pull-up
1	0	0	Open Drain – High
1	0	1	Slow Strong drive
1	1	0	High Impedance – Analog
1	1	1	Open Drain - Low

In Table 1, ‘x’ corresponds to the port number and ‘n’ corresponds to the bit in the drive mode register and the port pin to be configured. For instance, to configure Port 0 Pin 1 as resistive pull-down, clear bit ‘1’ of PRT0DM0, PRT0DM1, and PRT0DM2 registers. Refer technical reference manual of the device for a more detailed overview.

Important point to be noted is all the PRTxDM0 and PRTxDM1 registers are in Register Bank 1 (refer TRM for more details on register banks) where as all the PRTxDM2 registers are in Register bank 0. This knowledge is required to use the drive mode registers in assembly, where the user has to select the register bank before accessing the registers, Code 1. In C, compiler takes care of bank assignments based on the register used.

Code 1:

```
M8C_SetBank1
or   reg[PRT2DM0], 0x20
and  reg[PRT2DM1], ~0x20
M8C_SetBank0
and  reg[PRT2DM2], ~0x20
```

In the assembly example Code 1, the first line is a call to the M8C_SetBank1 macro, which switches the register bank to ‘1’. This is done because PRT2DM0 and PRT2DM1 are in register bank 1. Then using the “OR” instruction and using a mask of 0x20, bit 5 of PRT2DM0 register is set. Then using “AND” instruction and a mask of inverse of 0x20, bit 5 of the PRT2DM1 register is cleared. Using M8C_SetBank0, it is switched back to register bank ‘0’, and using the “AND” instruction and a mask of inverse of 0x20, bit 5 or the PRT2DM2 register is cleared. The “OR” and “AND” instructions are read, modify, or write instructions. The content of the register is first read, a “OR” or “AND” operation is done on the value and then the result is written back to the same register. With this method, particular bits are modified without affecting the others.

Code 2:

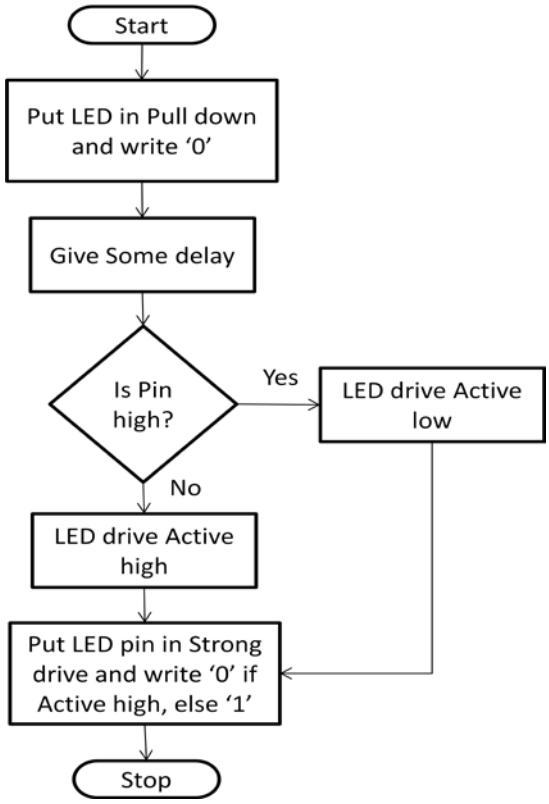
```
PRT2DM0 |= 0x20;
PRT2DM1 &= ~0x20
PRT2DM2 &= ~0x20;
```

In C, the code (Code 2) becomes much simpler as the switching of the banks is taken care of by the C compiler. The “bitwise AND” (&=) or the “bitwise OR” (|=) must be used with the corresponding masks on the registers.

Example 1: Detecting LED drive mode

In systems, LEDs are usually sinked through GPIOs to turn them ON. In certain systems instead of sinking, GPIOs source current to LEDs to turn them ON. Though the LED implementation is known when designing the system, there might be cases where you may want to upgrade/update the design without changing the firmware – say you included a sourcing GPIO LED design and found that the GPIO is not capable of sourcing enough current or you move to a LED with higher current rating and change the design to LED sink mode but you want the device to adjust itself depending on the mode the LEDs are connected to the GPIOs. This example lets you add that feature to your design, where you find out the mode in which the LED is connected to a GPIO pin and then turn ON/OFF the LED accordingly. The example utilizes the dynamic drive mode reconfiguration capability of PSoC to implement this. The following flow chart explains how LED drive mode is detected internally.

Figure 4. Detecting LED Drive Mode Algorithm



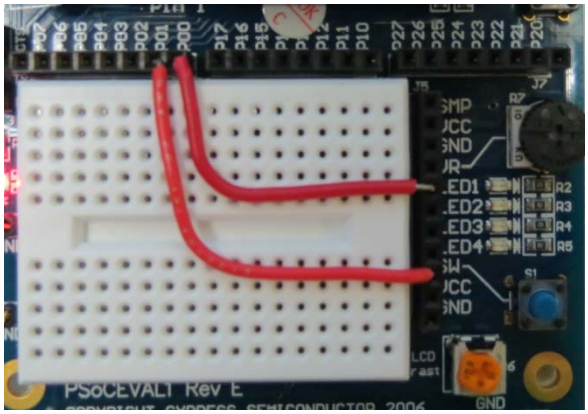
Hardware Requisites

- CY3210 – PsoCEval1 with 28 Pin CY8C29466-24PXI PDIP PSoC 1.
- Spare LED and 1 kΩ resistor – for checking LED sink mode
- CY3217 – MiniProg1
- Connecting wires

Test Procedure

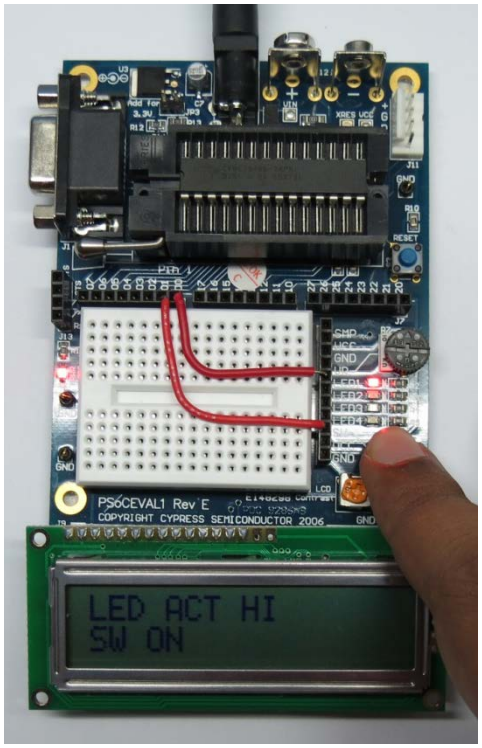
1. Insert CY8C29466-24PXI device into the 28 pin PDIP socket provided in the CY3210 board
2. Program the device using MiniProg1 with "AN2094_GPIO_DM_Reconfig.hex" available in the root directory of AN2094_GPIO_DM_Reconfig project attached with this application note
3. Connect P0_1 to SW and P0_0 to LED1 as shown in Figure 5.

Figure 5. Wire Connections for Example 1



4. Connect character LCD to J9 header in CY3210 board
5. Remove JP3 in CY3210 kit for 5 V operation
6. Power the board using MiniProg1 or a 5 V DC adapter.
7. LCD row 0 should display "LED ACT HI" and when SW is pressed LED1 glows and LCD row 1 displays "SW ON" as in Figure 6.
8. Similarly, if you wire up a spare LED in Sink mode via a 1 kΩ resistor and connect it to P0_0, the LCD will display "LED ACT LOW" and LED ON/OFF follows SW ON/OFF.

Figure 6. Example 1 Output



Shadow Registers

Reading and Writing to a Port

After configuring I/O ports, the PORT DATA registers are used to write or read data. PORT DATA registers are 8 bits wide. The bytes show the value read from pins of a particular port or the content of these registers is written directly to the port.

Port 0 Data Register (PRT0DR, Address = Bank 0, 00h)

Port 1 Data Register (PRT1DR, Address = Bank 0, 04h)

Port 2 Data Register (PRT2DR, Address = Bank 0, 08h)

Port 3 Data Register (PRT3DR, Address = Bank 0, 0Ch)

Port 4 Data Register (PRT4DR, Address = Bank 0, 10h)

Port 5 Data Register (PRT5DR, Address = Bank 0, 14h)

Port 6 Data Register (PRT6DR, Address = Bank 0, 18h)

Port 7 Data Register (PRT7DR, Address = Bank 0, 1Ch)

To write to a particular port pin, use the corresponding mask and bitwise "AND" or "OR" operation. For example, to set and clear P0[0] (Code 3 for ASM and Code 4 for C) :

Code 3:

```
or reg[PRT0DR], 0x01 ; Set P0[0]
and reg[PRT0DR], ~0x01 ; Clear P0[0]
```

Code 4:

```
PRT0DR |= 0x01; // Set P0[0]
PRT0DR &= ~0x01; // Clear P0[0]
```

To read from a port pin, read the PRTxDR register and use the corresponding bit mask. For example, to check the status of P0[1] and update an LED on P0[0] (Code 5 and Code 6):

Code 5:

```
mov A, reg[PRT0DR]
and A,0x02
jnz PinHigh
; Code to process Pin cleared state
or reg[PRT0DR], 0x01 ; Set P0[0]
PinHigh:
; Code to process Pin set state
and reg[PRT0DR], ~0x01 ; Clear P0[0]
```

Code 6:

```
if (PRT0DR & 0x02)
{
   // Code to process Pin Set state
   PRT0DR |= 0x01; // Set P0[0]
}
else
{
   // Code to process Pin cleared state
   PRT0DR &= ~0x01; // Clear P0[0]
}
```

Use of Shadow Registers

In many designs, same port can have an output pin (LED pin) as well as an input pin (Switch input – pull-up/down mode). In such designs, instruction that is used to update the output pin might latch the input pin permanently to a '1' or '0'.

For instance consider the following scenario; there is a switch input on P0_1, which is configured in pull-down mode (the switch is between V_{DD} and the Pin). And there is a LED output on P0_0, which follows the switch state on P0_1 and the pin is configured as a strong drive (output pin). Now let us assume the switch is pressed, so LED needs to be turned ON. The read-modify-write instructions shown in Code 6 will do the following:

1. Read – PRT0DR = x x x x x x 1 0 (Bit 0 = 0 → LED Off; Bit 1 = '1' → as Switch pressed)
2. Modify – (PRT0DR | 00000001) = x x x x x x 1 1
3. Write – PRT0DR = x x x x x x 1 1 (Bit 0 = 1 → LED ON; sets Bit 1 which connects the pin to V_{DD} internally as writing '1' to the pin in Pull-down drive mode connects it to V_{DD})
4. It will be observed that even after the switch is released, LED never turns OFF because to the device the switch is always ON (connected to V_{DD}) irrespective of the actual switch state outside. Further switch presses will not be recognized by the device.

To overcome this scenario, a variable called shadow register is used for every such ports (having input and output combination). When using a shadow register, all the writes to the pin happens through this variable and this variable should be initialized in such a way that input pins are maintained properly all the time. For instance, the value for a pull-up or ODL input pin is set to '1' in this register and a pull-down or ODH input is set to '0', so whenever a read-modify-write happens on other output pins the input pin states are preserved. Now a read-modify-write depicted above becomes:

1. Read – Port_0_Data_Shade = x x x x x x 0 0 (Bit 0 = 0 → LED off; Bit 1 = '0' as initialized, this not the data from the port directly)
2. Modify – (Port_0_Data_Shade | 00000001) = x x x x x x 0 1
3. Write - Port_0_Data_Shade = x x x x x x 0 1
4. Write to Port – PRT0DR = Port_0_Data_Shade

So the same Code 6 with shadow registers becomes:

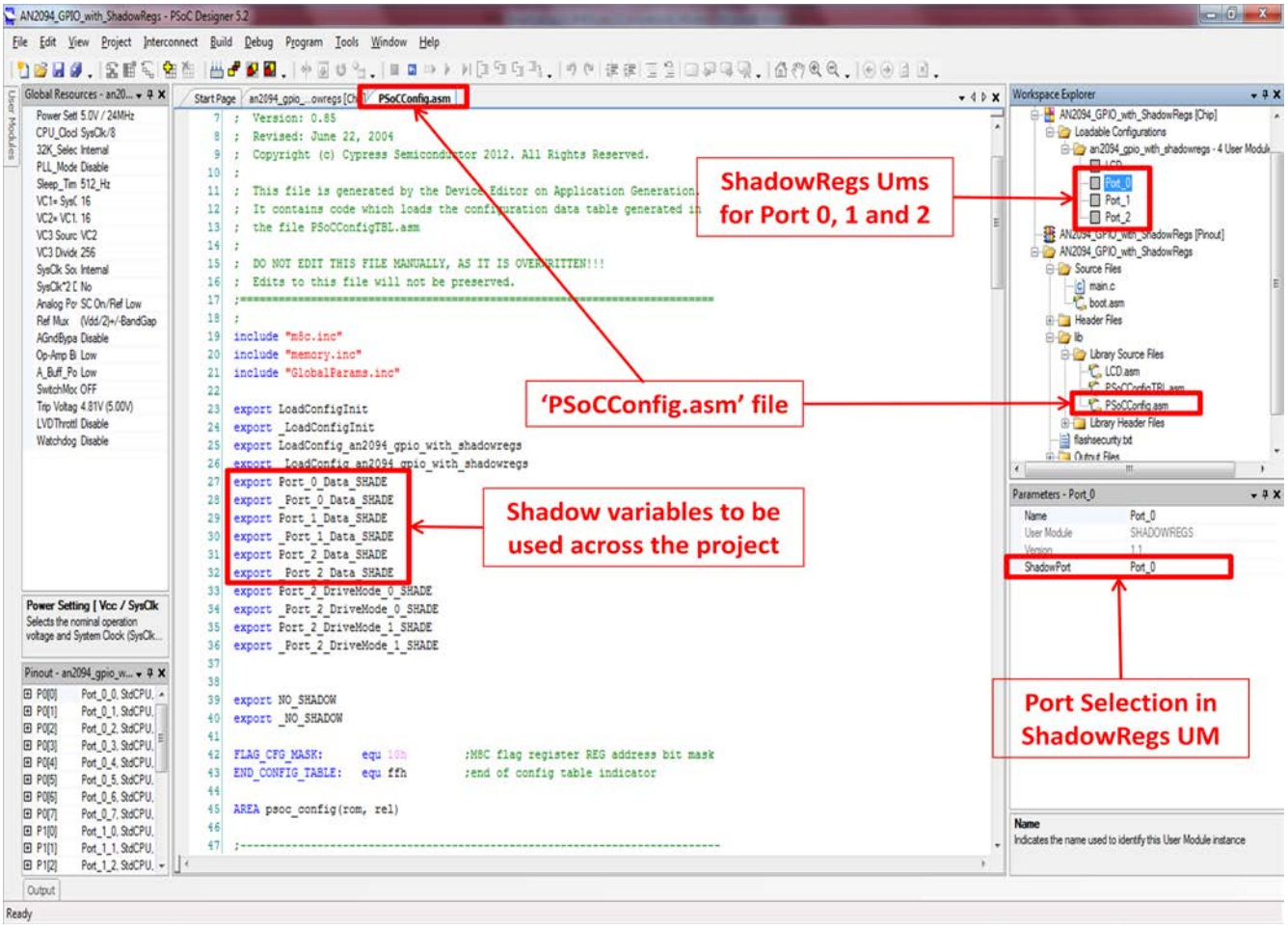
Code 7:

```
// Use 'extern' when using ShadowRegs UM
extern BYTE Port_0_Data_Shade;

// Using shadow variables in code
if(PRT0DR & 0x02)
{
      Port_0_Data_Shade |= 0x01;
```

```
        PRT0DR = Port_0_Data_Shade;
}

else
{
        Port_0_Data_Shade &= ~0x01;
        PRT0DR = Port_0_Data_Shade;
}
```

In this method, care should be taken to make sure all the read-modify-write to the port happens through the shadow register, also considering user module (UM) read/writes like TX8SW, which switches the TX pin in software. In order to make sure they happen through shadow registers and are synchronized, the ShadowRegs UM available under “Misc Digital” UM category can be used. By placing this UM and assigning a port to it creates a variable called Port_x_Data_SHADE in “*PSoCConfig.asm*” file in the “Library source files” directory (Figure 7), where ‘x’ is the port number. Now you can use this variable across files by importing it using ‘extern BYTE Port_x_Data_SHADE’. By doing read-write-modify through this variable in your code will make sure that input pins do not get stuck and the port data is synchronized across files (Code 7).

Figure 7. Shadow Variable Location in PSoC Designer



Example 2: Use of Shadow Registers

To demonstrate the importance and use of shadow registers a simple setup using the CY3210-PSoCEval1 board is created as in Figure 9. In this example, there is a provision in hardware to enable/disable shadow register feature on power up. When P0_2 is connected to V_{DD} during power-up, shadow registers are disabled and when it is connected to GND, shadow registers are enabled. This example will demonstrate the scenario explained in the section Use of Shadow Registers– where same port has an input switch and output LED.

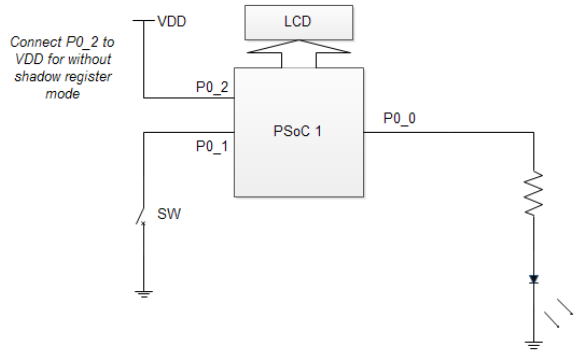
Hardware Requisites

- CY3210 – PSoCEval1 with 28 Pin CY8C29466-24PXI PDIP PSoC 1
- CY3217 – MiniProg1
- Connecting wires

Test Procedure

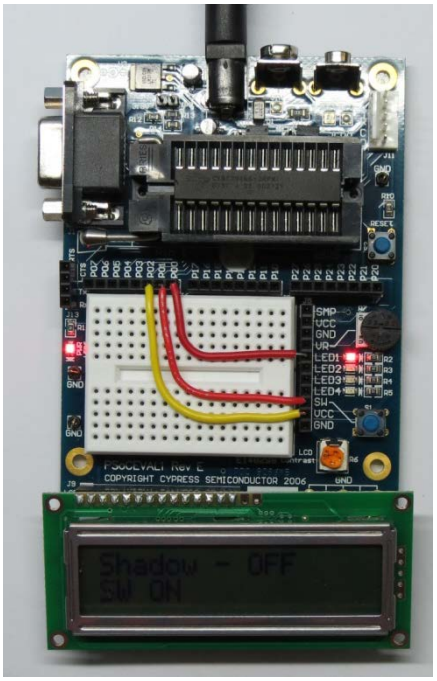
1. Insert CY8C29466-24PXI device into the 28 pin PDIP socket provided in the CY3210 board
2. Program the device using miniprog1 with "AN2094_GPIO_with_ShadowRegs.hex" available in the root directory of AN2094_GPIO_with_ShadowRegs project attached with this application note.
3. Connect character LCD to J9 header in CY3210 board
4. For testing the project without shadow variables, connect P0_0 to LED1, P0_1 to SW and P0_2 to V_{DD}, Figure 8.

   Figure 8. Pin Connections for Example 2 without Shadow Registers



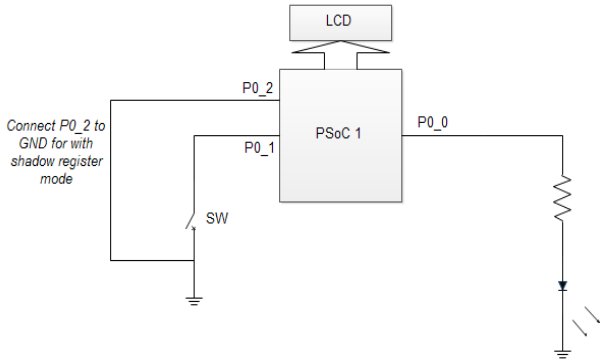
5. Remove JP3 in CY3210 kit for 5 V operation
6. Power the board using MiniProg1 or a 5 V DC adapter.
7. Now press SW once and release, and you will observe LCD row 1 displaying "SW ON" all the time and LED1 will be ON. This is because there is no shadow variable used Figure 9.

   Figure 9. Example 2 Output without Shadow Registers



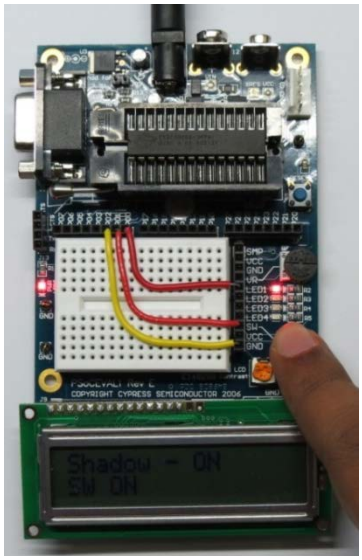
8. Now power off the board, and Connect P0_2 to GND Figure 10

   Figure 10. Pin Connections for Example 2 with Shadow Registers



9. Power ON the board.
10. LCD row 0 will display "Shadow - ON" and SW press will work as expected Figure 11.

Figure 11. Example 2 Output with Shadow Registers



GPIO Interrupts

Interrupts are another important part of the GPIO system, especially when there is a need to process a digital signal with priority. The interrupt system in PSoC is a vast topic, to know about all the available interrupts in PSoC and their priorities refer the respective device TRMs. This topic will discuss the GPIO interrupts alone. Each GPIO pin in PSoC can be configured to generate an interrupt on rising edge, falling edge or change from previous read event. This way of configuring the event, which triggers the GPIO interrupt can be done in two different ways – one way is to configure interrupts in the PinOut window (as explained in Device Editor Configuration on page 4, point 4) and another in firmware. The registers associated with configuring and enabling GPIO interrupts are tabulated in Table 3.

Table 3. GPIO Interrupt Configuration Related Registers

Associated Register	Description	Values
PRTxIE	Interrupt Enable register for each port 'x'. Setting or clearing a bit in this register enables/disables the interrupt on that particular pin.	1 – Enable 0 – Disable interrupt
PRTxIC0 and PRTxIC1	Interrupt control registers – used to set the type of event that triggers the interrupt.	Table 4
INT_MSK0	Interrupt enable mask register 0	Bit 5 of the register is Global GPIO interrupt enable/disable bit. INT_MSK0_GPIO macro can be used as a mask to enable/disable the 5th bit in the register.
INT_CLR0	Posted interrupt read and clear register 0	Bit 5 of the register is GPIO posted interrupt bit. Writing a 0 to the bit clears any posted GPIO interrupts. If the bit reads 1 then there is a posted GPIO interrupt or write a 1 to post a GPIO interrupt through Software[4]

Table 4. Interrupt Control Registers Setting

PRTxIC1 [n]	PRTxIC0 [n]	Interrupt Type	Description
0	0	Disabled	Interrupt disabled
0	1	Falling Edge	Interrupt on 1 to 0 transition of the input signal
1	0	Rising Edge	Interrupt on 0 to 1 transition of the input signal
1	1	Change from Read	Change in pin's current state with respect to last read value of PRTxDR[n]

In Table 4, 'x' denotes the port number and 'n' denotes the bit of the register/pin of the port to be configured.

While using GPIO interrupts, it should be kept in mind that there is only one ISR associated with all the pin interrupts. It is the responsibility of the user to check at the beginning of the ISR, which pin caused the GPIO ISR. In the example Code 8 and Code 9, interrupts on Pin 0 and Pin 1 are enabled. Pin 0 interrupt is configured as a falling edge ISR whereas P0_1 is configured as 'change from read'. GPIO interrupt is enabled using "M8C_EnableIntMask" macro and finally global interrupt is enabled. In the GPIO ISR, there are 'if' control structures placed to check which of the pin(s) caused this instance of the ISR and the data is processed accordingly.

Code 8:

```
/* P0_0 configured as falling edge
interrupt */
```

[4] Write 0 AND ENSWINT = 0 Clear posted interrupt if it exists.
Write 1 AND ENSWINT = 0 No effect.
Write 0 AND ENSWINT = 1 No effect.
Write 1 AND ENSWINT = 1 Post an interrupt for general purpose inputs and outputs (pins).

ENSWINT bit is bit 7 of INT_MSK3 register.

```
PRT0IC0 |= 0x01;
PRT0IC1 &= ~0x01;

/* P0_1 configured as Change from Read
interrupt */
PRT0IC0 |= 0x02;
PRT0IC1 |= 0x02;

/* Enable P0_1 and P0_0 interrupts */
PRT0IE |= 0x03;

/* Enable GPIO interrupts */
M8C_EnableIntMask(INT_MSK0, INT_MSK0_GPIO);

/* Enable Global interrupts */
M8C_EnableGInt;
```

Code 9:

```
/* Function prototype for GPIO ISR */
#pragma interrupt_handler GPIO_ISR

/* GPIO ISR in C where GPIO interrupts are
processed */
void GPIO_ISR(void)
{
/* variable to have a copy of prev P0_1
value for change from read comparison */
static BYTE port0_prevValue;

/* Check if interrupt because of P0_0
falling edge:
First condition checks for P0_0 to be '0'
Second condition checks if there is change
in present and previous values */
if(((PRT0DR & 0x01) == 0) && ((PRT0DR ^
port0_prevValue) == 0x01))
{
/* Process P0_0 interrupt */
}

/* Check if interrupt because of P0_1
change from read */
if ((PRT0DR ^ port0_prevValue)==0x02)
{
/* Process P0_1 interrupt */
}

/* Store values of P0_0 and P0_1 for next
ISR */
port0_prevValue = PRT0DR & 0x03;
}
```

Do's and Don'ts while Using Interrupts

- Global interrupt enable bit must be set using "M8C_EnableGInt" macro
- Drive mode of the GPIO should not be set to high-impedance analog or else interrupt will never occur for that particular pin.
- When a GPIO interrupt is configured as 'Change from Read', the Pin value should be read using PRTxDR register for the next interrupt to occur. Only a change in the last read value of PRTxDR and current Pin state triggers this interrupt.
- The ISR function should be defined using "#pragma interrupt_handler" (as shown in Code 9) directive for the ISR to properly execute and return control.
- The ISR function defined in C or ASM should be placed in the 'boot.tpl' file at the GPIO ISR location provided with an 'ljmp' instruction as shown in Figure 13.

Example 3: LED Toggling Using Interrupts

To demonstrate the use of GPIO interrupts a simple LED toggle algorithm is implemented in this example. Rising edge interrupt is enabled on the pin which is connected to a switch. In the ISR, a simple LED toggling is done. An oscilloscope can be connected on the LED pin along with a square wave signal input to the switch pin; it can be observed that the frequency of the signal at the LED pin will be half of the frequency given to the switch pin.

Hardware Requisites

- CY3210 – PsoCEval1 with 28 Pin CY8C29466-24PXI PDIP PSoC 1.
- CY3217 – MiniProg1
- Connecting wires

Test Procedure

1. Insert CY8C29466-24PXI device into the 28 pin PDIP socket provided in the CY3210 board
2. Program the device using MiniProg1 with "AN2094_GPIO_Interrupt_Usage.hex" available in the root directory of AN2094_GPIO_Interrupt_Usage project attached with this application note.
3. For testing the project, connect P0_0 to LED1, P0_1 to SW as in Figure 5.
4. Remove JP3 in CY3210 kit for 5 V operation
5. Power the board using MiniProg1 or a 5 V DC adapter.
6. Press SW and see LED1 toggling on each press.
7. Instead of a switch, a square wave signal[5] can also be connected to the P0_1 and it can be observed that the signal on P0_0 is half the frequency of the input signal at P0_1, Figure 12.

[5] The maximum frequency of the input signal depends on the CPU clock, GPIO interrupt latency and GPIO pin capacitance.

Figure 12. Example 3 Output

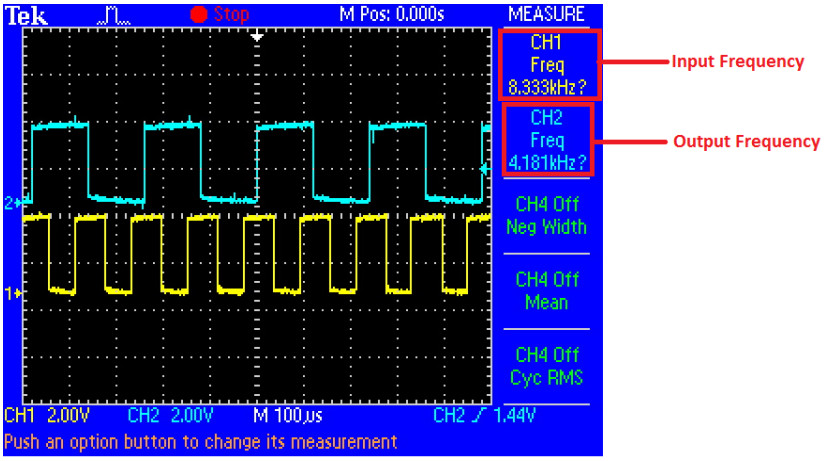
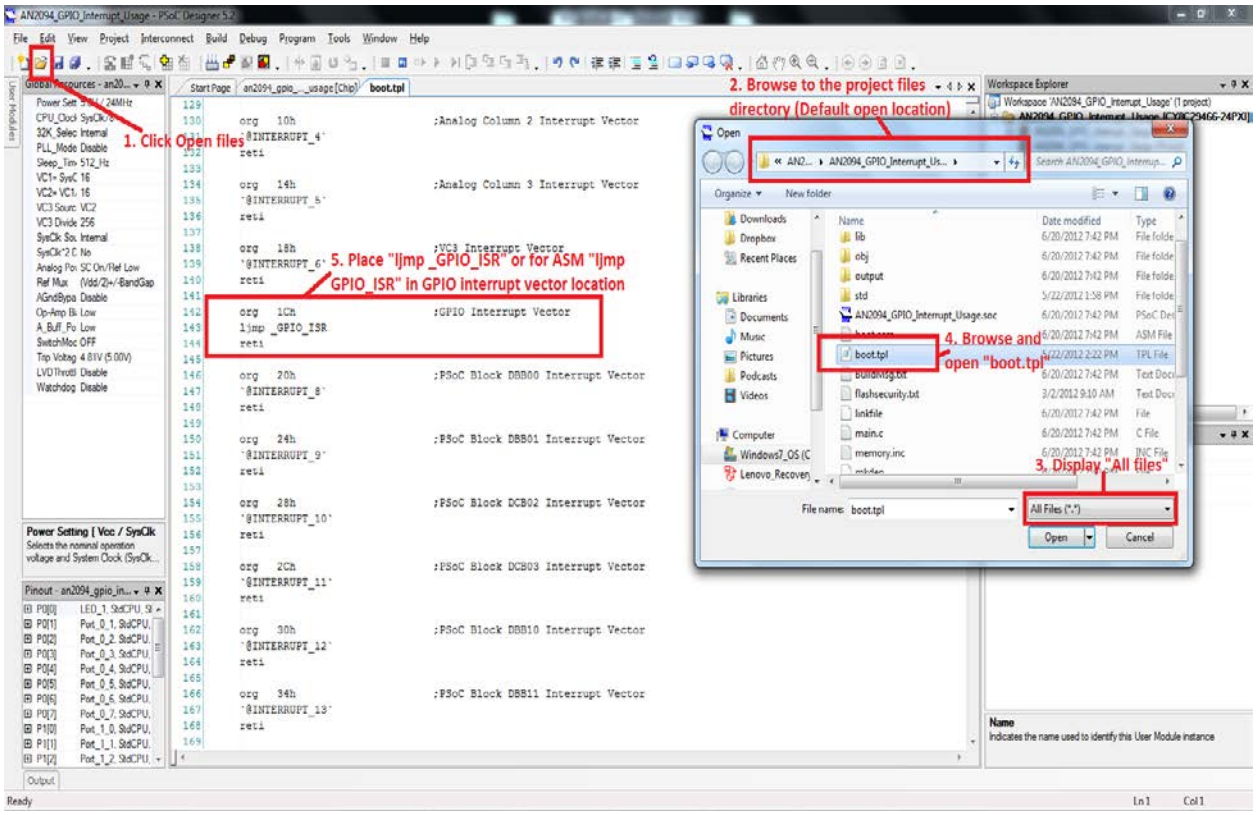


Figure 13. GPIO ISR Location in 'boot.tpl'



Other GPIO Resources and Tips

GPIO Global Select Register

The PRTxGS register decides if a GPIO pin is under the control of the CPU or is connected to the Global In or Global Out bus. When the bit corresponding to the pin in PRTxGS register is cleared, the pin can be controlled by the CPU by writing to the PRTxDR register. When the bit in PRTxGS register is set and the drive mode of the pin is set to high impedance (analog or digital), it is routed to Global In bus, which serves as inputs to various digital blocks available. If the bit is set and when the drive mode is other than high impedance, the connection to Global Out bus is enabled, which connects outputs from various digital blocks and comparators. Refer this section for setting the global select register in device editor.

Analog Mux (AMUX) Bus Control Register (MUX_CRx)[6]

MUX_CRx register enables/disables connection of a GPIO pin to the internal analog mux bus. This analog mux bus is then available as input to various analog blocks inside PSoC. For instance, setting the bit '0' of MUX_CR1 connects P1_0 to AMuX bus. Refer respective device TRM for more details on the AMUX settings and routing.

Naming a Pin

Each pin can have a unique name, which can be assigned as explained in Device Editor Configuration. By giving a name to a pin, PSoC Designer will automatically generate macros for all the registers associated with pin in the *PSoCGPIOINT.h* for C files and *PSoCGPIOINT.asm* for ASM files easy access. The macro list includes macros for:

- Port data register (PRTxDR)
- Port Drive mode registers (PRTxDMy)
- Port interrupt enable register (PRTxIE)
- Port interrupt setup registers (PRTxICy)
- Port global select register (PRTxGS)
- Pin Mask

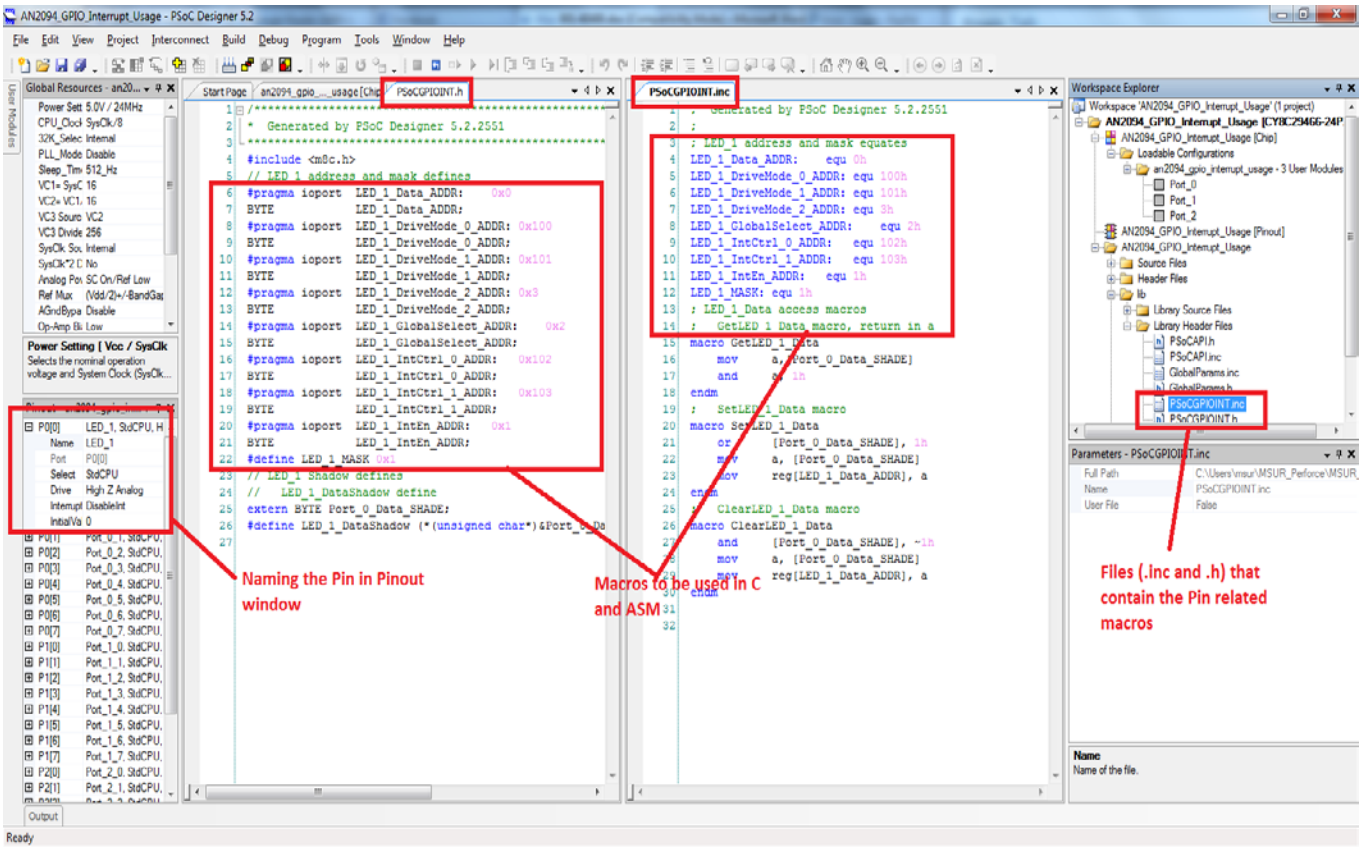
Table 5 gives an overview of the macros generated in '*PSoCGPIOINT.h*' file for use in C files and '*PSoCGPIOINT.inc*' for use in ASM files. These macros can be directly used in any function to access that pin related settings and information.

Table 5. Macros Associated with a Named Pin

Pin Name	LED_1
Port data register	LED_1_Data_ADDR
Port drive mode 0 register	LED_1_DriveMode_0_ADDR
Port drive mode 1 register	LED_1_DriveMode_1_ADDR
Port drive mode 2 register	LED_1_DriveMode_2_ADDR
Port Global Select register	LED_1_GlobalSelect_ADDR
Port interrupt enable register	LED_1_IntEn_ADDR
Port interrupt control 0 register	LED_1_IntCtrl_0_ADDR
Port interrupt control 1 register	LED_1_IntCtrl_1_ADDR

[6] Only available in CY8C21x34, 21x45, 22x45, 24x94, 28xxx family of devices.

Figure 14. Pin Related Macros



Registers and their Associated Register Banks

There are two banks available in the PSoC 1 register map. So it is required for the user to know and understand which bank each register belongs, for him to modify or read its content in assembly. In C code, compiler takes care of the register bank settings. To change the register bank to bank 0 in ASM, M8C_SetBank0 macro can be used. Similarly for bank 1 you can use M8C_SetBank1 macro. Table 6 provides the register bank details for each of the GPIO register discussed in this application note.

Table 6. GPIO Related Registers and their Register Banks

Register	Register bank
PRTxDR	0
PRTxDM0	1
PRTxDM1	1
PRTxDM2	0
PRTxIE	0
PRTxGS	0
PRTxIC0	1
PRTxIC1	1
INT_MSK0	0
INT_CLR0	0
MUX_CRx	1

Document History

Document Title: AN2094 - PSoC® 1 – Getting Started with GPIO

Document Number: 001-40480

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1532004	SFV	11/13/2007	Recatalogued application note.
*A	1778285	SFV	12/18/2007	Associated Project files zipped with source document.
*B	2188526	MAXK	06/05/2008	Corrected Table 1. Drive Mode Configuration. (project files zipped with source files)
*C	3181445	MAXK	02/24/2011	Adapted example project to operate on CY3210-EVAL1 board. Updated firmware for PSoC Designer 5.1 SP1. General information and readability updates. Template changes.
*D	3283657	MAXK	06/15/2011	No Technical updates. Document title updated.
*E	3665015	MSUR	07/03/2012	Changed title to Getting started with GPIO. Covered relevant topics to get started with GPIOs. Updated the associated project and template. Complete rewrite.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com