AN2094

PSoC® 1 - Getting Started With GPIO

Author: Meenakshi Sundaram R

Associated Part Family: CY8C24x23A, CY8C24x94, CY8C21x34, CY8C20x34, CY8C21x23, CY8C21x45, CY8C22x45, CY8C27x43, CY8C28xxx, CY8C29x66, CY7C64215, CYWUSB6953

Related Application Notes: None

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/AN2094.

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

Contents

1 Introduction

The general-purpose input and output (GPIO) is a critical part of any microcontroller (MCU) as it is the bridge between the external world and the MCU. The type and nature of this bridge to the external world depends on the end application. PSoC has powerful and flexible general-purpose I/O (GPIO) pins that provide more features than traditional MCUs.

For instance, an ADC requires a GPIO to be an analog input, whereas an I^2C or SPI digital communication block requires the same GPIO to be digital. To correctly set up this bridge to the external world, you need to know the end application and the GPIO system of the MCU that is used. PSoC, similar to any other microcontroller, has its own GPIO system.

This application note discusses the application-specific parameters of the GPIO system. A detailed technical overview of the system is available in the General-Purpose I/O chapter of the PSoC Core section in the respective device Technical Reference Manual (TRM).

Topics discussed in this application note include:

- **GPIO drive modes:** Discusses the types of drive modes available in PSoC 1, usage of each drive mode, and the procedure to dynamically reconfigure the drive mode in firmware with an example.
- **Shadow registers:** Describes the significance and the usage of GPIO shadow registers with an example.
- **GPIO interrupts:** Explains GPIO interrupts in PSoC 1 with a simple LED toggle example using interrupts.

This document assumes that you are familiar with the PSoC Designer™ IDE.

2 Getting Started

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device for your design, and to help you to quickly and effectively integrate the device into your design. For a comprehensive list of resources, see the knowledge base article How to Design with PSoC® 1, PowerPSoC®, and PLC – KBA88292. Following is an abbreviated list for PSoC 1:

- Overview: PSoC Portfolio, PSoC Roadmap
- Product Selectors: PSoC 1, PSoC 3, PSoC 4, PSoC 5LP
- In addition, PSoC Designer includes a device selection tool.
- Application notes: Cypress offers a large number of PSoC application notes covering a broad range of topics, from basic to advanced level. Recommended application notes for getting started with PSoC 1 are:
  - AN75320 - Getting Started with PSoC® 1.
  - AN2094 - PSoC® 1 - Getting Started with GPIO.
  - AN74170 - PSoC® 1 Analog Structure and Configuration with PSoC Designer™
  - AN2041 - Understanding PSoC® 1 Switched Capacitor Analog Blocks
  - AN2219 - PSoC® 1 Selecting Analog Ground and Reference

Note: For CY8C29X66 devices related application notes, click here.

- Development Kits:
  - CY3210-PSoCEval1 supports all PSoC 1 mixed-signal array families, including automotive, except the CY8C25/26xxx devices. The kit includes an LCD module, potentiometer, LEDs, and breadboarding space.
  - CY3214-PSoCEvalUSB features a development board for the CY8C24x94 PSoC device. Special features of the board include USB and CapSense development and debugging support.

Note: For CY8C29X66 devices related development kits, click here.

The MiniProg1 and MiniProg3 devices provide interfaces for flash programming and debug.

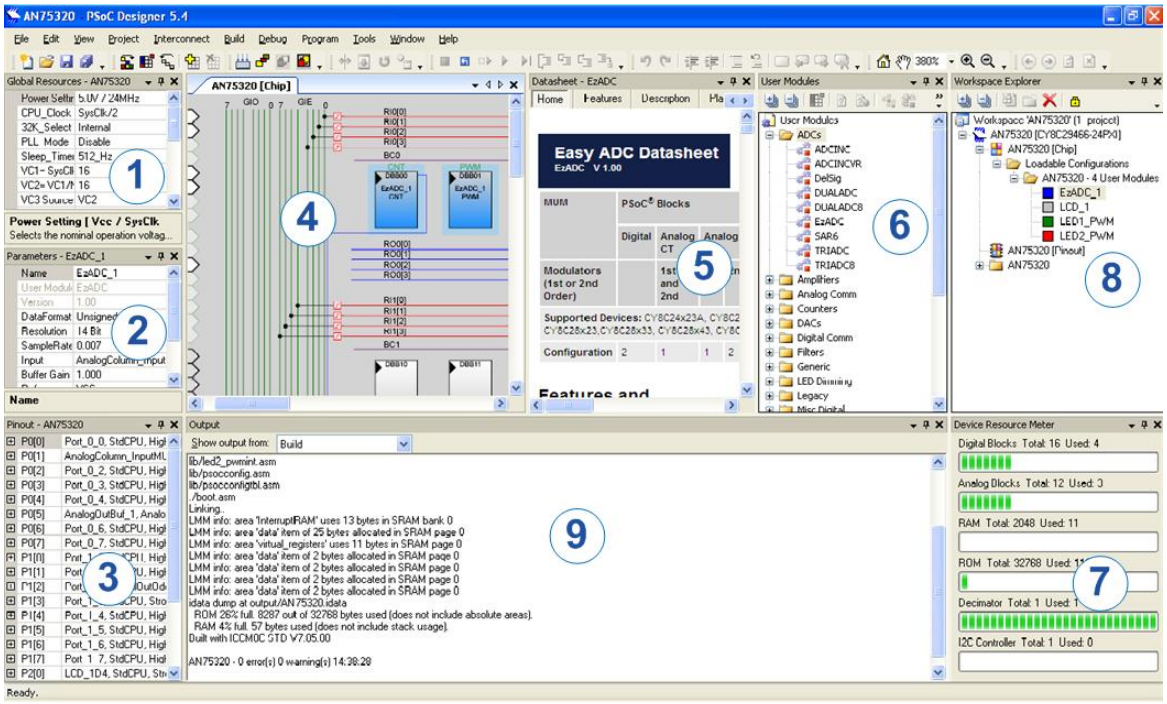
2.1 PSoC Designer

PSoC Designer is a free Windows-based Integrated Design Environment (IDE). Develop your applications using a library of pre-characterized analog and digital peripherals in a drag-and-drop design environment. Then, customize your design leveraging the dynamically generated API libraries of code. Figure 1 shows PSoC Designer windows. **Note:** This is not the default view.

1. Global Resources – all device hardware settings.
2. Parameters – the parameters of the currently selected User Modules.
3. Pinout – information related to device pins.
4. Chip-Level Editor – a diagram of the resources available on the selected chip.
5. Datasheet – the datasheet for the currently selected UM
6. User Modules – all available User Modules for the selected device.
7. Device Resource Meter – device resource usage for the current project configuration.
8. Workspace – a tree level diagram of files associated with the project.
9. Output – output from project build and debug operations.

Note: For detailed information on PSoC Designer, open PSoC Designer IDE, go to **Help** > **Documentation**, open the “Designer Specific Documents” folder, and open the “IDE User Guide.pdf” document.

Figure 1. PSoC Designer Layout



2.2 Code Examples

The following webpage lists the PSoC Designer based code examples. These code examples can speed up your design process by starting you off with a complete design, instead of a blank page, and also show how PSoC Designer User Modules can be used for various applications.

http://www.cypress.com/documentation/code-examples/psoc-1-code-examples

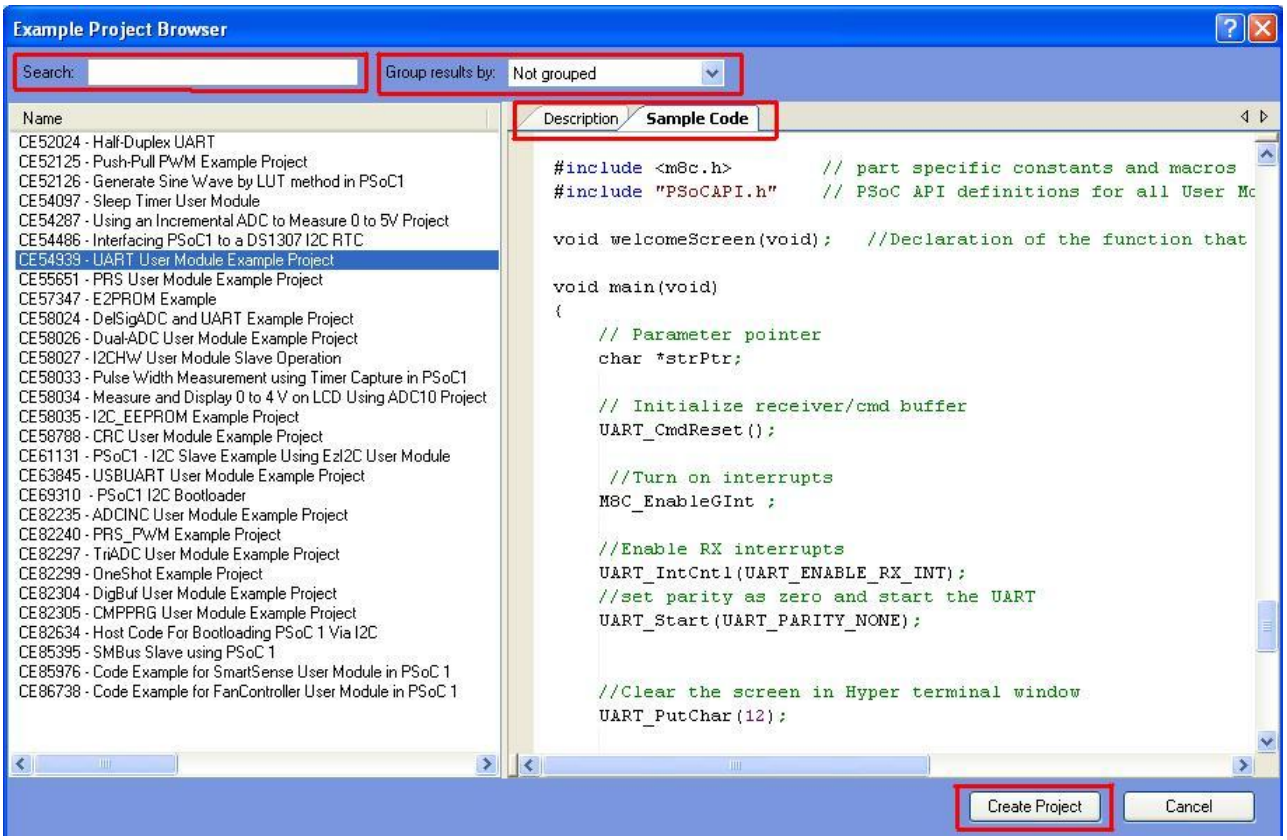
To access the code examples integrated with PSoC Designer, follow the path **Start Page > Design Catalog > Launch Example** *Browser* as shown in Figure 2.

Figure 2. Code Examples in PSoC Designer

In the Example Projects Browser shown in Figure 3, you have the following options:

- Keyword search to filter the projects.
- Listing the projects based on category.
- Review the datasheet for the selection (on the Description tab).
- Review the code example for the selection. You can copy and paste code from this window to your project, which can help speed up code development, or
- Create a new project (and a new workspace if needed) based on the selection. This can speed up your design process by starting you off with a complete, basic design. You can then adapt that design to your application.

Figure 3. Code Example Projects, with Sample Codes



2.3 PSoC Designer Help

Visit the PSoC Designer home page to download the latest version of PSoC Designer. Then, launch PSoC Designer and navigate to the following items:

- **IDE User Guide**: Choose **Help > Documentation > Designer Specific Documents > IDE User Guide.pdf**. This guide gives you the basics for developing PSoC Creator projects.
- **Simple User module Code Examples**: Choose **Start Page > Design Catalog > Launch Example Browser**. These code examples demonstrate how to configure and use PSoC Designer User modules.
- **Technical Reference Manual**: Choose **Help > Documentation > Technical Reference Manuals**. This guide lists and describes the system functions of PSoC 1 devices.
- **User module datasheets**: Right-click a User module and select “Datasheet.” This datasheet explains the parameters and APIs of the selected user module.
- **Device Datasheet**: Choose **Help > Documentation > Device Datasheets** to pick the datasheet of a particular PSoC 1 device.
- **Imagecraft Compiler Guide**: Choose **Help > Documentation > Compiler and Programming Documents > C Language Compiler User Guide.pdf**. This guide provides the details about the Imagecraft compiler specific directives and Functions.

2.4 Technical Support

If you have any questions, our technical support team is happy to assist you. You can create a support request on the Cypress Technical Support page.

If you are in the United States, you can talk to our technical support team by calling our toll-free number: +1-800-541-4736. Select option 8 at the prompt.

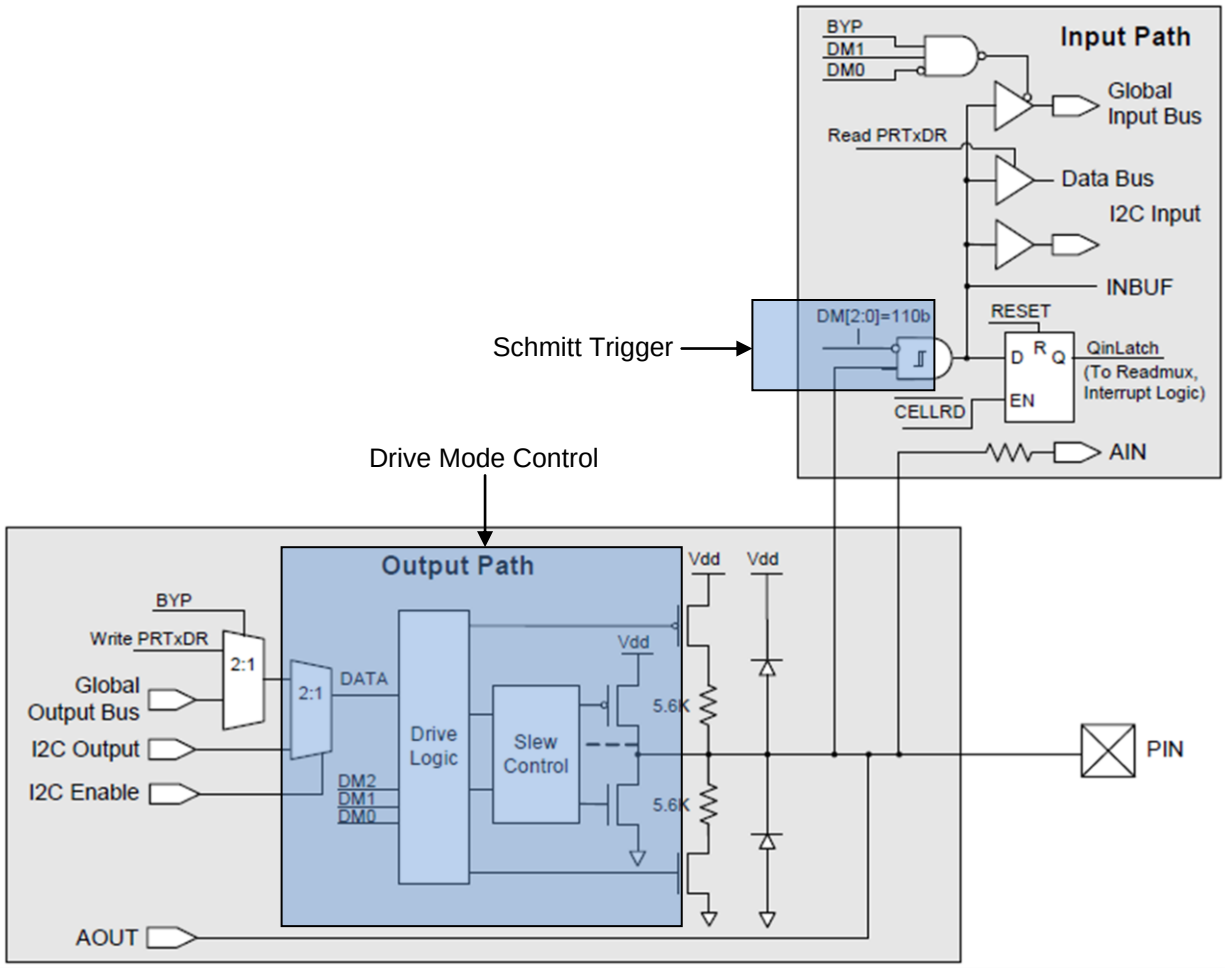
You can also use the following support resources if you need quick assistance.

- Self-help
- Local Sales Office Locations

3 GPIO Architecture

Figure 4 shows the architecture of the PSoC 1 GPIO.

Figure 4. GPIO Cell Architecture in PSoC 1



There are two major parts in the GPIO cell: the Input path and the Output path.

The input path has the Schmitt trigger circuit that acts as an interface between the pin and the MCU for digital signals. The output of the Schmitt trigger connects to the MCU's data bus, global input bus, I^2C input, interrupt controller, and so on. The pin is also directly connected to the internal analog bus that interfaces to the analog blocks inside the PSoC device.

The output path has the drive mode select logic that can configure the pin to one of eight drive modes. The input to the drive mode select comes from the internal data bus, which includes the data register, I^2C bus, and global output bus (connected to output of the digital block). There is also a connection to the internal analog bus (on select pins), which connects to the analog output buffer.

4 GPIO Drive Modes

PSoC 1 offers eight drive modes to configure a GPIO pin, as shown in Table 1. Figure 5 shows the GPIO cell configuration for each of the drive modes.

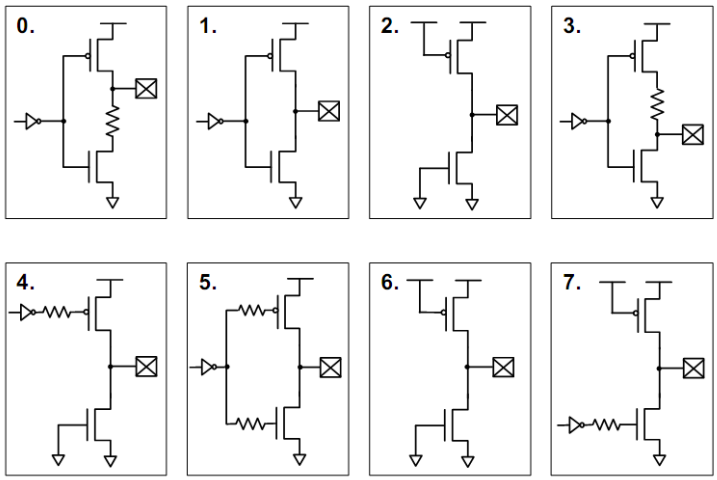
Table 1. Drive Mode Details

Sl. No.	Drive mode	Description	Application
1	High-Z	High-impedance digital input mode. In this mode, the pin acts as a digital input. Writing either '1' or '0' to the pin in this mode will have no effect.	Digital input interfacing to a signal source with a strong drive, pull-up, or pull-down, open-collector with external pull-up resistor, or open-emitter with external pull-down resistor.
2	High-Z analog	High-impedance analog mode. In this mode, the pin acts as an analog pin. Similar to digital High-Z, except that in this mode the digital input circuit (the Schmitt trigger in Figure 1) in the GPIO cell is disabled. If you are using the pin as a digital input, make sure that you select "HighZ", not "HighZ Analog" drive mode.	Analog input and on select pins, as analog input and output.
3	Open drain-high (ODH)	In this mode, writing a '1' drives the pin to V_{DD} while a '0' is high-impedance state. This is similar to an open-emitter configuration.	To provide an open-emitter interface with external pull-down resistor. This drive mode implements a wired OR connection.
4	Open drain-low (ODL)	In this mode, writing a '0' drives the pin to GND while writing a '1' results in a high-impedance state. This is similar to an open-collector configuration.	To provide an open-collector interface with external pull-up resistors. This implements a wired AND connection. Example – I^2C pins.
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 kΩ approximately).	As an interface to a signal with open-emitter output or to a switch connected to V_{DD}. It can be used as an output to interface LEDs in the current sink mode.
7	Pull-up	In this mode, writing a '1' drives the pin to V_{DD} through a resistor (5.6 kΩ approximately), and a '0' drives it to GND.	As an interface to a signal with open-collector output such as tachometer signal from motors or to a switch connected to GND. It can be used as an output to interface LEDs in the current source mode.
8	Strong Slow	This mode is similar to the 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 radiated interference.

Figure 5. 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

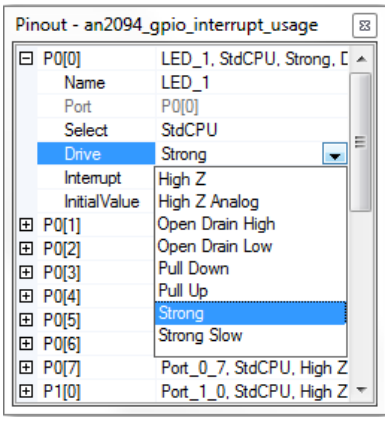


The GPIO can be configured in two ways. The first method is to define the configuration as part of the initialization in PSoC Designer's Device Editor. This method is useful when the pin configuration is fixed all the time. The other method is to configure the pin in the firmware. This method gives the flexibility of configuring the GPIO during runtime.

4.1 Device Editor Configuration

The 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 position of this window may vary according the arrangement of windows). The table is shown in Figure 6.

Figure 6. PSoC Designer Pinout Window ("Drive" List)



The various fields shown in Figure 6 are:

1. The **Name** field shows the name of the pin. You can rename the pin to make its purpose more obvious. Renaming the pin generates macros for the pin, such as the pin data register, pin mask, and drive mode registers in *PSoCGPIOINT.inc* and *PSoCGPIOINT.h* files. This is explained in detail in the section Naming a Pin.
2. The **Port** field shows the physical mapping of the pin. This field is not editable.
3. The **Select** field configures some of the following special behaviors of pins:
   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:** Depending on the device family, some of the Port 0 pins are connected to internal analog output buffers.
   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] through the PRTxDR data register.
   e. **Global_IN, Global_OUT:** Global inputs and outputs provide the 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 used as an input to the digital PSoC blocks.
   f. **AnalogMuxInput**[2]: Enables you to connect the pin to the analog mux bus, which you can route to various analog blocks inside the PSoC device.

      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.
   g. The **Drive** field sets the drive mode of the pin as explained in Table 1 and Figure 5.
   h. The **Interrupt** field in the Pinout window sets the interrupt type of the pins. The pins may have rising edge, falling edge, or both interrupts. See GPIO Interrupts.
   i. The **Initial Value** field in the Pinout 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.
   j. **AnalogMUXBus**[2]: Enables or disables the connection of the pin to the AMUX bus in the Chip Editor. To do the same in the firmware, see Analog Mux (AMUX) Bus Control Register).

4.2 Code-Level Configuration

Another method to configure I/O pins is to directly modify the associated registers in the firmware[3] using assembly or C language. This method allows you to dynamically configure I/O ports during program execution. The following registers control a GPIO:

1. **PRTxDR register**: This is the data register that controls the output state of a pin. Each port has an associated data register with each bit representing a pin. For example, PRT0DR controls Port0; Bit#0 of PRT0DR controls P0[0]. The state of a GPIO pin also can be read using the PRTxDR register. See Reading and Writing to a Port.
2. **PRTxDMx registers**: Each port has three registers to control the drive mode. The drive mode of the pins can be dynamically changed during run time by writing to these registers. See Modifying the Drive Mode of a GPIO pin.
3. **PRTxICx registers**: Each port has two registers that control the interrupt type of the pin (rising edge, falling edge, change in state and no interrupt). The interrupt type of a pin can be changed dynamically during run time by writing to these registers. See GPIO Interrupts.
4. **PRTxGS register:** This register is used to connect or disconnect the GPIO pin to the Global Input or Global Output bus. See GPIO Global Select Register.

[1] The CPU can control the pin's output state only in this mode; that is, the register write to the port data register will take effect on the pins.

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

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

5. **MUX_CRx register:** In devices that have the analog mux bus, this register is used to connect or disconnect a pin to the analog mux bus. Each port is represented by a register and each bit in the register controls the corresponding port pin. For example, Bit#0 of MUX_CR0 register controls P0[0].

4.3 Reading and Writing to a Port

When a port pin is disconnected from the global bus by clearing the corresponding bit in the PRTxGS register (by configuring the pin as StdCPU in the GPIO configuration window), and the drive mode of the pin is not *HighZ* or *HighZ Analog*, the state of the pin can be controlled by writing to the PRTxDR register.

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]:

In assembly:

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

In C:

```
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]:

In assembly:

```
    mov A, reg[PRT0DR]
       and A, 0x02
       jnz PinHigh

       ; Code to process Pin cleared
    state
       and reg[PRT0DR], ~0x01 ; Set P0[0]
       jmp Exit

PinHigh:
       ; Code to process Pin set state
       or reg[PRT0DR], 0x01 ; Clear P0[0]

Exit:
```

In C:

```
```

```
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]
}
```

If a port pin is given a meaningful name in the GPIO configuration window, the data register and the pin mask may also be accessed using the pin macros generated by PSoC Designer. See the Naming a Pin section.

4.4 Modifying the Drive Mode of a GPIO pin

Each port has three registers that sets the drive mode of every port pin. They are PRTxDM0, PRTxDM1, and PRTxDM2 registers, where 'x' is the port number. One bit from each of these three registers together configures a particular pin. For example, bit0 of PRT0DM0, PRT0DM1, and PRT0DM2 controls the P0[0] drive mode. Table 2 provides the configuration details.

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 2, '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 the PRT0DM0, PRT0DM1, and PRT0DM2 registers. Refer to the device TRM for more details.

You must note that all the PRTxDM0 and PRTxDM1 registers are in Register Bank 1 (refer to the TRM for more information about register banks), whereas all the PRTxDM2 registers are in Register Bank 0. This knowledge is required to use the drive mode registers in assembly, where you have to select the register bank before accessing the registers. In C, the compiler takes care of bank assignments based on the register used.

In Assembly:

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

In the assembly example, 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. Bit 5 of the PRT2DM0 register is set by using the "OR" instruction and a mask of 0x20. Next, bit 5 of the PRT2DM1 register is cleared by using an AND instruction and a mask of inverse of 0x20.

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, an OR or AND operation is done on the value, and then the result is written back to the same register. With this method, you can modify particular bits without affecting the others.

In C:

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

In C, the code is much simpler because 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.

If a port pin is renamed in the GPIO configuration window, the drive mode registers may also be accessed using the pin macros generated by PSoC Designer. See Naming a Pin.

The modification of GPIO drive mode is demonstrated in Example #1, which is available for download from the application note’s web page. See the Appendix for details of this example project.

5 Shadow Registers

5.1 Use of Shadow Registers

In many designs, the same port can have both input (for example, switch with pull-up or pull-down input) and output pins (for example, an LED). In such designs, the 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: A switch input on P0_1 is configured in pull-down mode (the switch is connected between V_{DD} and the pin). An LED output on P0_0 follows the switch state on P0_1 and the pin is configured as a strong drive (output pin). The following code accomplishes this.

```
if (PRT0DR & 0x02)
{
   // Switch pressed. Turn on the LED
   PRT0DR |= 0x01; // Set P0[0]
}
else
{
   // Switch released. Turn off the LED
   PRT0DR &= ~0x01; // Clear P0[0]
}
```

Now, assume the switch is pressed, so the LED needs to be turned on. The code "PRT0DR |= 0x01" is a read-modify-write instruction that does 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. In step #3, you can see that a '1' is written to P0[1], which will permanently change the state of this pin to 1. Now even if the switch is released, the data register will keep driving a '1' onto the pin and the code will always read a 1 on the switch input even if the switch is released.

To overcome this scenario, a variable called the shadow register is used for such ports (having an input and output combination).

When you use a shadow register, all the writes to the pin happens through this variable, and this variable should be initialized with the correct states for input pins. The value for a pull-up or Open Drain Low input pins should be set to '1' in this register and a pull-down or Open Drain High input should be set to '0'. When the state of an output pin has to be changed, the read-modify-write instruction must be performed on the shadow register and then the shadow register should be copied to the port data register. All reads will take place directly on the port data register. The following code implements the shadow register:

```
// 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;
       }
```

Now the read-modify-write operation becomes:

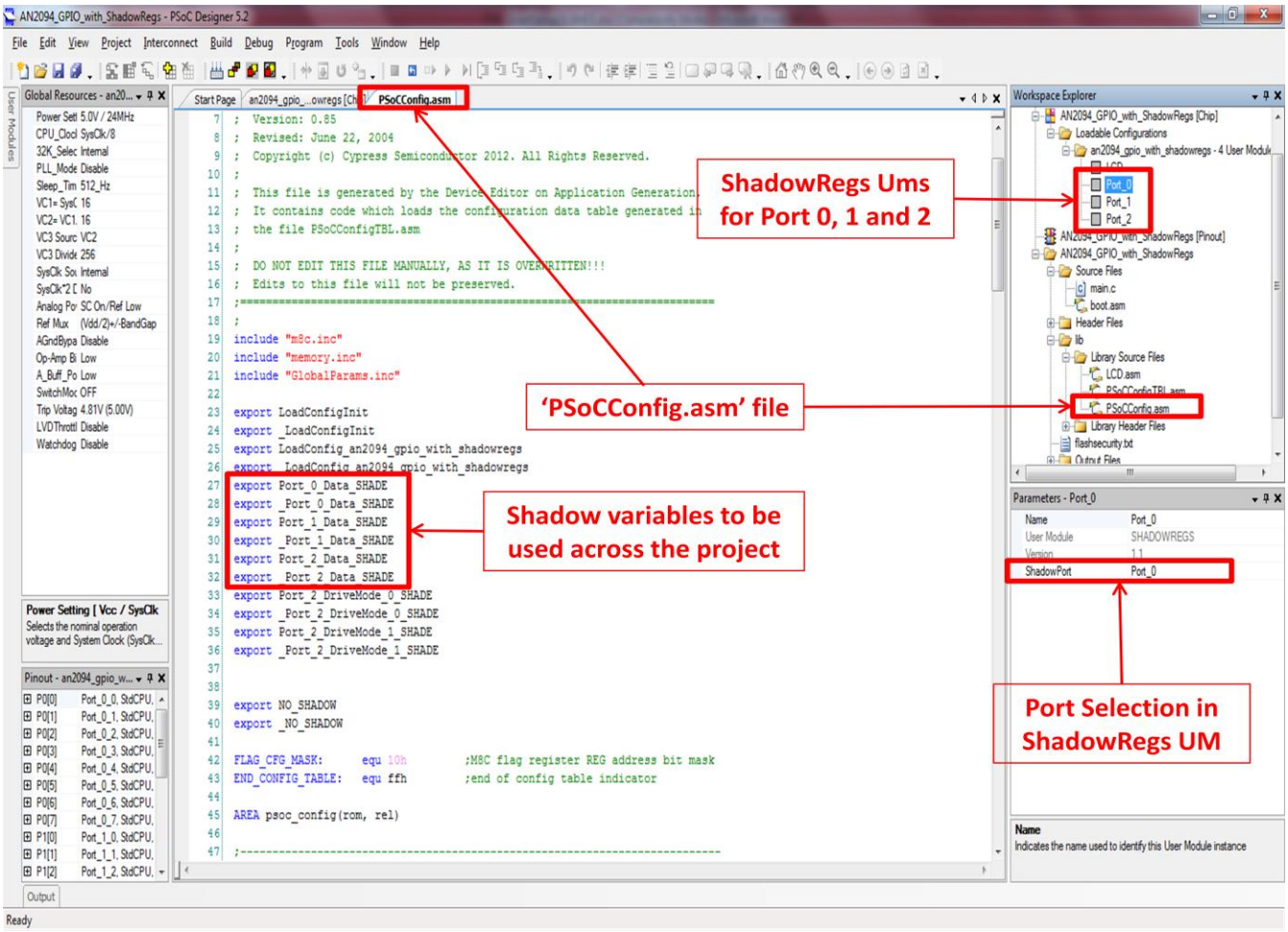
1. Read – Port_0_Data_Shade = x x x x x x 0 0 (Bit 0 = 0 → LED off; Bit 1 = '0' as initialized, this not the data from the port directly)
2. Modify – (Port_0_Data_Shade | 00000001) = x x x x x x 0 1
3. Write – Port_0_Data_Shade = x x x x x x 0 1
4. Write to Port – PRT0DR = Port_0_Data_Shade

PSoC Designer provides a user module (UM) called ShadowRegs, which is available under the Misc Digital user module category. By placing this UM in the project and assigning a port, a variable called Port_x_Data_SHADE is created in the *PSoCConfig.asm* file in the *Library source files* directory (See 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'.

User modules such as TX8SW, which manipulate the GPIO pins, will also use the shadow register and therefore having input pins mixed with output pins used by other user modules poses no problem.

When you rename a pin and place a shadow register in the port, an alias for the shadow register with the pin name is also created in *psocgpioint.h* and *psocgpioint.inc* files. See the Naming a Pin section.

Figure 7. Shadow Variable Location in PSoC Designer



The usage of shadow registers is demonstrated in Example #2, which is available for download from the application note's web page. See Appendix for details of this example project.

6 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 to the respective device TRMs. This section will discuss only the GPIO interrupts.

Each GPIO pin in PSoC can be configured to generate an interrupt on rising edge, falling edge, or change from a 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 the interrupts in the GPIO configuration window (see Device Editor Configuration) and the other is in firmware. The registers that are associated with configuring and enabling GPIO interrupts are listed 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 – rising edge, falling edge, or change from read.	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 fifth 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.

To implement a GPIO interrupt, do the following:

1. In the GPIO configuration window, set the type of interrupt for the pin. This can also be done in code by writing to the PRTxICx registers.
2. In the main application code, enable GPIO interrupt by setting bit 5 of the INT_MSK0 register. This can be done by using the M8C_EnableIntMask macro.
3. Enable the interrupt for the corresponding pin by writing to the PRTxIE register. If the interrupt type is selected in the GPIO configuration window, then PSoC Designer automatically sets the PRTxIE register bit during the boot-up process.
4. Write a C or assembly ISR for processing the interrupt. If the ISR is written in C, then declare the ISR using the #pragma interrupt_handler directive to tell the compiler that the function is an ISR.

[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.

5. Place the code to redirect the interrupt to the ISR function. This can be done in two ways:
   a. When the interrupt is enabled in the GPIO configuration window, PSoC Designer generates a library file called *psocgpioint.asm*. This file has a placeholder function PSoC_GPIO_ISR where the interrupt may be redirected.

      For example, if you have a C function called MyGpioIsr, place the code “ljmp _MyGpioIsr” (an underscore has to be added to the function name while calling a C function from assembly).

   b. Directly add the redirect instruction to the boot code. To do this, open the *boot.tpl* file in the project folder. This is the template file using which PSoC Designer generates the *boot.asm* file. In the *boot.tpl* file, in the GPIO interrupt vector, comment out the code “`@INTERRUPT_7`” and add the redirect instruction to your GPIO ISR. Save the *boot.tpl* file and generate application. Now the *boot.asm* file will have the redirect instruction to the GPIO ISR.

      Note that there is only one interrupt vector associated with GPIO interrupts. If interrupt is enabled on multiple pins, it is the responsibility of the application code to detect which pin caused the interrupt and process it accordingly.

      In the example code below, interrupts on P0_0 and P0_1 are enabled. The P0_0 interrupt is configured as a falling edge interrupt, whereas P0_1 is configured as ‘change from read’. The GPIO interrupt is enabled using the M8C_EnableIntMask macro, and finally global interrupt is enabled. In the GPIO_ISR, there are ‘if’ control structures placed to check which pin caused this instance of the ISR, and the data is processed accordingly.

Setting Up and Enabling GPIO Interrupt

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

Writing the GPIO ISR

```
/* 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 it was ‘1’ in the last ISR */
if(((PRT0DR & 0x01) == 0) && ((PRT0DR & 0x01) == 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;
}
```

6.1.1 Setting Up the Redirect to the ISR

Inside the *psocgpioint.asm* file:

```
;-----------------------------------------
;  FUNCTION NAME: PSoC_GPIO_ISR
;
;  DESCRIPTION: Unless modified, this implements only a null handler stub.
;
;----------------------------------------- PSoC_GPIO_ISR:

;@PSoC_UserCode_BODY@ (Do not change this line.)
;---------------------------------------
; Insert your custom code below this banner
;---------------------------------------
   ljmp _GPIO_ISR
;---------------------------------------
; Insert your custom code above this banner
;---------------------------------------
;@PSoC_UserCode_END@ (Do not change this line.)
   reti
```

Similarly, the following code shows the same implementation in assembly

Inside the *boot.tpl* file:

```
org   1Ch     ;GPIO Interrupt Vector
;`@INTERRUPT_7`
ljmp _GPIO_ISR
      reti
```

After making these changes, save the *boot.tpl* file and generate the application.

6.1.2 Setting Up and Enabling GPIO Interrupt

```
; P0_1 configured as change from read
; Interrupt

M8C_SetBank1
or reg[PRT0IC0], 0x02
or reg[PRT0IC1], 0x02

; Enable P0_1 interrupt
M8C_SetBank0
or reg[PRT0IE], 0x02

;Enable GPIO interrupts
M8C_EnableIntMask INT_MSK0, INT_MSK0_GPIO

;Enable Global interrupts
M8C_EnableGInt
```

Code 1: GPIO ISR

```
export MyGpioIsr

area text
;GPIO ISR in ASM where all GPIO ISRs ;are processed
GPIO_ISR:

;Preserve CUR_PP, X and A
push A
push X
mov A, reg[CUR_PP]
push A

;Change CUR_PP to port0PrevValue's Ram ;page (port0PrevValue can be defined ;in any
'.c' file included in the ;project as a global BYTE variable)
RAM_SETPAGE_CUR >_port0PrevValue

;Read PRT0DR into A
mov A, reg[PRT0DR]

;take a copy of PRT0DR into X for ;storing in port0PrevValue at the end
mov X, A

;XOR PRT0DR value in A and PRT0DR's ;prev value
xor A, [<_port0PrevValue]

;AND with 0x02 to read P0_1's state
and A, 0x02

;If zero flag is set, no change in ;state, so not P0_1 ISR
jz .NoP0_1_ISR
;
;Comes here if the XOR operation ;resulted a non zero result
;
;Process code for P0_1 ISR
;
.NoP0_1_ISR:
;
;Process code for other GPIO ISRs
;
RAM_SETPAGE_CUR >_port0PrevValue
mov [<_port0PrevValue], X

;Restore CUR_PP, X and A in order they ;were pushed
pop A
mov reg[CUR_PP], A
pop X
pop A
reti
```

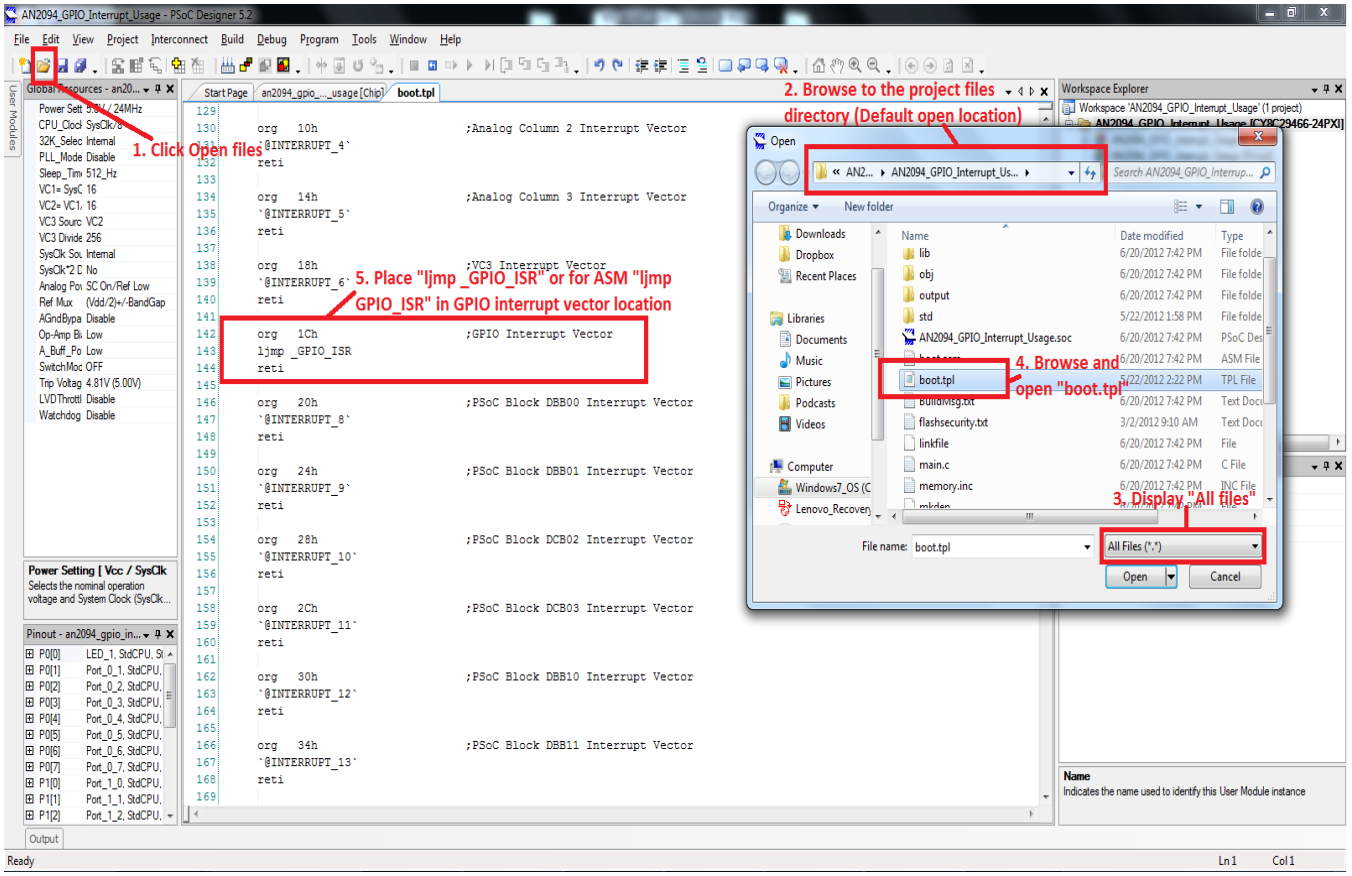
The method to redirect to the ISR remains the same as we used in the C example. In the ljmp instruction placed inside the *psocgpioint.asm* file or the *boot.tpl* file, the underscore before the MyGpioIsr is not needed.

Example #3 demonstrates the implementation of GPIO interrupts. You can download it from the application note's web page. See the Appendix for details of this example project.

6.2 Do's and Don'ts While Using Interrupts

- Global interrupt enable bit must be set using the M8C_EnableGInt macro.
- The 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 the 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 in C should be defined using the "#pragma interrupt_handler" directive for the ISR to properly execute and return control.
- The ISR function defined in the *asm* file should preserve the registers used and restore them before exiting. The function should use the RETI instruction to return instead of the normal RET instruction.
- The redirect to the ISR function defined in C or ASM may either be placed inside the *psocgpioint.asm* file or the *boot.tpl* file. The *boot.tpl* file and the location of the GPIO ISR are shown in Figure 8.

Figure 8. GPIO ISR Location in the *boot.tpl* File



7 Other GPIO Resources and Tips

7.1 GPIO Global Select Register

The PRTxGS register determines 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 the 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, the pin is connected to the Global bus – Global In and Global Out – and can be directly connected to the input or output of a digital block.

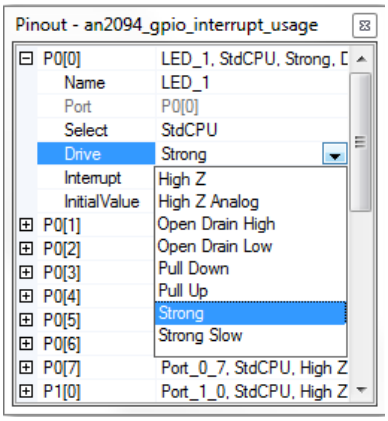
7.2 Analog Mux (AMUX) Bus Control Register

The Analog Mux (AMUX) Bus Control Register (MUX_CRx)[5] enables or disables the 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 the PSoC device. For instance, setting the bit '0' of MUX_CR1 connects P1_0 to the AMuX bus. Refer to the respective device TRM for more information about the AMUX settings and routing.

7.3 Naming a Pin

In the GPIO configuration window, each pin is assigned a unique meaningful name, as shown in Figure 9.

Figure 9. Naming a Pin



When a pin is given a name, PSoC Designer generates macros for all the registers associated with the pin in the *PSoCGPIOINT.h* for C files and *PSoCGPIOINT.asm* for ASM files for easy access. The macro list includes macros for:

- Port data register (PRTxDR)
- Port drive mode registers (PRTxDMx)
- Port interrupt enable register (PRTxIE)
- Port interrupt setup registers (PRTxICx)
- Port global select register (PRTxGS)
- Pin mask
- Shadow register

[5] Available only in CY8C21x34, 21x45, 22x45, 24x94, and 28xxx family of devices.

Table 5 provides an overview of the macros generated in the *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 the 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
Pin Mask	LED_1_MASK
Shadow Register	LED_1_DataShadow

The advantage of using pin names and pin macros is that if a pin is moved to a different port, PSoC Designer will automatically update the registers associated with the pin macros and no change to the application code is required.

The following code snippets show the usage of these macros:

To directly write to the LED_1 pin using the data register:

```
/* Write 1 to LED_1 pin */
LED_1_Data_ADDR |= LED_1_MASK;

/* Write 0 to LED_1 pin */
LED_1_Data_ADDR &= ~LED_1_MASK;
```

To write to the LED_1 pin using the shadow register:

```
/* Write 1 to LED_1 */
LED_1_DataShadow |= LED_1_MASK;
LED_1_Data_ADDR = LED_1_DataShadow;

/* Write 0 to LED_1 */
LED_1_DataShadow &= ~LED_1_MASK;
LED_1_Data_ADDR = LED_1_DataShadow;
```

To change the drive mode of LED_1 to strong mode:

```
/* Set LED_1 drive mode to strong */
LED_1_DriveMode_0_ADDR |= LED_1_MASK;
LED_1_DriveMode_1_ADDR &= ~LED_1_MASK;
LED_1_DriveMode_2_ADDR &= ~LED_1_MASK;
```

To connect or disconnect LED_1 from global bus:

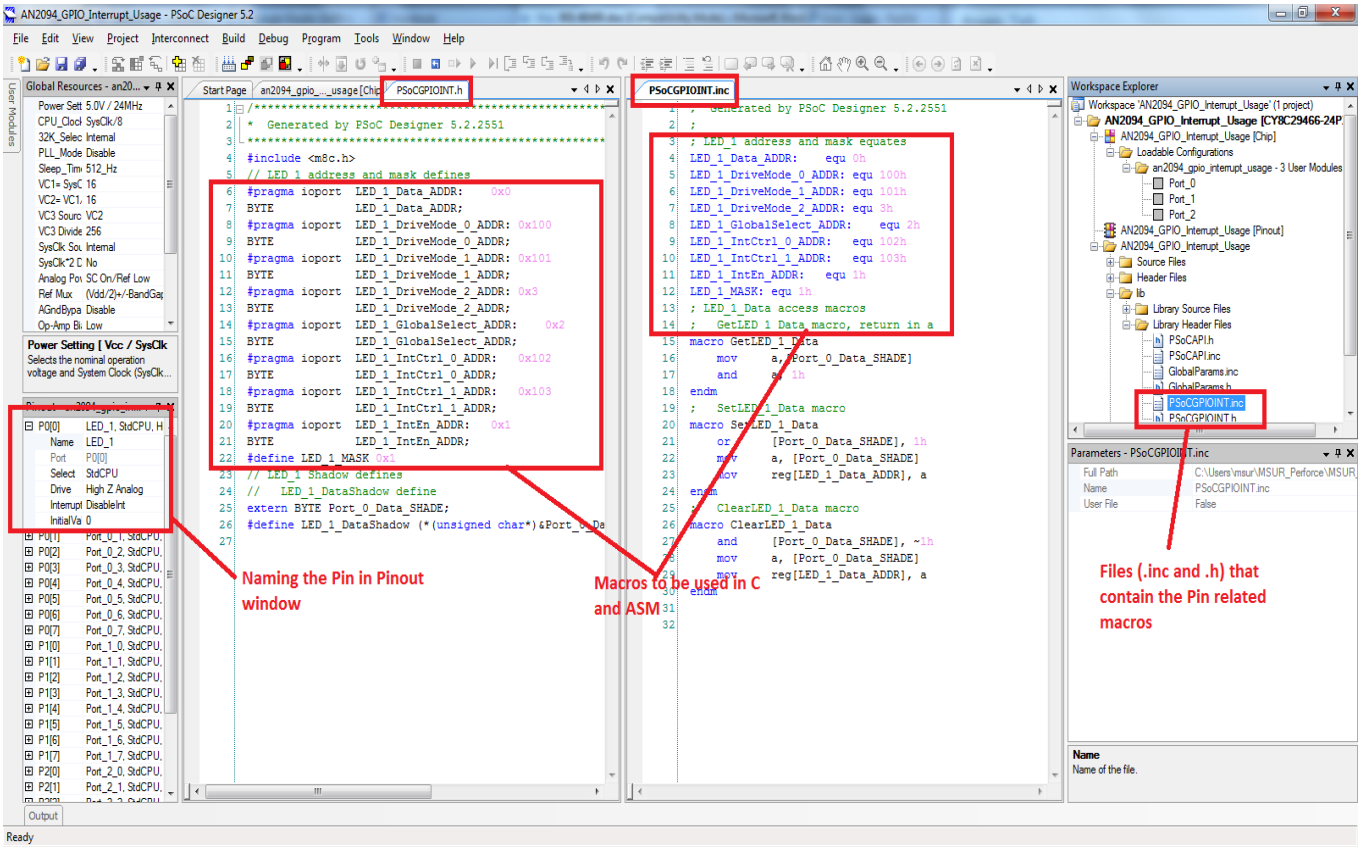
```
/* Connect LED_1 to global bus */
LED_1_GlobalSelect_ADDR |= LED_1_MASK;

/* Disconnect LED_1 from global bus */
LED_1_GlobalSelect_ADDR &= ~LED_1_MASK;
```

To read from a pin named SW and write to the pin named LED_1:

```
if (SW_Data_ADDR & SW_MASK)
{
  /* Write 1 to LED_1 */
  LED_1_DataShadow |= LED_1_MASK;
  LED_1_Data_ADDR = LED_1_DataShadow;
}
else
{
  /* Write 0 to LED_1 */
  LED_1_DataShadow &= ~LED_1_MASK;
  LED_1_Data_ADDR = LED_1_DataShadow;
}
```

Figure 10 shows the pin macros in the *psocgpiont.h* and *psocgpioint.inc* files.

Figure 10. Pin-Related Macros



7.4 Registers and Their Associated Register Banks

Two banks are available in the PSoC 1 register map. Each of the port configuration registers belongs to one of these register banks. It is necessary to know to which bank a particular register belongs, to access the register in assembly. In C code, the compiler automatically takes care of bank switching.

To change the register bank to bank 0 in ASM, use the M8C_SetBank0 macro. Similarly, for bank 1 use the M8C_SetBank1 macro. Table 6 provides the register bank details for each of the GPIO registers 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

8 Example Projects

8.1 Project 1: Detecting LED Drive Mode

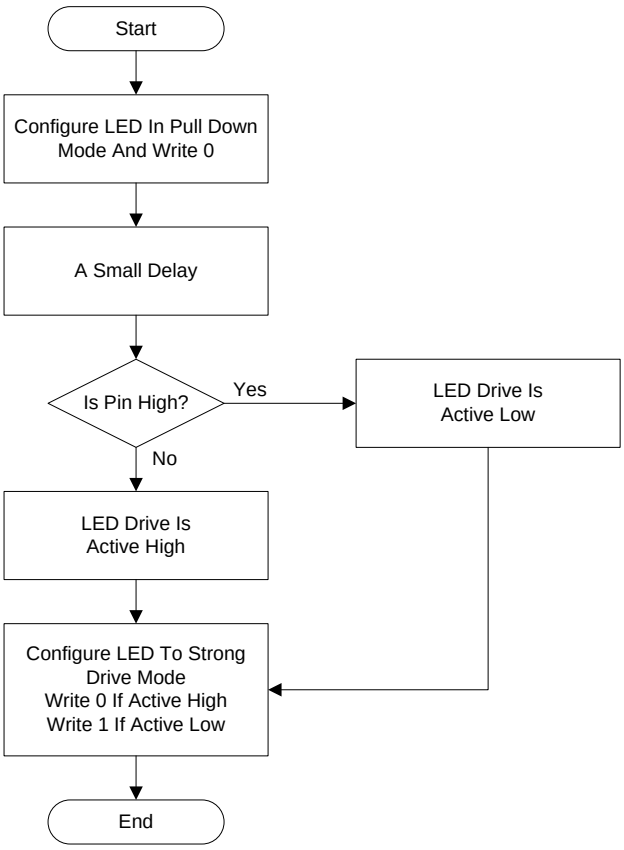
This example project demonstrates how the drive mode of a pin may be reconfigured on the fly using the PRTxDMx registers.

In most systems, LEDs are usually connected to sink current through the GPIOs to turn them on. In certain systems, instead of sinking, the GPIOs source current to the LEDs to turn them on. Though LED implementation is known when the system is designed, there may be cases where you want to upgrade or update the design without changing the firmware.

For example, you have included a sourcing GPIO LED design and find that the GPIO is not capable of sourcing enough current. Or you move to a LED with a 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 or OFF the LED accordingly.

The flow chart in Figure 11 explains how the LED drive mode is detected internally.

Figure 11. Detecting LED Drive Mode Algorithm



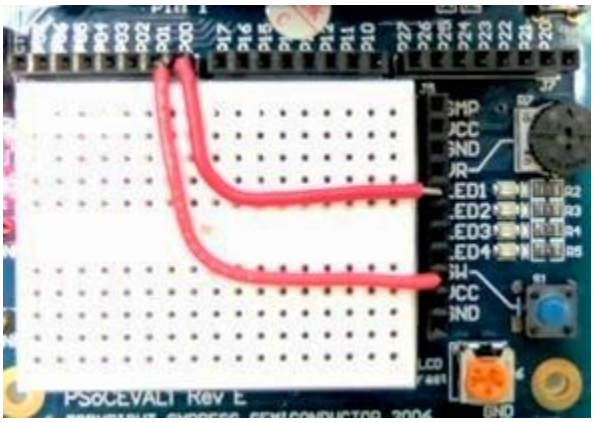
To implement the example project following hardware are required:

- CY3210 PSoCEval1 with 28-pin CY8C29466-24PXI PDIP PSoC 1.
- Spare LED and 1-kΩ resistor to check the LED sink mode
- CY3217 MiniProg1 or CY8CKIT-002 MiniProg3.
- Connecting wires

To test the example **project,** follow the following steps

1. Insert the CY8C29466-24PXI 28-pin device into the 28-pin PDIP socket provided on the CY3210 board.
2. Connect MiniProg1 or MiniProg3 to the programing header (J11) of the CY3210 board.
3. Open PSoC programmer 3.23.1 or later and connect MiniProg1 or MiniProg3.
4. Browse the *AN2094_GPIO_DM_Reconfig.hex* file available in the root directory of the AN2094_GPIO_DM_Reconfig project attached with this application note i.e. open folder *AN2094* > *AN2094_GPIO_DM_Reconfig* > *AN2094_GPIO_DM_Reconfig.hex*
5. Program the device using selected file by using program button of PSoC programmer. To know more about programming options please see AN2015 - PSoC 1 Reading and Writing Flash & E2PROM.
6. Connect P0_1 to SW and P0_0 to LED1, as shown in Figure 12.

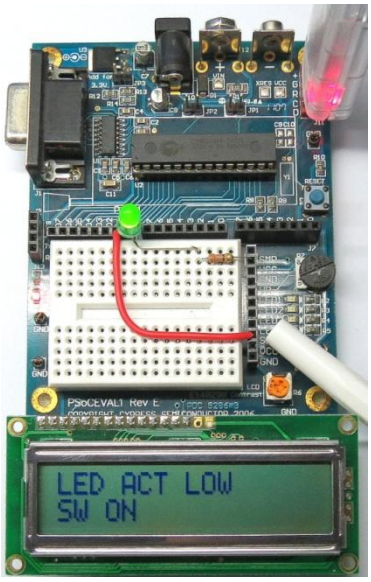
Figure 12. Wire Connections for project 1



7. Connect the character LCD (provided with CY3210 PSoCEval1 kit) to the J9 header on the CY3210 board.
8. Remove JP3 on the CY3210 kit for 5-V operation.
9. Power the board using a MiniProg1 or MiniProg3 or a 5-V DC adapter.
10. LCD row 0 should display “LED ACT HI” and when SW is pressed, LED1 glows and LCD row 1 displays “SW ON”, as shown in Figure 13.
11. Similarly, if you wire up a spare LED in sink mode via a 1-kΩ resistor (one end connected to LED and other end connected to Vcc) and connect it to P0_0, the LCD will display “LED ACT LOW” and LED ON/OFF follows SW ON/OFF, as shown in Figure 14.

Figure 13. Example 1 Output - LED Active HIGH

Figure 14. Example 1 Output - LED Active LOW



8.2 Project 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 shown in Figure 15.

In this example, the hardware has a provision to enable or disable the 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 demonstrates the scenario explained in the section Use of Shadow Registers, where the same port has an input switch and an output LED.

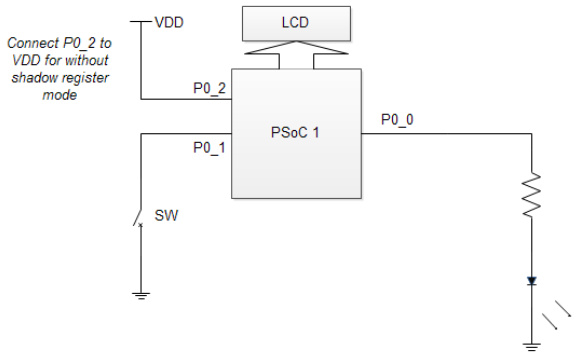
8.2.1 Hardware Required

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

8.2.2 Test Procedure

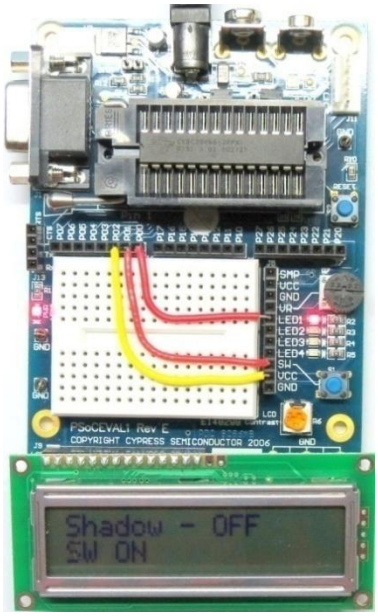
1. Insert the CY8C29466-24PXI 28-pin device into the 28-pin PDIP socket provided on the CY3210 board.
2. Connect MiniProg1 or MiniProg3 to the programing header (J11) of the CY3210 board.
3. Open PSoC programmer 3.23.1 or later and connect MiniProg1 or MiniProg3.
4. Browse the *AN2094_GPIO_with_ShadowRegs.hex* file available in the root directory of the AN2094_GPIO_with_ShadowRegs project attached with this application note i.e. *AN2094>AN2094_GPIO_with_ShadowRegs> AN2094_GPIO_with_ShadowRegs.hex*
5. Program the device using selected file by using program button of PSoC programmer. To know more about programming options please see AN2015 - PSoC 1 Reading and Writing Flash & E2PROM
6. Connect the character LCD (provided with CY3210 PSoCEval1 kit) to the J9 header on the CY3210 board.
7. To test the project without shadow variables, connect P0_0 to LED1, P0_1 to SW, and P0_2 to V_{DD}, as shown in Figure 15.

Figure 15. Pin Connections for Example 2 without Shadow Registers



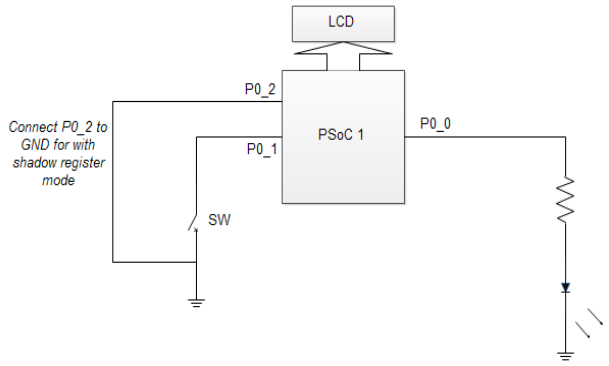
8. Remove JP3 in the CY3210 kit for 5-V operation.
9. Power the board using a MiniProg1 or MiniProg3 or a 5-V DC adapter.
10. Now, press SW once and release; observe that LCD row 1 displays “SW ON” and LED1 will be ON. This is because there is no shadow variable used.

Figure 16. Project 2 Output without Shadow Registers



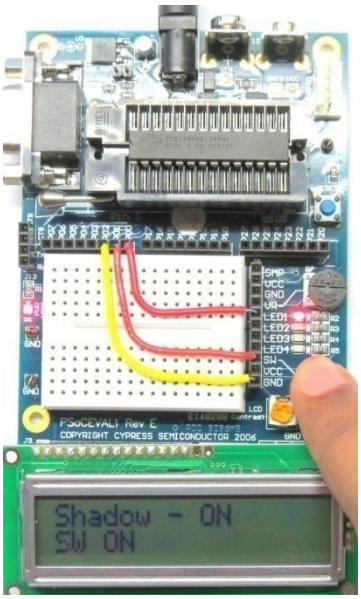
11. Power off the board and connect P0_2 to GND, as shown in Figure 17.

Figure 17. Pin Connections for Example 2 With Shadow Registers



6. Power up the board.
7. LCD row 0 will display “Shadow - ON”. Pressing the switch will display SW ON and the LED will glow as shown in Figure 18. Releasing the switch will display SW OFF and the LED will be turned off.

Figure 18. Example 2 Output With Shadow Registers



8.3 Project 3: LED Toggling Using Interrupts

To demonstrate the use of GPIO interrupts, a simple LED toggle algorithm is implemented in this example. The rising edge interrupt is enabled on the pin, which is connected to a switch. In the ISR, a flag is set to indicate the rising edge on the switch. The LED is toggled in the while loop after a simple 2-ms debounce. The LED ON/OFF status is displayed on the LCD.

8.3.1 Hardware Required

- CY3210 PSoCEval1 with 28-pin CY8C29466-24PXI PDIP PSoC 1.
- CY3217 MiniProg1
- Connecting wires

8.3.2 Test Procedure

1. Insert the CY8C29466-24PXI 28-pin device into the 28-pin PDIP socket provided on the CY3210 board.
2. Connect MiniProg1 or MiniProg3 to the programing header (J11) of the CY3210 board.
3. Open PSoC programmer 3.23.1 or later and connect MiniProg1 or MiniProg3.
4. Browse the *AN2094_GPIO_Interrupt_Usage.hex* file available in the root directory of the AN2094_GPIO_with_ShadowRegs project attached with this application note i.e. *AN2094> AN2094_GPIO_Interrupt_Usage> AN2094_GPIO_Interrupt_Usage.hex*
5. Program the device using selected file by using program button of PSoC programmer. To know more about programming options please see AN2015 - PSoC 1 Reading and Writing Flash & E2PROM
6. Connect the character LCD (provided with CY3210 PSoCEval1 kit) to the J9 header on the CY3210 board.
7. To test the project, connect P0_0 to LED1 and P0_1 to SW as shown in Figure 19.
8. Remove JP3 in the CY3210 kit for 5-V operation.
9. Power the board using MiniProg1 or a 5-V DC adapter.
10. Press SW and see LED1 toggling on each press.

11. Row 0 on the LCD displays “GPIO ISR Proj”; row 1 displays the LED ON/OFF status, as shown in Figure 19 and Figure 20.

Figure 19. Example 3 Output With LED ON

Figure 20. Example 3 Output With LED OFF

8.4 Additional Code Examples

More code examples using PSoC 1 devices are available in PSoC Designer 5.4. SP1 or later. To access them, go to: **Start Page > Design Catalog > Launch Example Browser**.

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	Re-catalogued 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.
*F	4053232	MSUR	07/08/2013	Updated images showing the example outputs (Figure 6, 7, 10, 12, 13, and 14). Added Code 10 and 11 to demonstrate assembly implementation of GPIO ISR. Updated references related to the above modifications. Updated projects to include drive mode setting through assembly. Updated projects to PSoC designer 5.3. Updated GPIO ISR project to have a better switch debounce.
*G	4330651	GRAA	04/02/2014	Added a section GPIO Architecture and explained the architecture of the GPIO cell Added a section Code Level Configuration and explained all the registers associated with GPIO. Changed the order of sections to improve the flow of the AN In the GPIO Interrupts section, added code snippets to show how the redirect to a GPIO ISR is achieved through psocgpioint.asm file or boot.tpl file Added more details to the “Naming a Pin” section. Provided code snippets on the usage of all the GPIO macros. Moved all the example projects to appendix to improve the flow of AN.
*H	4371610	MSUR	06/05/2014	Sunset Review.
*I	4494118	DIMA	09/05/2014	Added the Additional Code Examples section to provide a reference to code examples integrated with PSoC Designer 5.4.
*J	4952015	ASRI	10/15/2015	Added instructions in Project 1: Detecting LED Drive Mode, Project 2: Use of Shadow Registers and Project 3: LED Toggling Using Interrupts, to test the example projects associated with this application note. Updated the projects to PSoC Designer 5.4 SP1. Added Getting Started.
*K	5732618	AESATP12	05/16/2017	Updated logo and copyright.

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

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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