

## PSoC<sup>®</sup> 4 – Using GPIO Pins

**Author: Rajiv Badiger**

**Associated Project: Yes**

**Associated Part Family: PSoC 4**

**Software Version: PSoC Creator™ 3.2 or higher**

**Related Application Notes: For a complete list of the application notes, [click here](#).**

**If you have a question, or need help with this application note, contact the author at [rjvb@cypress.com](mailto:rjvb@cypress.com)**

AN86439 explains how to use PSoC<sup>®</sup> 4 GPIO pins effectively with various use case examples to take full advantage of their features. Major topics include GPIO basics, configuration options, mixed-signal use, interrupts, and low-power behavior.

### Contents

1	Introduction.....	2	10.9	Using Both Analog and Digital on a GPIO .....	44
2	PSoC Resources .....	2	10.10	Gang Pins for More Drive/Sink Current .....	46
3	PSoC Creator .....	3	10.11	Control Register Handling in Deep-Sleep..	49
4	Example Projects.....	4	11	Related Application Notes .....	51
5	PSoC Creator Help.....	5	12	Summary .....	52
6	Technical Support.....	5	13	About the Authors.....	52
7	GPIO Pin Basics.....	6	A	Appendix A: PSoC 4 GPIO Compared to PSoC 1, PSoC 3, and PSoC 5LP GPIO .....	53
7.1	Physical Structure of GPIO Pins .....	6	B	Appendix B: PSoC 4 Development Boards .....	53
7.2	Pin Routing .....	8		Document History.....	54
7.3	Startup and Low-Power Behavior .....	15		Worldwide Sales and Design Support.....	55
7.4	GPIO Interrupt .....	16		Products.....	55
8	Overvoltage-Tolerant (OVT) Pins .....	17		PSoC <sup>®</sup> Solutions .....	55
9	GPIO Pins in PSoC Creator .....	18		Cypress Developer Community.....	55
9.1	Pins Component Symbols.....	18		Technical Support .....	55
9.2	Pins Component Customizer .....	18			
9.3	Pins Component Interrupts .....	20			
9.4	Manual Pin Assignments .....	23			
9.5	PSoC Creator APIs .....	23			
9.6	Debug Logic on GPIO Pins.....	24			
9.7	Add Multiple GPIO Pins as a Logical Port.....	24			
9.8	Represent Off-Chip Components.....	26			
10	GPIO Tips and Tricks .....	28			
10.1	Toggle an LED.....	29			
10.2	Read an Input and Write to an Output .....	30			
10.3	Drive an Output from a Digital Logic Gate .....	30			
10.4	Set the GPIO Input/Output Synchronization ..	31			
10.5	Toggle GPIOs Faster with Data Registers.....	36			
10.6	Configure GPIO Output Enable Logic.....	39			
10.7	Pin Interrupt .....	41			
10.8	Configure GPIO Interrupt Settings with Firmware .....	42			

## 1 Introduction

PSoC has powerful and flexible general-purpose I/O (GPIO) pins that provide more features than traditional MCUs. In PSoC, the GPIOs are controlled not only by configuring the registers in the firmware, similar to traditional MCUs, but are also driven by custom digital logic and analog block signals. This application note explains the basics of PSoC 4 GPIO pins and demonstrates techniques for using them effectively for different functions.

This application note assumes that you are familiar with PSoC Creator™ and the PSoC 4 architecture. If you are new to PSoC 4, read [AN79953 – Getting Started with PSoC 4](#). If you are new to PSoC Creator, visit the [PSoC Creator home page](#). For information on device packages or GPIO specifications, see the [PSoC 4 datasheet](#). If you already know the PSoC 4 device and the PSoC Creator, you can jump to the section [GPIO Tips and Tricks](#).

## 2 PSoC Resources

Cypress provides a wealth of data at [www.cypress.com](http://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 [KBA86521, How to Design with PSoC 3, PSoC 4, and PSoC 5LP](#). The following is an abbreviated list for PSoC 4:

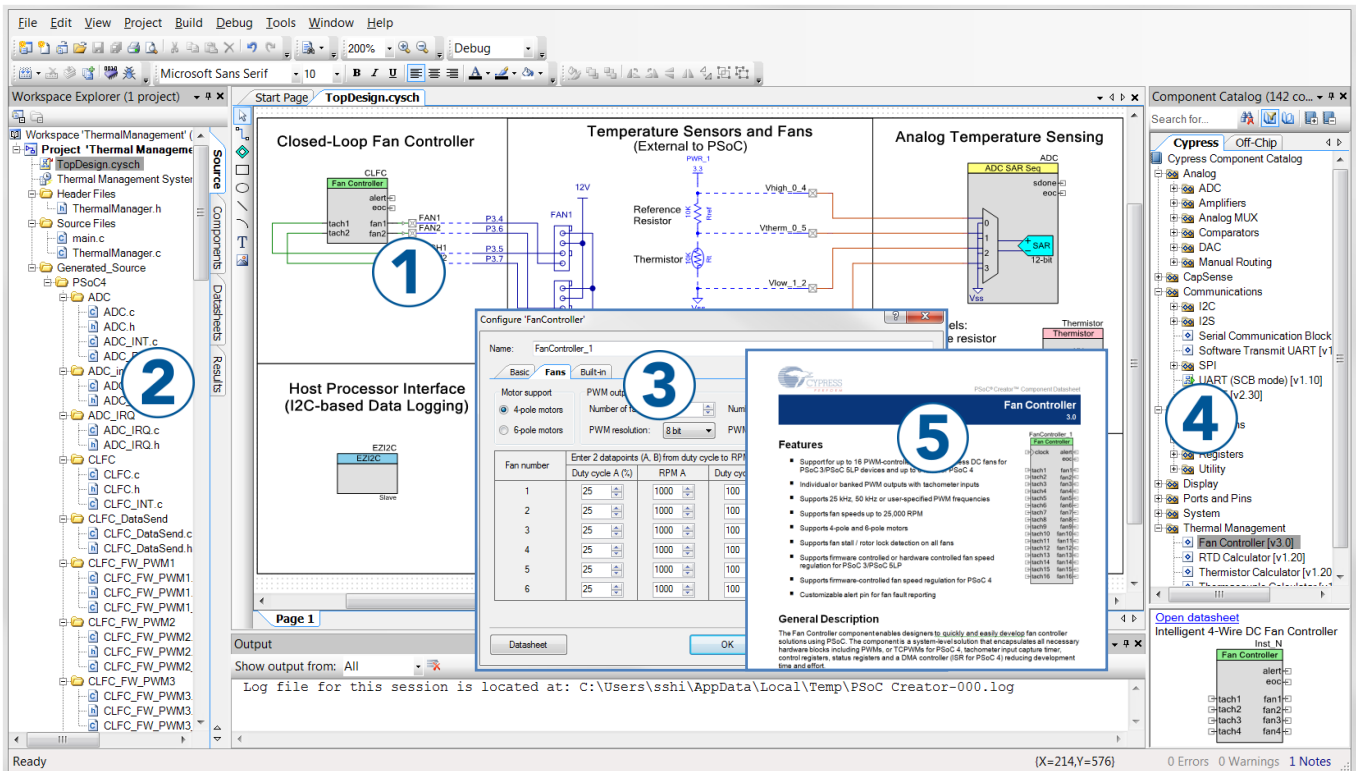
- **Overview:** [PSoC Portfolio](#), [PSoC Roadmap](#)
- **Product Selectors:** [PSoC 1](#), [PSoC 3](#), [PSoC 4](#), or [PSoC 5LP](#). In addition, [PSoC Creator](#) includes a device selection tool.
- **Datasheets:** Describe and provide electrical specifications for the [PSoC 4000](#), [PSoC 4100](#), and [PSoC 4200](#), [PSoC 4xx7 BLE](#), [PSoC 4200-M](#) device families
- **CapSense Design Guide:** Learn how to design capacitive touch-sensing applications with the PSoC 4 family of devices.
- **Application Notes and Code Examples:** Cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples. PSoC Creator provides additional code examples – see [Code Examples](#).
- **Technical Reference Manuals (TRM):** Provide detailed descriptions of the architecture and registers in each PSoC 4 device family.
- **Development Kits:**
  - [CY8CKIT-040](#), [CY8CKIT-042](#), [CY8CKIT-042-BLE](#), and [CY8CKIT-044](#) PSoC 4 Pioneer Kits are easy-to-use and inexpensive development platforms. These kits include connectors for Arduino™ compatible shields and Digilent® Pmod™ daughter cards.
  - [CY8CKIT-049](#) is a very low-cost prototyping platform for sampling PSoC 4 devices.
  - [CY8CKIT-001](#) is a common development platform for all PSoC family devices.
- The [MiniProg3](#) device provides an interface for flash programming and debug.

### 3 PSoc Creator

PSoc Creator is a free Windows-based Integrated Design Environment (IDE). It enables concurrent hardware and firmware design of systems based on PSoc 3, PSoc 4, and PSoc 5LP. See Figure 1 – with PSoc Creator, you can:

1. Drag and drop Components to build your hardware system design in the main design workspace
2. Codesign your application firmware with the PSoc hardware
3. Configure Components using configuration tools
4. Explore the library of 100+ Components
5. Review Component datasheets

Figure 1. PSoc Creator Features



## 4 Example Projects

PSoC Creator includes a large number of example projects. These projects are available from the PSoC Creator Start Page, as Figure 2 shows.

Example projects can speed up your design process by starting you off with a complete design, instead of a blank page. The example projects also show how you can use PSoC Creator Components for various applications. Example projects and datasheets are included, as Figure 3 shows.

In the **Find Example Project** dialog shown in Figure 3, you have several options:

- Filter for examples based on architecture or device family (such as PSoC 3, PSoC 4, or PSoC 5LP); category; or keyword
- Select from the menu of examples offered based on the **Filter Options**
- Review the datasheet for the selection (on the **Documentation** 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 2. Code Examples in PSoC Creator

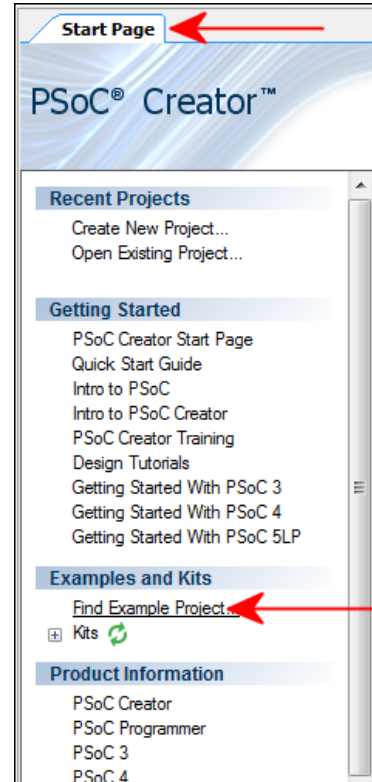
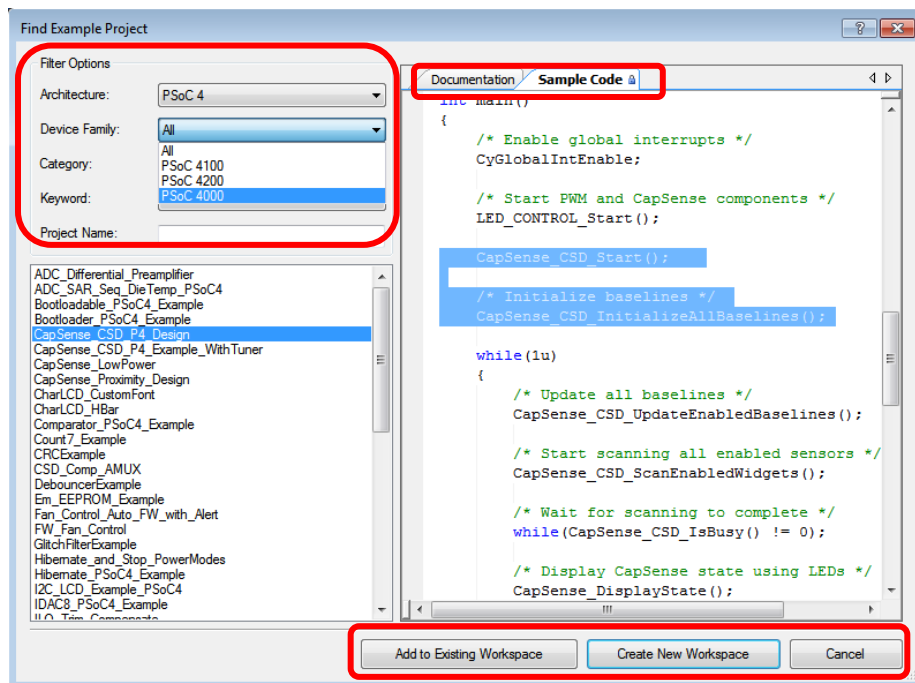


Figure 3. Example Projects with Sample Code



## 5 PSoC Creator Help

Visit the [PSoC Creator home page](#) to download the latest version of PSoC Creator. Then, launch PSoC Creator and navigate to the following items:

- **Quick Start Guide:** Choose **Help > Documentation > Quick Start Guide**. This guide gives you the basics for developing PSoC Creator projects.
- **Simple Component example projects:** Choose **File > Open > Example projects**. These example projects demonstrate how to configure and use PSoC Creator Components.
- **Starter designs:** Choose **File > New > Project > PSoC 4 Starter Designs**. These starter designs demonstrate the unique features of PSoC 4.
- **System Reference Guide:** Choose **Help > System Reference > System Reference Guide**. This guide lists and describes the system functions provided by PSoC Creator.
- **Component datasheets:** Right-click a component and select “Open Datasheet.” Visit the [PSoC 4 Component Datasheets](#) page for a list of all PSoC 4 Component datasheets.
- **Document Manager:** PSoC Creator provides a document manager to help you to easily find and review document resources. To open the document manager, choose the menu item **Help > Document Manager**.

## 6 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](#)

## 7 GPIO Pin Basics

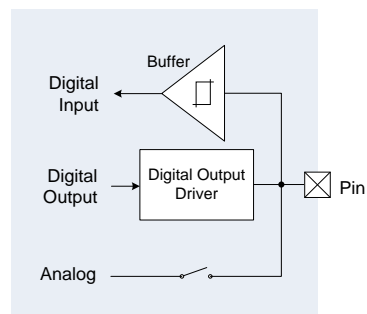
The PSoC 4 GPIO pins offer the following features:

- Analog and digital input and output capability
- LCD segment drive support (not available in PSoC 4000)
- CapSense® support
- Interrupt on level, rising-edge, falling-edge, or both edges
- Slew-rate control
- Input threshold select (CMOS / LVTTTL / 1.8-V CMOS)
- Overvoltage-tolerant pins (available only in PSoC 4 BLE and PSoC 4 M-Series) with hot swap capability

### 7.1 Physical Structure of GPIO Pins

The pin connections with the resources in the PSoC 4 device can be represented as shown in [Figure 4](#).

Figure 4. Simplified GPIO Block Diagram



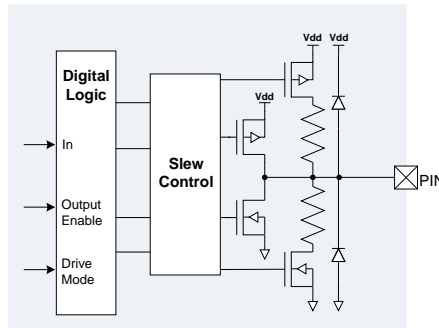
A detailed block diagram of the GPIO structure is available in the “I/O System” chapter of the [PSoC 4 Architecture TRM](#). Each pin can act as an input or an output to the CPU and the digital peripheral such as the Timer, PWM, or I<sup>2</sup>C. It can also act as an analog pin for use with opamps and ADC. At any given time, you can use a pin for only digital input, or only digital output, or only analog pin, or even combinations of these three. For example, if you enable both the digital output and the input, it provides a digital bidirectional pin. The input buffer provides high-impedance to the external input. It is configurable to the following signaling standards:

- CMOS
- LVTTTL
- 1.8-V CMOS (available only in PSoC 4 BLE and PSoC 4 M-Series)

For the input threshold values, see the [device datasheet](#).

The digital output driver supports different drive modes and slew-rate control (see [Figure 5](#)).

Figure 5. Digital Output Driver



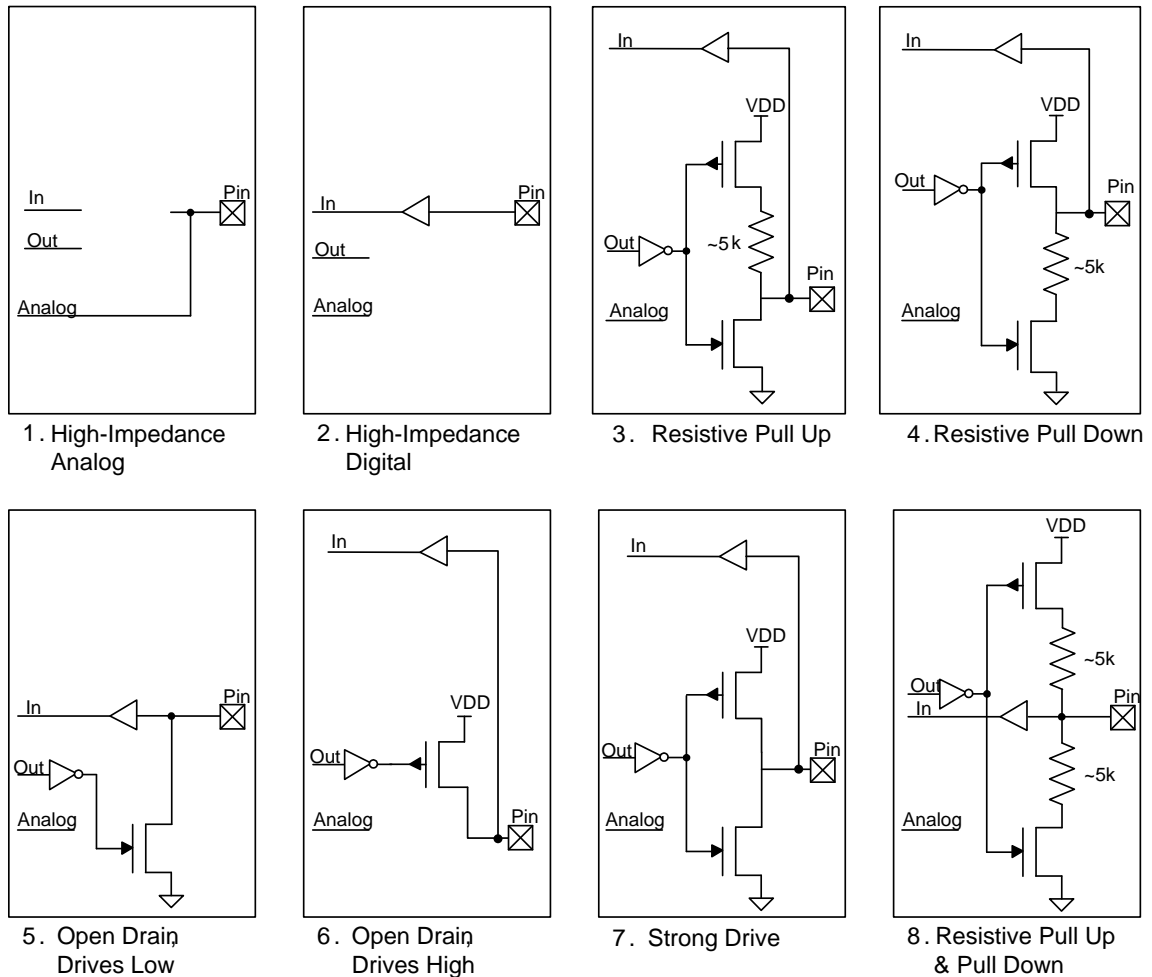
Slew-rate control is provided to reduce EMI and cross-talk. There are two options – Fast and Slow. Slew rate is set to Fast by default. Use the Slow option when the signals are not speed-critical.

The circuit shown in [Figure 5](#) supports eight drive modes in PSoC 4, as listed in [Table 1](#).

Table 1. Drive Modes and Applications

#	Drive Mode	Application Examples
1	High-impedance Analog	Analog input/output
2	High-impedance Digital	Digital input
3	Resistive Pull-Up (~5 KΩ)	Interface to open drain-low input, such as tachometer output from motors or a switch connected to ground. It can also be used to drive LEDs.
4	Resistive Pull-Down (~5 KΩ)	Interface to an open drain-high input or a switch connected to VDD. It can be used as an output to interface LEDs in current sink mode.
5	Open Drain, Drives Low	Provides high-impedance in the high state and a strong drive in the low state; This configuration is used for I <sup>2</sup> C pins. This mode works in conjunction with an external pull-up resistor.
6	Open Drain, Drives High	Provides strong drive in the high state and high-impedance in the low state. This mode works in conjunction with an external pull-down resistor.
7	Strong Drive	CMOS output drive in both low and high state
8	Resistive Pull-Up and Resistive Pull-Down (~5 KΩ)	Adds series resistor in both high and low states

Figure 6. Drive Modes



**Note 1:** The resistor values for the pull-up and pull-down drive modes, shown in Figure 6, are approximate values; see the [device datasheet](#) for the resistor value specifications. Use an external resistor if higher accuracy is required. In this case, the pin must be configured as open drain drive high or open drain drive low.

**Note 2:** At all times, avoid the device VDD getting powered from an external voltage at the pin through the ESD clamp diodes. This can happen if the PSoC 4 device is not powered and an external voltage is applied at the GPIO or when an external voltage at the GPIO is greater than the device VDD. This is, however, not applicable to the [Overvoltage-Tolerant \(OVT\) Pins](#) as there are no clamp diodes.

## 7.2 Pin Routing

### 7.2.1 Digital Routing

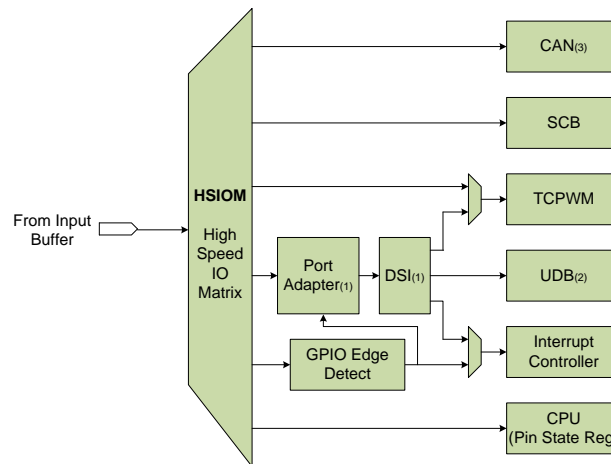
A pin can be routed to different digital peripherals, such as the universal digital block (UDB), serial communication block (SCB), timer/counter/pulse-width modulator (TCPWM) block, LCD driver, CAN block, interrupt controller, and the data register which is read/written by the CPU. Figure 7 shows the routing for an input pin and Figure 8 shows the routing for an output pin.



As shown in [Figure 7](#) and [Figure 8](#), peripherals are connected to the pins using the high-speed I/O matrix (HSIOM). It multiplexes the signals from different peripherals to connect to a particular pin.

In PSoC 4, there are two routing possibilities: dedicated I/O routed through the HSIOM and flexible routing using digital system interconnect (DSI). DSI usage is not limited to routing the peripheral inputs and outputs to pins; it is also used to route signals between digital resources. The Port Adapter connects the HSIOM and the DSI. It also provides hardware to synchronize pin input and output signals.

Figure 7. Digital Pin Input Path

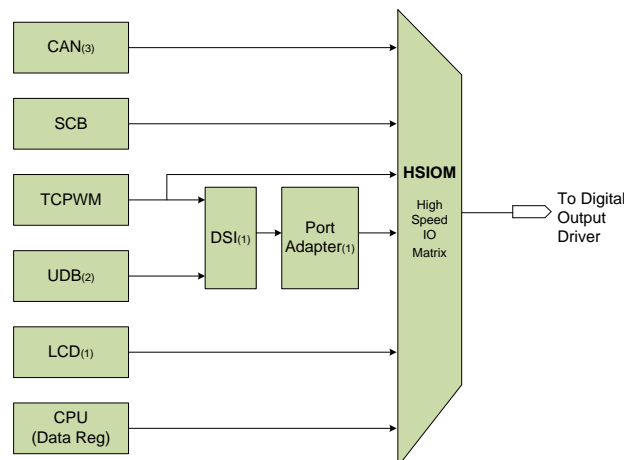


1. not applicable to PSoC 4000
2. not applicable to PSoC 4000, PSoC 4100, PSoC 41xx-BL and PSoC 4100M
3. only available in PSoC 4200M

SCB (I<sup>2</sup>C, UART, and SPI) and TCPWM have dedicated routes to some I/Os. The flexible routing option is available for UDB inputs and outputs, generating interrupts from the pins, and even for TCPWM. The LCD driver is present in all I/Os of the PSoC 4 parts (except PSoC 4000), with any I/O acting as a segment or a common driver for the LCD glass.

The GPIO Edge Detect block enables pin interrupts on rising-edge, falling-edge, and both edges. See the [GPIO Interrupt](#) section for details.

Figure 8. Digital Pin Output Path



1. not applicable to PSoC 4000
2. not applicable to PSoC 4000, PSoC 4100, PSoC 41xx-BL and PSoC 4100M
3. available only in PSoC 4200M

**Note:** PSoC 4 has multiple ports with a maximum of 8 pins per port. Port 4 and higher ports do not have the port adapter. These ports have the following restrictions:

- Cannot be routed through the DSI; thus UDB-based digital signals cannot be routed to the pins of these ports
- Cannot be used for analog blocks, such as SAR ADC, Opamp - Continuous Time Block mini (CTBm), and Low-Power Comparator (applicable only to PSoC 4100 and PSoC 4200)

- No input/output synchronization

However, these ports are useful in the following ways:

- As a GPIO controlled in firmware
- Direct connection to TCPWM, SCB, or CAN
- LCD and CapSense pins
- Interrupts generation

**Note:** Pins of the PSoC 4 device are shared for dedicated connections to different peripherals. To know the functions possible at each pin, see the “Pinouts” section in the respective [device datasheets](#).

### 7.2.2 Analog Routing

GPIO pins configured in the High-Impedance Analog (HI-Z) mode are connected to the analog resources by direct connections or through the analog switches and the analog mux (AMUX) bus, as [Figure 9](#) to [Figure 12](#) show.

Following are the key highlights of the analog routing in the PSoC 4000 parts shown in [Figure 9](#):

- All pins (except port 3) can connect to the AMUX buses, controlled by firmware. There are two buses: AMUXBUS\_A and AMUXBUS\_B.
- CapSense IDAC0 is connected to AMUXBUS\_A, and IDAC1 is connected to AMUXBUS\_B.
- CapSense CMOD is connected to P0[4], and the shield tank capacitor is connected to P0[2].
- Any pin can be used for the capacitive touch sensors (except Port 3) as the CapSense block connects to the sensors using the AMUX bus.

**Note:** Place the CMOD capacitor close to the pin. See [AN85951 PSoC 4 CapSense Design Guide](#) for layout guidelines.

Following are the key highlights of the analog routing in other PSoC 4 parts, as shown in [Figure 10](#), [Figure 11](#), and [Figure 12](#):

- There are two AMUX buses. All pins have the capability to connect to AMUXBUS\_A and AMUXBUS\_B. AMUX bus connection can be controlled by firmware or using the DSI signal. Note that in case of port 4 and higher port pins, where the DSI connection is not available, AMUX can be connected only in firmware.
- Direct connections are available for opamp inputs and outputs, which provide better performance due to lower trace resistance and parasitic capacitance.
- Direct connections are also available for low-power comparator (LPCOMP) inputs without switches.
- There are dedicated pins for CapSense CMOD and the shield tank capacitor. See [Figure 10](#), [Figure 11](#), and [Figure 12](#) to know the pins.
- CapSense IDAC0 is connected to AMUXBUS\_A, and IDAC1 is connected to AMUXBUS\_B.

- Any pin can be used for the capacitive touch sensors as the CapSense block connects to the sensors using the AMUX bus.
- AMUXBUS\_A and AMUXBUS\_B can be split using switches (marked in blue) as shown in Figure 11 and Figure 12. This is useful in case the AMUX buses are required for non-CapSense applications, such as opamp/comparator input and output routing, along with the CapSense in the system.
- The SAR sequencer connects the SAR ADC input to:
  - Port 2 in PSoC 4100 / PSoC 4200 and PSoC 4100M / PSoC 4200M and Port 3 in PSoC 41xx-BL / PSoC 42xx-BL
  - CTBm outputs
  - Temperature sensor output.

Multiplexing is done by controlling the switches shown in red in Figure 10, Figure 11, and Figure 12. Note that the SAR ADC can also take the input from any pin using AMUXBUS without the sequencer.

**Note:** The opamp output is connected to a dedicated pin without any switches. If the connection to AMUX bus is required, the AMUX switch associated with the dedicated pin is activated. This also allows other pins to act as opamp output pins if the corresponding AMUX switches are activated.

**Note:** When the SAR ADC is operated with differential inputs in the sequencer mode, the positive input can only be an even pin with negative input as the adjacent odd pin. For example, in PSoC 4200, P2[0] and P2[1] are pair pins with P2[0] as positive input and P2[1] as negative input.

Figure 9. PSoC 4000 Analog Routing Diagram

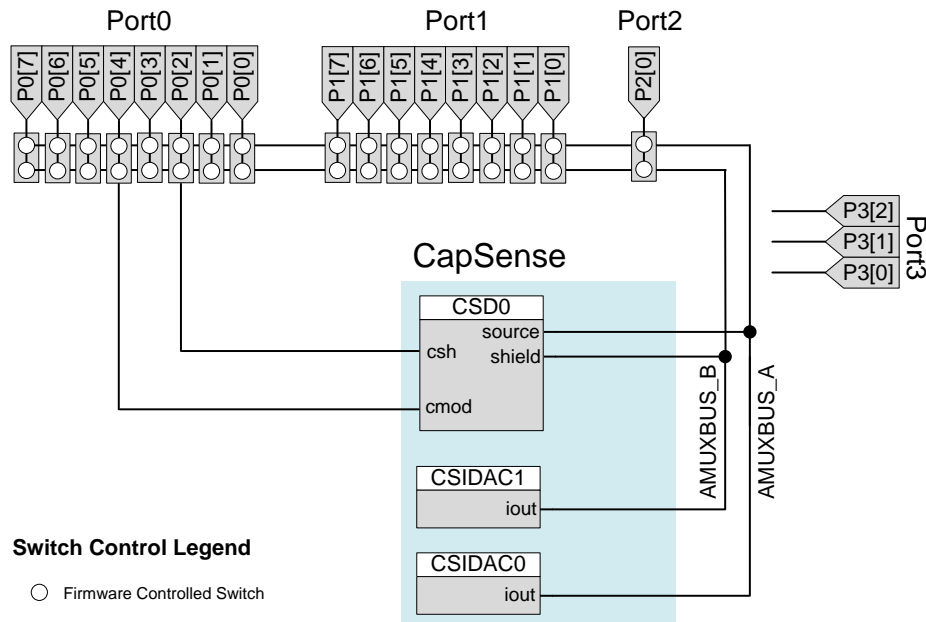


Figure 10. PSoC 4200/PSoC 4100 Analog Routing Diagram

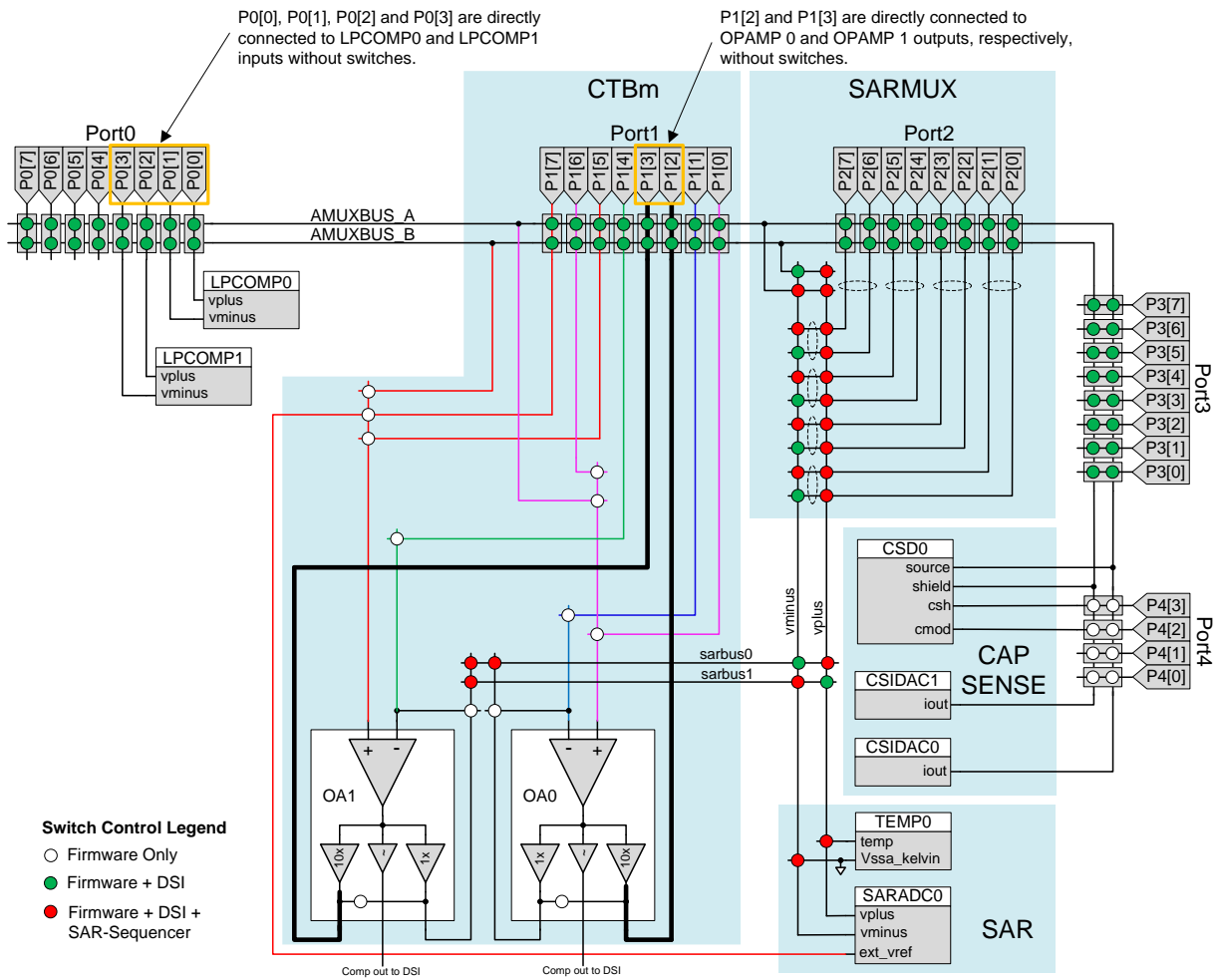


Figure 11. PSoC 41xx-BL/PSoC 42xx-BL Analog Routing Diagram

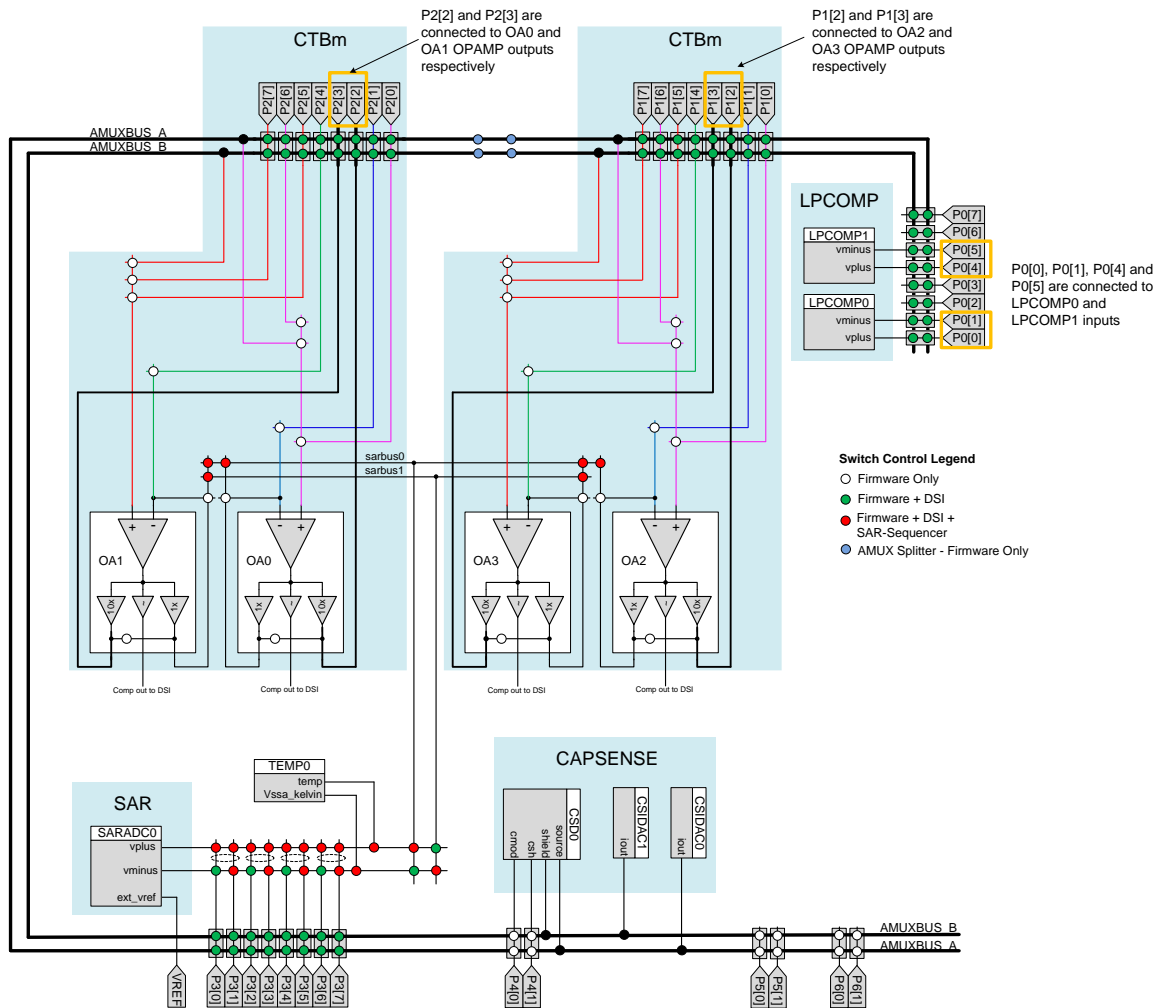
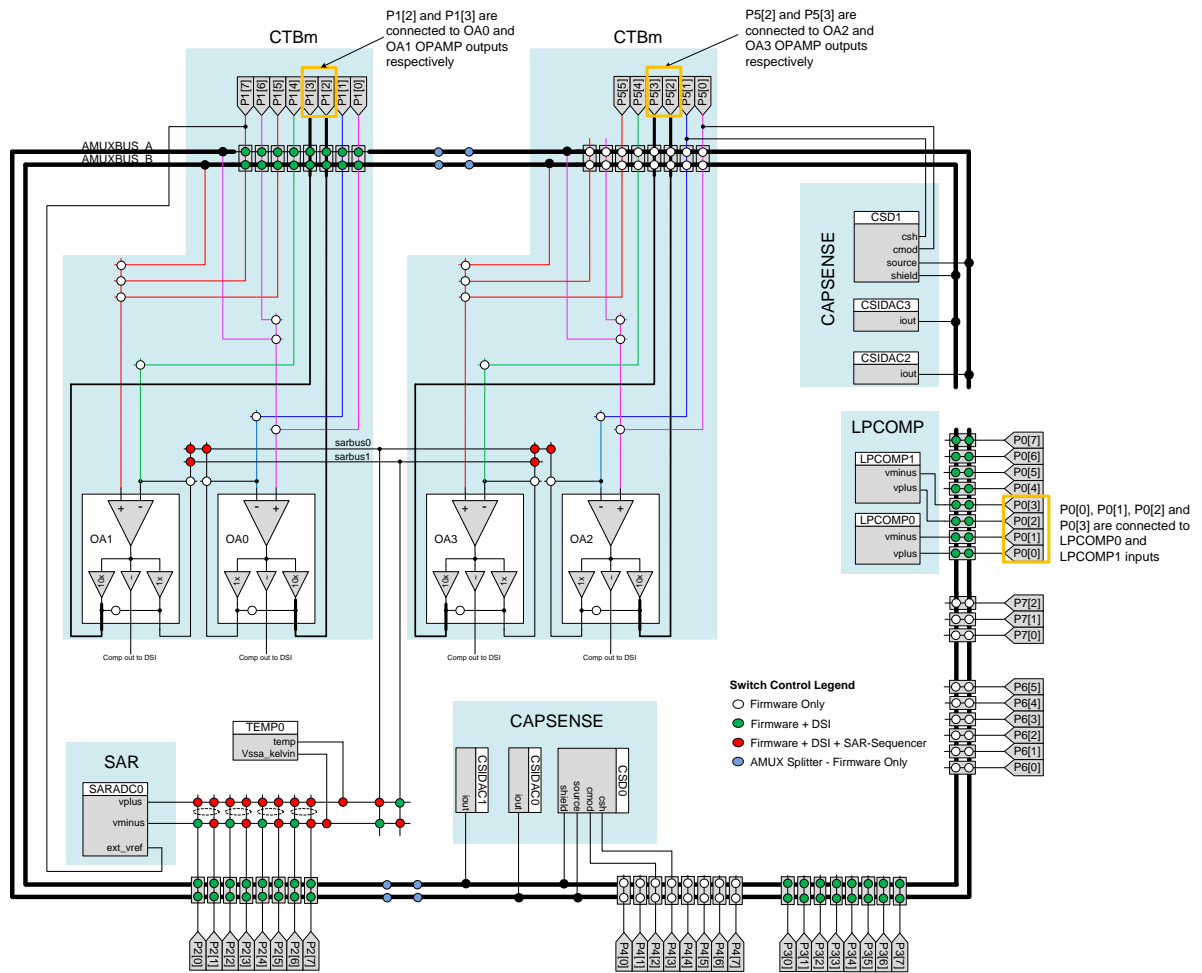


Figure 12. PSoC 4100M/PSoC 4200M Analog Routing Diagram



**Note:** The PSoC Creator IDE tool provides an analog routing diagram for a design similar to those illustrated in Figure 9 to Figure 12. See the Analog tab in the .cydwr file of the project in PSoC Creator.

### 7.3 Startup and Low-Power Behavior

On reset/power-up, all GPIO pins start up in the high-impedance analog mode, that is, with input buffer and output driver disabled. It remains in this mode until the reset is released. Then, each pin is set to its initial configuration before invoking the main () function.

**Note** In the PSoC 4000 parts (all parts except the 24-pin QFN), pin P1[6] is temporarily configured as XRES during power-up until the device executes the start-up code. Do not pull this pin down during power-up as this keeps the device in reset.

See [KBA91258 – I/O System Restrictions in the PSoC 4000 Family](#) for more information.

PSoC 4 has four power modes. [Table 2](#) shows the power mode availability in the PSoC 4 families.

Table 2. Low Power Modes

Device	Sleep	Deep Sleep	Hibernate	Stop
PSoC 4000	✓	✓	✗	✗
PSoC 4100/4200	✓	✓	✓	✓
PSoC 4 BLE	✓	✓	✓	✓
PSoC 4 M	✓	✓	✓	✓

In the Sleep mode, the GPIOs are active and can be actively driven by the peripherals. Only the CPU is inactive in this mode. In the Deep-Sleep mode, the pins driven by the deep-sleep sources such as I<sup>2</sup>C, LCD driver, opamp, and comparator are functional. The I<sup>2</sup>C pins can wake the device up on an address match event. The segment LCD, connected to the device pins, is periodically refreshed even in the Deep Sleep mode.

The PSoC 4 parts (except PSoC 4000) have an additional feature that freezes the GPIOs in Deep-Sleep, Hibernate, and Stop mode. Unfreezing of GPIOs also happens automatically when the low power mode is exited. However, note that the GPIOs driven by Deep-Sleep peripherals are active in Deep-Sleep mode and are not frozen.

In the case of Hibernate and Stop mode, wake up happens with a device reset. This clears the GPIO configuration and the state. To retain the pin state, use the `CySysPmFreezeIo()` and `CySysPmUnfreezeIo()` APIs. Note that you do not need to call the `CySysPmFreezeIo()` function for Stop mode since it is automatically called when the user invokes Stop mode using the API `CySysPmStop()`. However, you should call the `CySysPmFreezeIo()` API just before the function call to enter the Hibernate mode. The GPIOs are unlocked with the call to API `CySysPmUnfreezeIo()`. Call to this API is also required when the exit is made from the Stop mode. Note that the Frozen pin states and configurations are not maintained on an external reset (XRES) event.

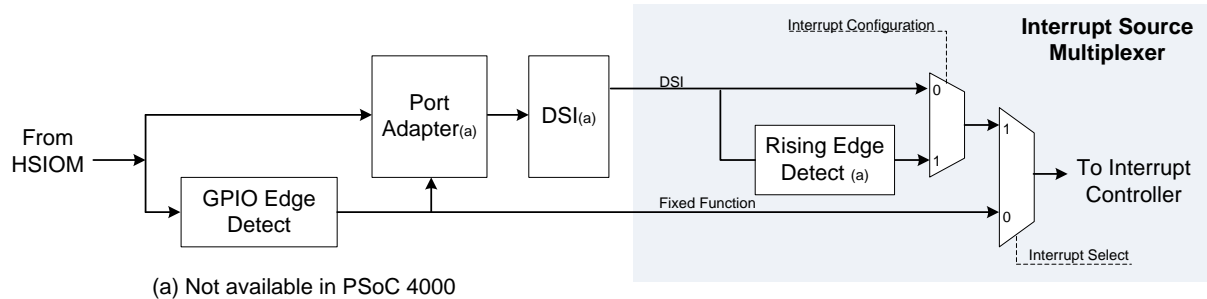
`CySysPmFreezeIo()` and `CySysPmUnfreezeIo()` are also useful in Deep-Sleep mode. An example of use of this feature is shown in the [Control Register Handling in Deep-Sleep](#) section. The UDB-based components, such as the control registers, are not active and lose the data in Deep Sleep, Hibernate, and Stop modes. If the Control Register is driving a pin, a glitch can occur when the PSoC device enters or exits these modes if the last state is '1'. To avoid this glitch, the GPIO should be frozen before entering the low-power mode.

For information on the low-power modes, see [AN86233 – PSoC 4 Low-Power Modes and Power Reduction Techniques](#).

## 7.4 GPIO Interrupt

Figure 13 shows the detailed diagram on the signal path from HSIOM to the Interrupt Controller.

Figure 13. GPIO Interrupt Signal Routing



At each of the 32 interrupt lines of the Interrupt Controller in the processor core, there is an “Interrupt Source Multiplexer”. This multiplexer block selects the source of interrupt and provides an option of rising-edge detection or the direct connection to the Interrupt Controller. There are two sources of interrupts-

1. Fixed Function Source
2. DSI Source

The “Interrupt Select” line selects the DSI or the fixed function source. The “Interrupt Configuration” selects the direct connection or the rising-edge detection logic route to connect to the Interrupt Controller.

A fixed function interrupt source has a fixed-interrupt vector; this means that the interrupt source has a dedicated connection to one of the 32 interrupt lines of the Cortex M0. Interrupt source on this route is directly connected to the Interrupt Controller. When the interrupt source is routed through the DSI, the vector selection is not fixed. This routing also provides an option of rising-edge detection in addition to the direct connection.

**Note:** The interrupt vector table is available in the Interrupts chapter of the [Technical Reference Manual](#).

The use of Interrupt Source Multiplexer is not limited to the GPIO interrupts; it is also used for all other sources. The GPIO interrupt, in addition to the resources present in the Interrupt Source Multiplexer, makes use of its own GPIO Edge Detect block as Figure 13 shows.

The GPIO interrupt signal from the HSIOM is routed in the following ways:

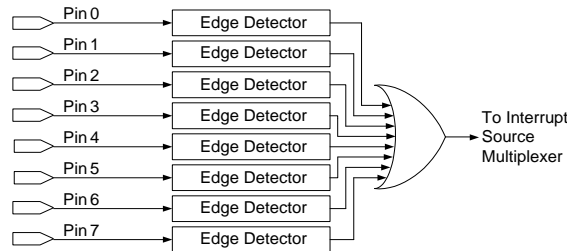
- Route 1: Fixed function route through the GPIO Edge Detect block with the Interrupt Source Multiplexer configured to direct connection
- Route 2: DSI route through the GPIO Edge Detect block with the Interrupt Source Multiplexer configured to rising-edge
- Route 3: DSI route through the GPIO Edge Detect block with the Interrupt Source Multiplexer configured to direct connection
- Route 4: DSI route bypassing the GPIO Edge Detect block with the Interrupt Source Multiplexer configured to rising-edge
- Route 5: DSI route bypassing the GPIO Edge Detect block with the Interrupt Source Multiplexer configured to direct connection

The section [Pins Component Interrupts](#) explains how different routes are configured.

Figure 14 shows the GPIO Edge Detect block. This block detects rising-edge, falling-edge, and both edges in the incoming GPIO signal. Individual GPIO interrupt signals within a port are ORed together to generate a single interrupt request. Thus, there is one interrupt vector for each port.



Figure 14. GPIO Edge Detect



As it is clear from [Figure 14](#), when the interrupt is triggered, the interrupt source is required to be identified. PSoC 4 provides a status register to identify the interrupting pin. After reading the status register, it is important to clear it whenever the GPIO Edge Detect logic is used to avoid the following behavior:

1. Single interrupt trigger and non-responsive to subsequent requests when the Interrupt Source Multiplexer is configured to rising-edge. This scenario will occur if Route 2 is used.
2. Repetitive interrupts for a single request when the Interrupt Source Multiplexer is configured to direct connection. This scenario will occur if Route 1 or Route 3 is used.

When the GPIO interrupt takes the route without the GPIO Edge Detect block, there is no need to clear the interrupt. However, when the rising-edge detection logic in the Interrupt Source Multiplexer is also bypassed, it results in a level type interrupt (Route 5). In this case, the interrupt is triggered repeatedly as long as the pin signal is high. Thus, it is recommended to configure the Interrupt Source Multiplexer to a rising-edge interrupt when the GPIO Edge Detect block is bypassed (Route 4).

**Note:** The GPIO interrupt logic continues to function in the Sleep, Deep Sleep, and Hibernate modes; thus, any pin can be used as a wakeup source. A dedicated wakeup pin, P0[7], is available to wake the device from Stop mode in PSoC 4200 / PSoC 4100 and PSoC 4 M parts. For a PSoC 4 BLE device, the wakeup pin is P2[2].

#### 7.4.1 Limitations in GPIO Interrupt

1. Port 4 and higher ports do not have a Port Adapter. Thus, pin interrupt via DSI routing is not possible for these port pins.
2. PSoC 4000 and PSoC 4100/PSoC 4200 have one interrupt vector for each port. PSoC 4 BLE does not have a dedicated interrupt vector for the ports beyond Port 5 and PSoC 4 M does not have one for the ports beyond Port 4. However, a common port interrupt vector is allocated, which gets triggered when any port interrupt becomes active. See the [Pins Component datasheet](#) to understand how to use this common port interrupt.

An example project is shown in the [Pin Interrupt](#) section, which explains how to use the GPIO interrupt. To understand interrupts in general, see the application note [AN90799 – PSoC 4 Interrupts](#).

## 8 Overvoltage-Tolerant (OVT) Pins

Pins P5[0] and P5[1] in PSoC 4 BLE and Port 6 in PSoC 4 M are the OVT pins. These are similar to regular GPIOs with the following additional features:

1. Overvoltage-tolerant - There are no ESD clamp diodes at the OVT pins. In addition, the GPIO-OVT cell has the hardware to compare the I/O supply voltage (VDDIO) and the pin voltage. If the pin voltage exceeds the VDDIO voltage, the output driver is disabled and the pin is tristated. This results in negligible current sink at the pin. Note that the input buffer data, during overvoltage conditions at the pin, will not be valid if the external source's specification of VOH and VOL do not match with the trip points of the input buffer.
2. Provides better pull-down drive strength as compared to a regular GPIO
3. Serial Communication Block (SCB) - When configured as I<sup>2</sup>C and its lines routed to OVT pins, it meets the following I<sup>2</sup>C specifications:
  - a) Fast Mode Plus IOL Specification
  - b) Fast Mode and Fast Mode Plus Hysteresis and minimum fall time specifications

For more details on the I/O hardware, refer to the I/O System chapter of the [Technical Reference Manual](#).

## 9 GPIO Pins in PSoC Creator

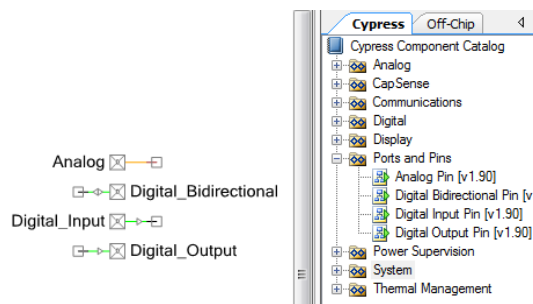
This section describes how to use PSoC Creator to configure and use GPIO pins.

### 9.1 Pins Component Symbols

The Pins Component is the recommended method for connecting internal PSoC resources to a physical pin. It allows PSoC Creator to automatically place and route the signals within the PSoC device based on the chosen pin configuration.

The standard Cypress Component Catalog contains four predefined GPIO configurations in the Ports and Pins class of symbols: analog, digital bidirectional, digital input, and digital output. Drag one of these Components to the schematic to add a pin to the project, as [Figure 15](#) shows.

Figure 15. Pins Component Symbol Types in PSoC Creator



### 9.2 Pins Component Customizer

Each component in PSoC Creator comes with a customizer to configure the component. [Figure 16](#) shows the pin component customizer, which is accessed by double-clicking on the component.

Figure 16. Pin Component Customizer

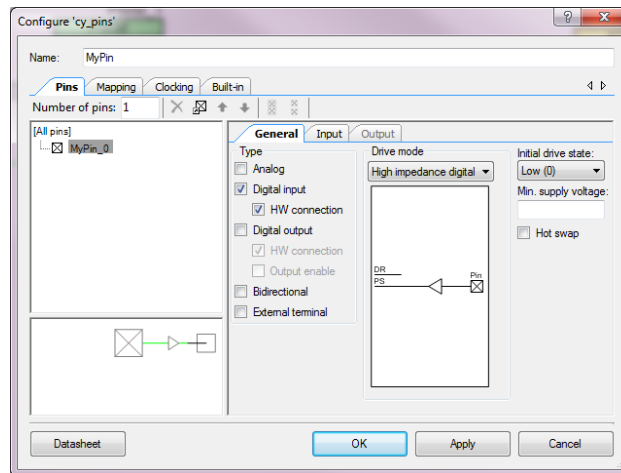


Table 3 describes some of the parameters in the Pin customizer. For details of all the parameters, see the [Pins Component datasheet](#).

Table 3. Pin Component Settings

Setting	Description
General Tab -> Type	This setting configures the pin type. Possible options are: Analog Digital input with or without hardware (HW) connection Digital output with or without HW connection and output enable Bidirectional pin.  When the digital input or output is configured with no HW connection, it means, the pin state is controlled by the CPU. Note that more than one selection can be made at once. For example, a pin can be configured for both Analog and Digital Input at the same time.
General Tab -> Drive Mode	This setting configures the pin with one of the eight drive modes described in the <a href="#">GPIO Pin Basics</a> section. <a href="#">Figure 17</a> lists the drive mode options in the pin customizer.
General Tab -> Initial Drive State	The “Initial drive state” parameter sets the data register value. This value is reflected at the pin if it is software-driven, given that the pin is set with an appropriate drive mode. If the pin is in output mode with “HW connection” enabled and “Output enable” disabled, the “initial drive state” acts as the enable control. Setting the initial state to 1 enables the pin, which is done as default value by PSoC Creator, as shown in <a href="#">Figure 17</a> . If the pin is configured as input, initial drive state can still be useful. For example, if resistive pull-up is required at the input pin, then the drive mode should be configured to “Resistive pull up” with initial state as high in order to turn on the pull up path through the resistor. Likewise, for resistive pull down, the initial drive state should be set to low to enable the pull down path.
Input Tab -> Threshold	CMOS and LVTTL input threshold setting is for an entire port. There are three options as <a href="#">Figure 18</a> shows. The “CMOS or LVTTL” option allows the PSoC Creator tool to select CMOS or LVTTL depending on the threshold setting for other pins in the port.
Input Tab -> Interrupt	This setting configures the GPIO Edge Detect block described in the <a href="#">GPIO Interrupt</a> section. For more details on this setting, refer to <a href="#">Pins Component Interrupts</a> .

Figure 17. Pin Drive Mode Setting and Initial Drive State

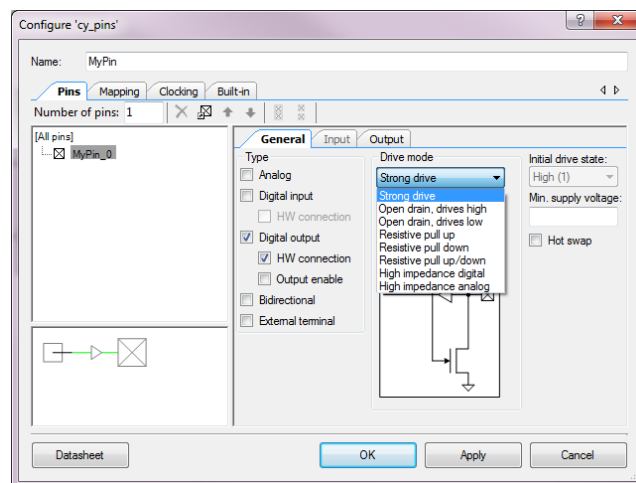
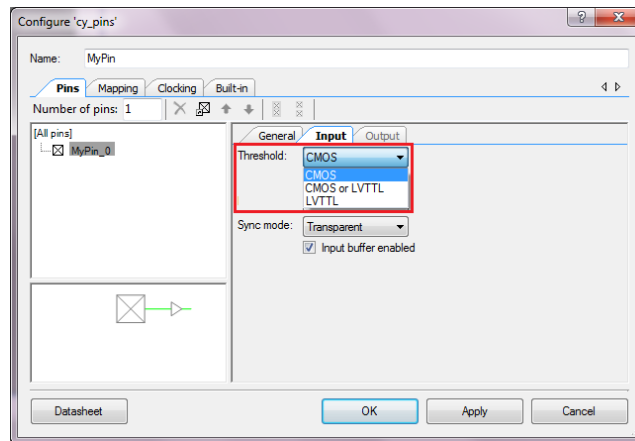


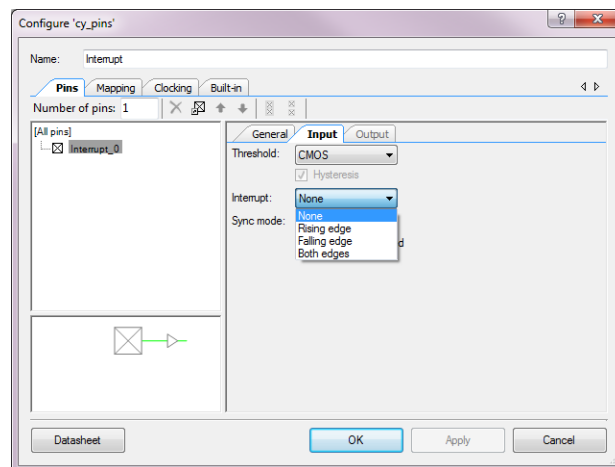
Figure 18. Pin Input Threshold Selection



### 9.3 Pins Component Interrupts

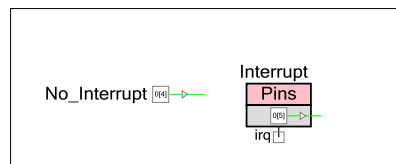
The Interrupt parameter in the pin customizer configures the GPIO Edge Detect block described in the [GPIO Interrupt](#) section.

Figure 19. Interrupt Configuration in PSoC Creator



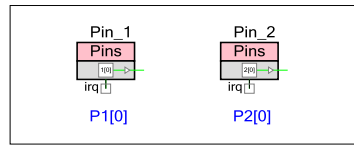
The Pins Component symbol changes when interrupts are enabled, as [Figure 20](#) shows.

Figure 20. Pins Component Symbol with Interrupts Enabled



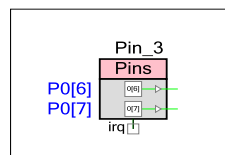
Note that you can use only one Pin Component with each physical GPIO port if the interrupt is enabled. The reason for this limitation is that all pin interrupts in a port are ORed together, as described in the [GPIO Interrupt](#) section. Therefore, only one IRQ signal can be shown on the schematic per port. For example, consider two Pin Components with interrupts enabled, as [Figure 21](#) shows. These components cannot be mapped to pins in the same physical port.

Figure 21. Two Pins Components with Interrupts Enabled



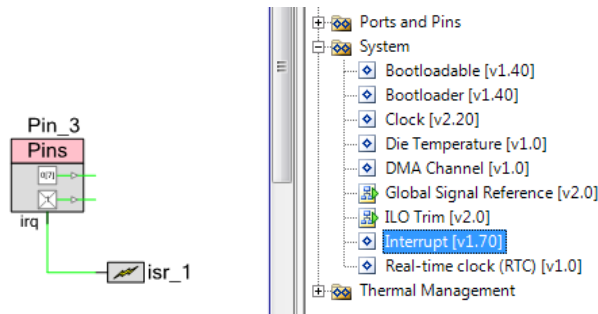
PSoC Creator will not allow you to assign the two components to the same port. The accepted method is to assign multiple pins to the same component, as [Figure 22](#) shows. This ensures that there is only one IRQ signal in the schematic for that physical port. You can still assign each pin its own interrupt edge type. The only limitation is that the pins must be contiguous in the same port. The interrupt source should be identified in the ISR; see the [Pin Interrupt](#) section.

Figure 22. Multiple Pins in the Same Port with Interrupts Enabled



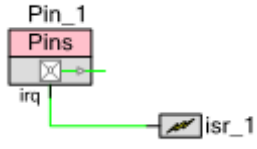
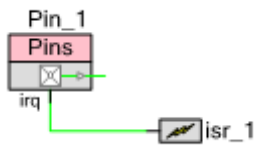
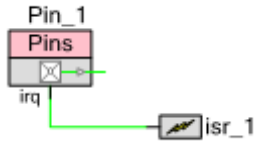


The IRQ of the pin component should be connected to the Interrupt component. This routes the GPIO interrupt signal to the interrupt controller. The interrupt component is in the Component catalog shown in [Figure 23](#).

Figure 23. Interrupt Component in the Catalog



The interrupt component configures the “Interrupt Source Multiplexer” to either direct connection (shown as “level” in the Interrupt Component customizer) or the rising-edge. The GPIO interrupt architecture is described in the [GPIO Interrupt](#) section along with the different routes available for the interrupt signal. Different routes are configured with help of the pin component and the interrupt component customizer settings as [Table 4](#) shows.

Table 4. GPIO Interrupt Configurations

Schematic	Interrupt setting in Pin Component	Interrupt Component Setting	Route	Details
	Rising-Edge or Falling-Edge or Both Edges	Level	Route 1	Interrupt on edges depending on the Pin Component Setting. It uses a fixed interrupt vector depending on the selected port. In this configuration GPIO interrupt status register should be cleared; otherwise, interrupts are triggered repeatedly on a single interrupt request.
	Rising-Edge or Falling-Edge or Both Edges	Rising-Edge	Route 2	Interrupt on edges depending on the Pin Component Setting. In this configuration GPIO interrupt should be cleared; otherwise, interrupt is triggered only once. Interrupt vector is not fixed.
	Rising-Edge or Falling-Edge or Both Edges	Level	Route 3	This is similar to Route 1. However, Route 3 is taken only if the interrupt vector is forced on to a desired DSI vector line. See the application note <a href="#">AN90799 – PSoC 4 Interrupts</a> to know how to force the interrupt vector.
	Disabled	Rising-Edge	Route 4	This configuration provides rising-edge interrupt. In this case, there is no need to clear the interrupt.
	Disabled	Level	Route 5	This configuration provides Level interrupt. Note that the interrupt is triggered repeatedly as long as the pin signal is high. In this case also, there is no need to clear the interrupt.

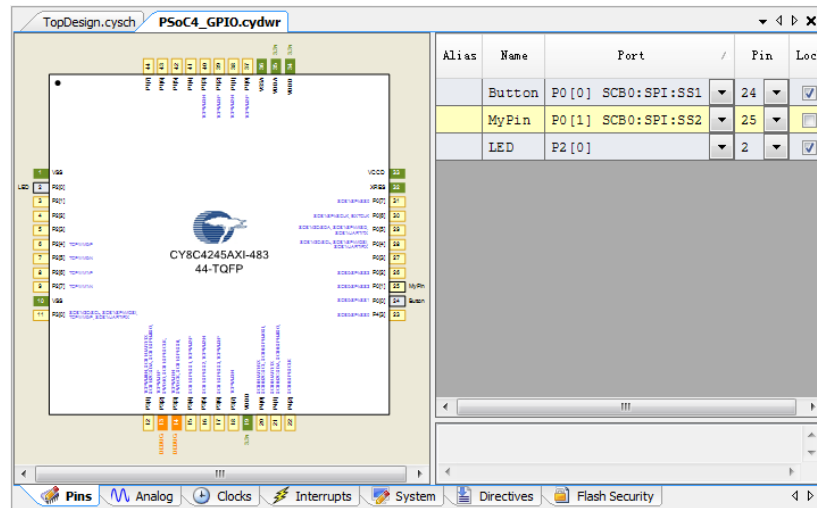
## 9.4 Manual Pin Assignments

You can use the **Pins** tab of the Design-Wide Resources (DWR) window to assign a Pins Component to a physical pin. PSoC Creator automatically assigns pins if the user does not choose any but this may lead to a pin placement that is more difficult to route on a PCB.

Figure 24 shows three assigned pins. The pins highlighted in gray are manually assigned and the pin highlighted in yellow is automatically assigned. Selecting the **Lock** option prevents the pin from being reassigned by PSoC Creator.

PSoC Creator makes it simple to reassign pins as needed, but you should consider pin selection before the boards are designed.

Figure 24. Pin Assignment in DWR Window



**Note:** PSoC Creator can use the unused pin switches for routing the analog signals. This is configured using the "Unused Bonded I/O" parameter in the **System** tab of the `.cydwr` file. Refer PSoC Creator Help for more details.

## 9.5 PSoC Creator APIs

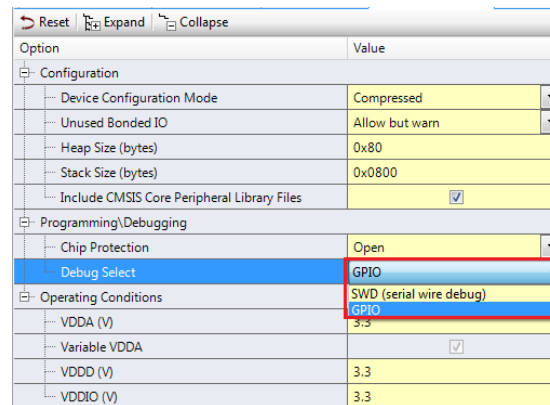
Cypress provides a set of API functions that you can use to control GPIOs dynamically through the firmware. The API for the Pins Component enables access on both a Component-wide and per-pin basis. See the "API" section of the [Pins datasheet](#) for more details.

Per-pin API functions, which are provided as part of `cy_boot` in the `cypins.h` file, are documented in the "Pins" section of the PSoC Creator System Reference Guide (**Help > Documentation > System Reference**). You can use these functions to control the configuration registers for each physical pin.

## 9.6 Debug Logic on GPIO Pins

The PSoC 4 serial wire debug (SWD) pins are shared on the port pins. Refer to the respective [device datasheet](#) for more information on the debug port pins. The debug function, however, can be disabled and the pins can be used as regular GPIOs by setting the “Debug Select” option to “GPIO” in the **System** tab of the DWR window as [Figure 25](#) shows.

Figure 25. Debug Port Disabled

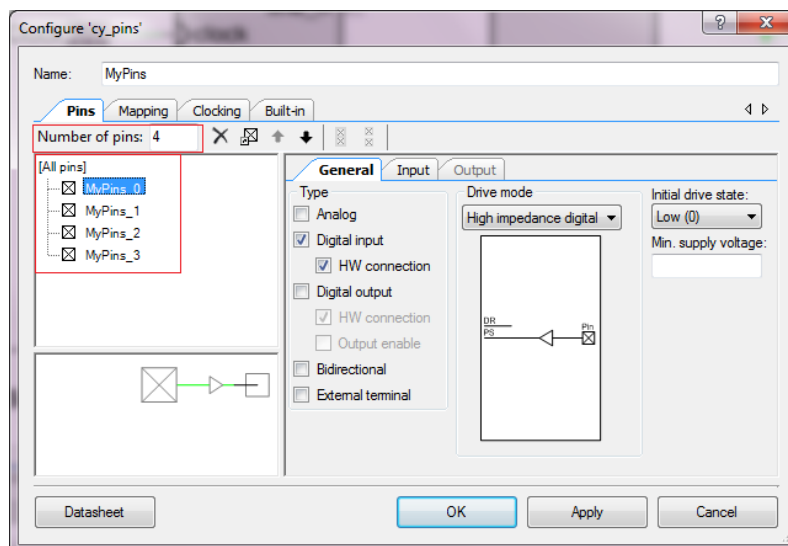


Note that disabling the debug interface does not affect the ability to program the device.

## 9.7 Add Multiple GPIO Pins as a Logical Port

In PSoC Creator, you can organize a group of as many as 36 pins into a logical port, which can then be referenced in code by the port's defined name. All the pins may be part of the same physical port, or they may form separate physical ports. In the Pin component customizer, set the **Number of Pins** required in a port. The pins appear in the list below the field as [Figure 26](#) shows. Each pin can be configured independently. Select **[All Pins]** to configure every pin in the Component with the same settings.

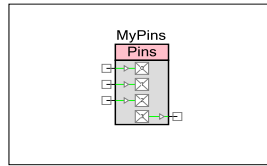
Figure 26. One of Four Pins Configured as a Digital Input





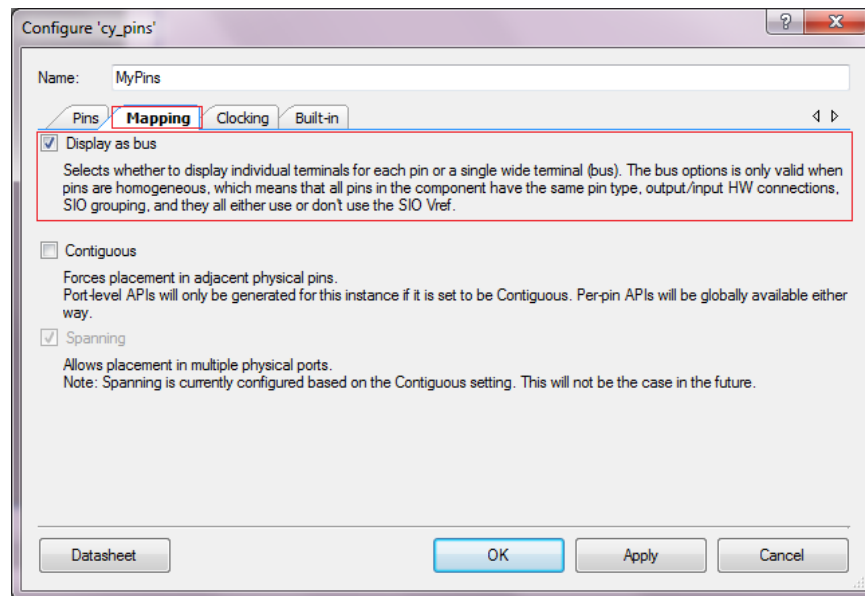
If the number of pins is configured to '4' with three digital inputs and one digital output, the schematic symbol appears as shown in [Figure 27](#).

Figure 27. Pins Component in Port Configuration



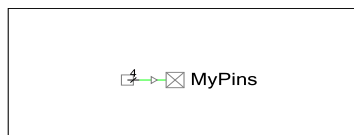
There is an option to display the port with bus instead of individual pin terminals. Select **Display as Bus** in the **Mapping** tab of the pin configuration window to display the port with a bus, as [Figure 28](#) shows. Note that all pins must be of the same type to display a bus.

Figure 28. Display as Bus Option



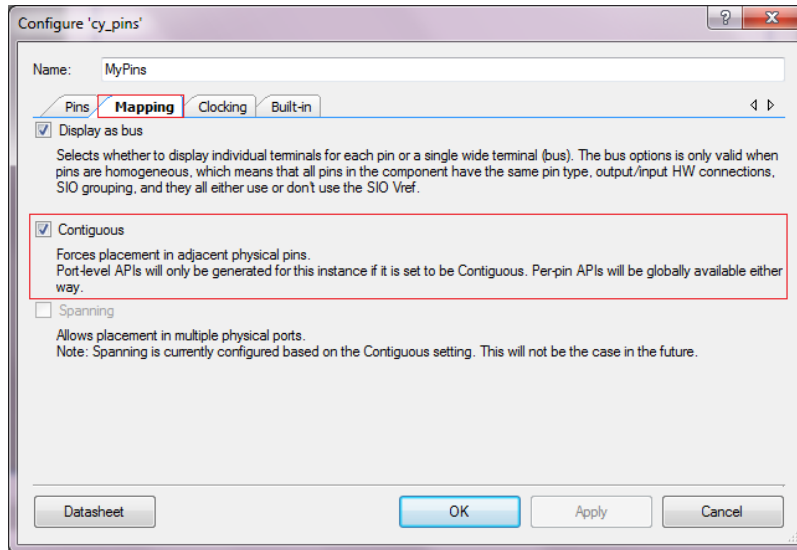
If the **Number of Pins** is configured to four digital outputs, the schematic symbol appears as shown in [Figure 29](#).

Figure 29. Four Pins Displayed with Bus



The pins with the bus terminal can be forced to map to adjacent pins by enabling **Contiguous** in the **Mapping** tab as Figure 30 shows.

Figure 30. Contiguous Pin Placement Option



When you select **Contiguous**, PSoC Creator modifies the list of available pin options to match the port's configuration, as Figure 31 shows. When the Contiguous option is disabled, any pin can be selected. When the Contiguous option is enabled, only adjacent pins can be selected as Figure 31 shows.

Figure 31. Pin Placement with Contiguous Disabled/Enabled

Alias	Name	Port	Pin	Lock
	MyPins [0]		▼	▼
	MyPins [1]		▼	▼
	MyPins [2]		▼	▼
	MyPins [3]		▼	▼

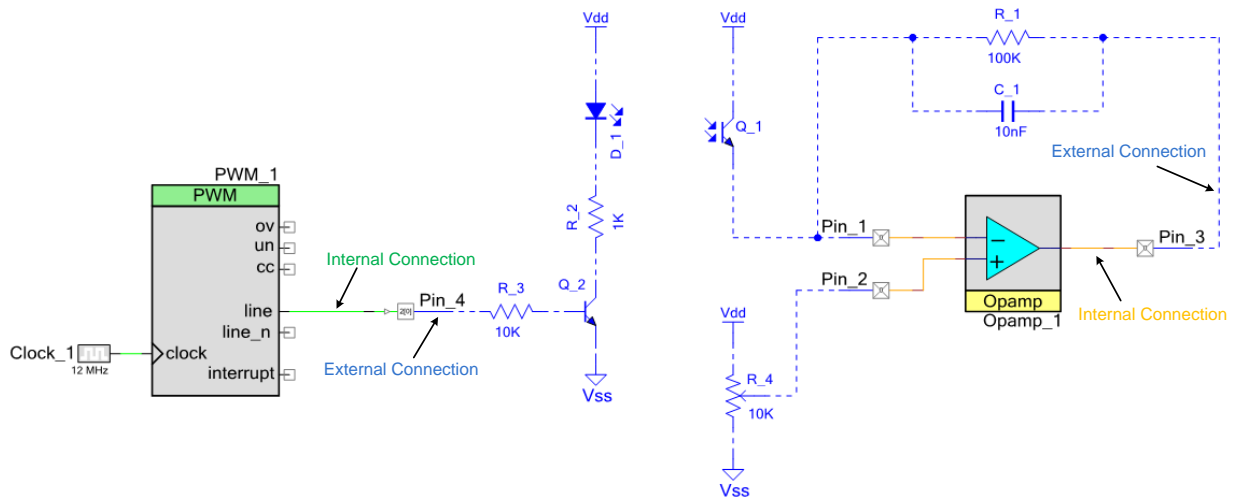
Alias	Name	Port	Pin	Lock
	\MyPins[3:0]\		▼	▼
	P0 [3:0]			
	P0 [4:1]			
	P0 [5:2]			
	P0 [6:3]			
	P0 [7:4]			
	P1 [3:0]			
	P1 [4:1]			
	P1 [5:2]			
	P1 [6:3]			
	P1 [7:4]			

These features are described in more detail in the pin configuration window and the Pins Component datasheet.

### 9.8 Represent Off-Chip Components

The Off-Chip Components Catalog provides a way to mix external and internal components on the same schematic, as Figure 32 shows. This makes it possible to improve documentation and convey clearly how the internal schematic fits in the entire design. Off-chip components serve the same function as comments in the code – they do not change the functionality of the PSoC design but, instead, provide a clearer picture of the entire system.

Figure 32. Design with Off-Chip Components



In the design shown in Figure 32, PWM\_1 and Opamp\_1 are internal blocks of the device. These blocks are connected to the external components using pins Pin\_1 through Pin\_4. Green and orange wires are the internal connections (green for digital signals and orange for analog signals); whereas, blue wires and components are external to the device. To make the connections with the external components in the schematic, enable the “External Terminal” parameter in the Pin component customizer as Figure 33 shows. This brings out an additional terminal on the schematic as Figure 34 shows.

Figure 33. Enabling External Terminal

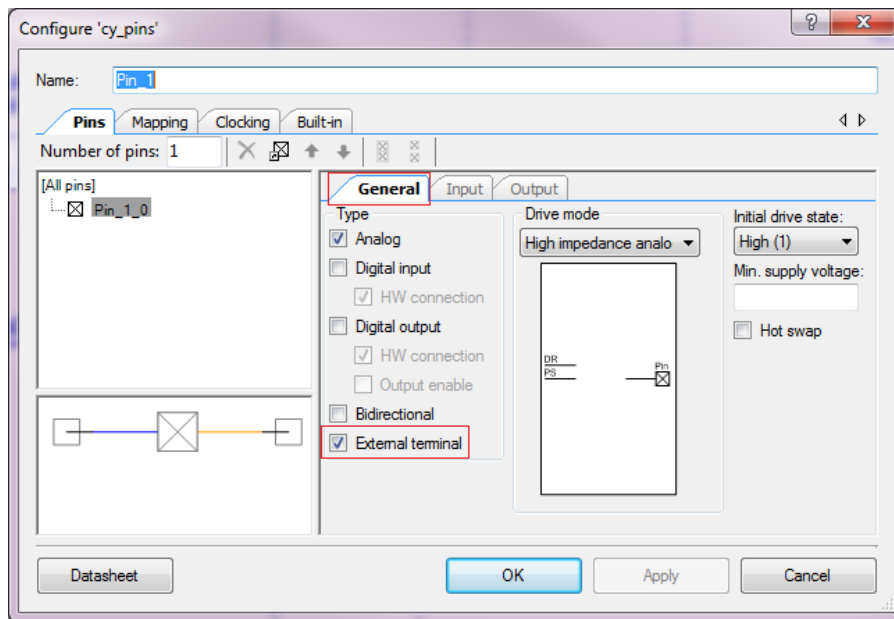
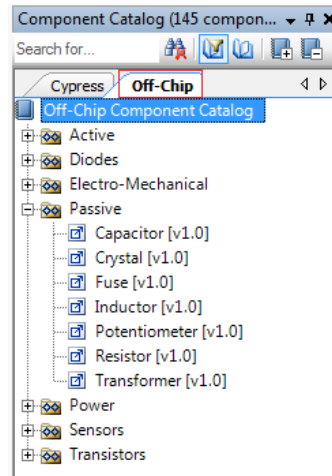


Figure 34. Pin Component With Internal and External Terminal



The components, to connect to the external terminals on the schematic, are available in the Off-Chip tab in the Components Catalog as Figure 35 shows.

Figure 35. Off-Chip Components Catalog



The components in this catalog cover those that are most likely to be connected to the pins of a PSoC device on the board. These components consist of resistors, capacitors, transistors, inductors, switches, and others. Drag the component and place it on the schematic as it is done in the case of internal components.

## 10 GPIO Tips and Tricks

This section provides practical examples of how to use GPIO pins. All these examples are included in the PSoC Creator project provided with this application note.

Table 5. PSoC Creator Projects

#	Section	Project	PSoC 4000	PSoC 4100	PSoC 4200	PSoC 41xx-BL	PSoC 42xx-BL	PSoC 4100M	PSoC 4200M
1	Toggle an LED	Project1_ToggleLED	✓	✓	✓	✓	✓	✓	✓
2	Read an Input and Write to an Output	Project2_ReadingPin	✓	✓	✓	✓	✓	✓	✓
3	Drive an Output from a Digital Logic Gate	Project3_HWDive	x	x	✓	x	✓	x	✓
4	Set the GPIO Input/Output Synchronization	Project4_GPIOSynchronization	x	✓	✓	✓	✓	✓	✓
5	Toggle GPIOs Faster with Data Registers	Project5_FastGPIOUpdate	✓	✓	✓	✓	✓	✓	✓
6	Configure GPIO Output Enable Logic	Project6_OutputEnable	x	x	✓	x	✓	x	✓
7	Pin Interrupt	Project7_PinInterrupt	✓	✓	✓	✓	✓	✓	✓
8	Configure GPIO Interrupt Settings with Firmware	Project8_GPIOIntConfig	✓	✓	✓	✓	✓	✓	✓
9	Using Both Analog and Digital on a GPIO	Project9_GPIOAnalogDigital	✓	✓	✓	✓	✓	✓	✓
10	Gang Pins for More Drive/Sink Current	Project10_GangingPins	x	x	✓	x	✓	x	✓
11	Control Register Handling in Deep-Sleep	Project11_ControlRegInDeepSleep	x	x	✓	x	✓	x	✓

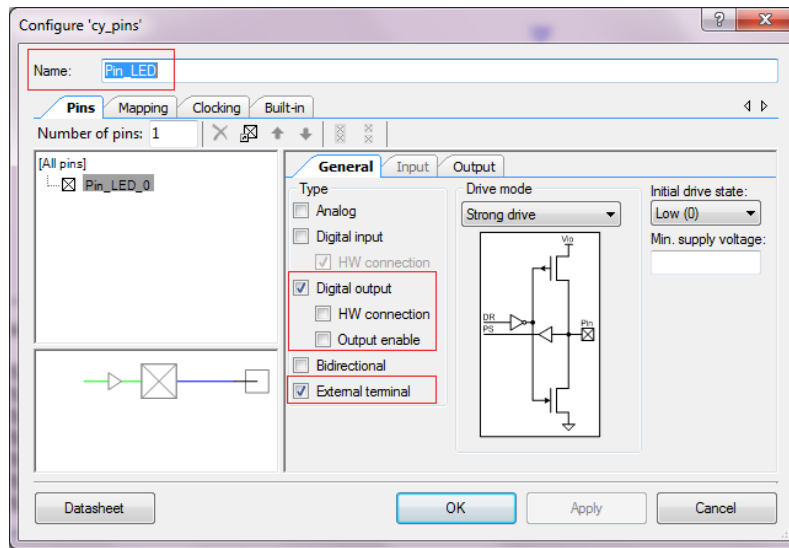
The Cypress development kits, listed in [Appendix B: PSoC 4 Development Boards](#), can be used for testing these projects.

### 10.1 Toggle an LED

The simplest use of a GPIO is to set the output of a pin HIGH or LOW in the firmware. This example demonstrates how to set the output to toggle an LED using the Pins Component API.

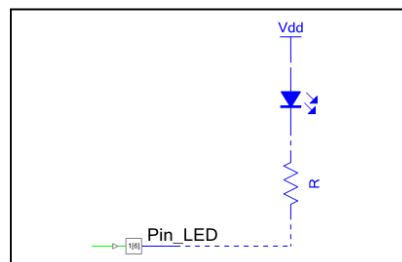
1. Place a Digital Output Pin Component in the project schematic.
2. Name the Component “Pin\_LED” and disable the hardware connection as [Figure 36](#) shows.

Figure 36. Pin\_LED Configuration



3. Enable the external terminal to connect to the external components on the schematic
4. Assign it to a physical pin (this example uses P1[6]) in the **Pins** tab of the DWR window).
5. Connect the physical pin to an LED. Note that the LED and resistor R are off-chip components. The Top Design schematic is shown in [Figure 37](#).

Figure 37. Toggle an LED Example Schematic



6. In *main.c*, use the Component API to set the output as follows:

```

for(;;)
{
    /* Set LED output to logic HIGH */
    Pin_LED_Write(1u);

    /* Delay of 500 ms */
    CyDelay(500u);

    /* Set LED output state to logic LOW */
    Pin_LED_Write(0u);

    /* Delay of 500 ms */
    CyDelay(500u);
}

```

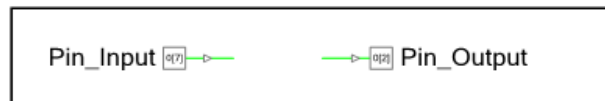
- Build the project and program the PSoC 4 device.  
The result is an LED blinking at a frequency of 1 Hz.

## 10.2 Read an Input and Write to an Output

This example demonstrates how to read from and write to a GPIO pin with the Component APIs. The output pin drives the inverse of the input pin state.

- Place two pins in the project schematic—one Digital Input Pin and one Digital Output Pin with hardware connection disabled—as [Figure 38](#) shows.

Figure 38. Input and Output Example Schematic



- Assign the pins for Pin\_Input and Pin\_Output in .cydwr
- Use the Component APIs to set the state of Pin\_Output based on Pin\_Input as follows:

```

for(;;)
{
    /* Set the output pin with an inverted value of input pin */
    Pin_Output_Write(~Pin_Input_Read());
}

```

- Build the project and program the PSoC 4 device.  
You can test this project by feeding a square wave from a signal generator to the Pin\_Input. The signal at Pin\_Output will be an inverted form of Pin\_Input signal.

## 10.3 Drive an Output from a Digital Logic Gate

The previous example showed the use of the processor core to read a pin and set another pin with an opposite of the read value. This example demonstrates the same task but with the use of configurable digital resources known as Universal Digital Blocks (UDBs). In this example, an input pin signal is routed to the NOT gate and the output of the NOT gate is routed to another pin. Follow these steps to create the project:

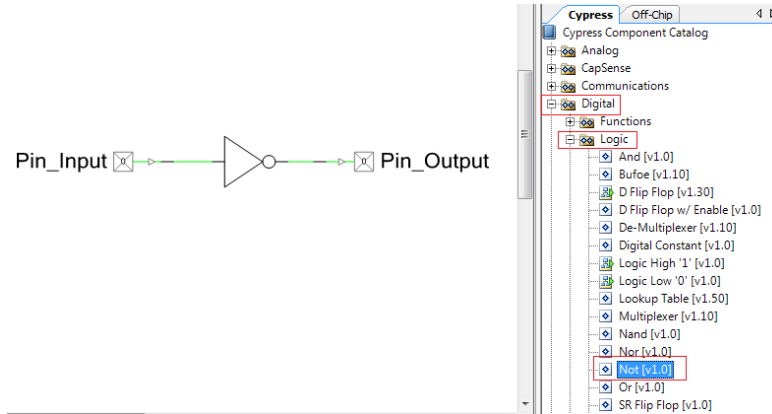
- Place two pins in the project schematic—one Digital Input Pin and one Digital Output Pin with hardware connection enabled—as [Figure 39](#) shows.

Figure 39. Input and Output Pins with HW Connection Enabled



- Place a NOT gate and connect to the pins as [Figure 40](#) shows.

Figure 40. NOT Gate Connection



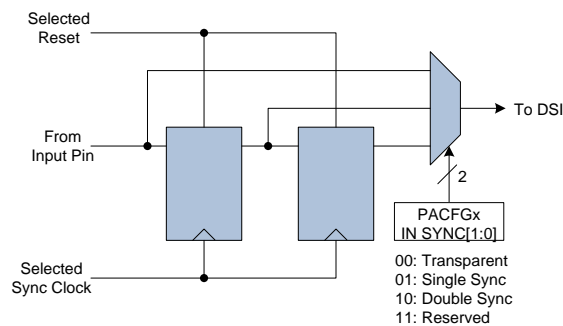
- Assign the pins for Pin\_Input and Pin\_Output in .cydwr
- This project does not require any code. Build the project and program the PSoC 4 device.

Similar to the previous project, you can test this project by feeding a square wave from a signal generator to the Pin\_Input. The signal at Pin\_Output will be an inverted form of Pin\_Input signal.

## 10.4 Set the GPIO Input/Output Synchronization

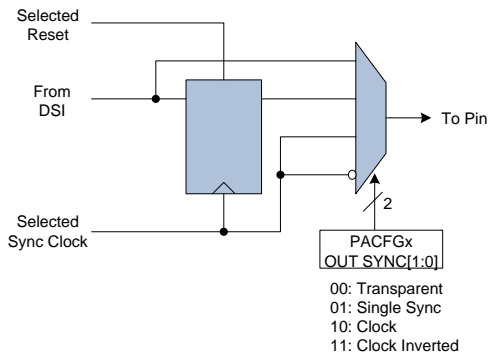
For digital input and output signals, the GPIO provides synchronization with an internal clock, HFCLK, or a digital signal as a clock in the PSoC 4 parts (except PSoC 4000). In addition, it provides the configuration for clock enable and synchronization logic reset. Port adapter logic is used for input and output synchronization, as [Figure 41](#) and [Figure 42](#) show.

Figure 41. Input Synchronization in PSoC 4



As [Figure 41](#) shows, the input synchronization circuit provides the options of transparent, single sync, and double sync.

Figure 42. Output Synchronization in PSoC 4



The output synchronization circuit provides the options of transparent, single sync, clock, and clock inverted, as shown in Figure 42. Clock and clock inverted route the sync clock to the output pin. These are set in the Pins Component customizer, as Figure 43 shows.

The synchronizer clock can be configured as HFCLK, an external signal (from DSI), or one of the pin signals. The synchronizer block reset signal can be an external signal (from DSI) or one of the pin signals. These are configured in the Pins Component customizer from the Clocking tab, as Figure 44 shows.

For more information on the parameters in the Clocking tab of the Pins Component customizer, refer to the [Pins Component datasheet](#).

Figure 43. Sync Mode Setting

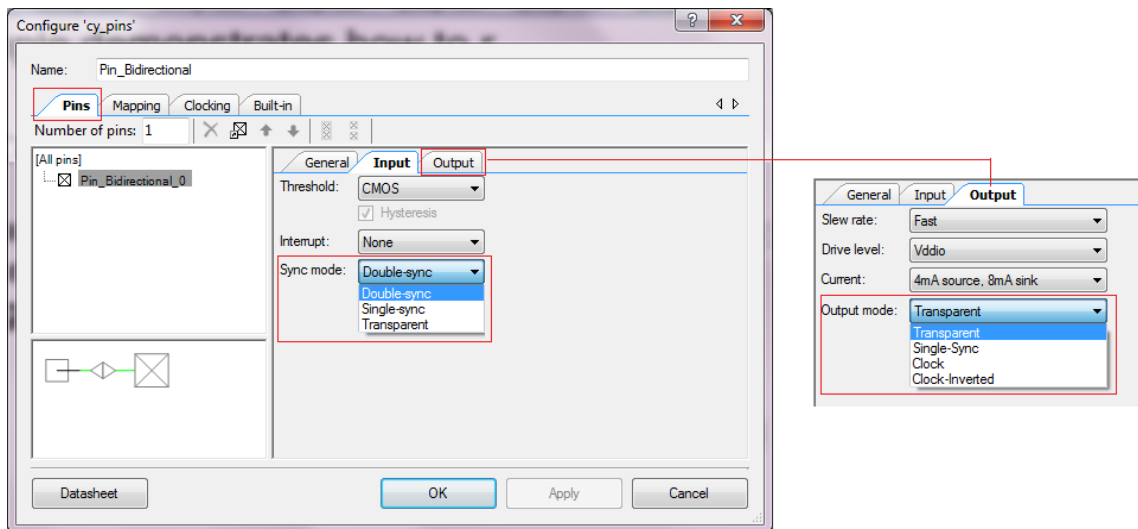
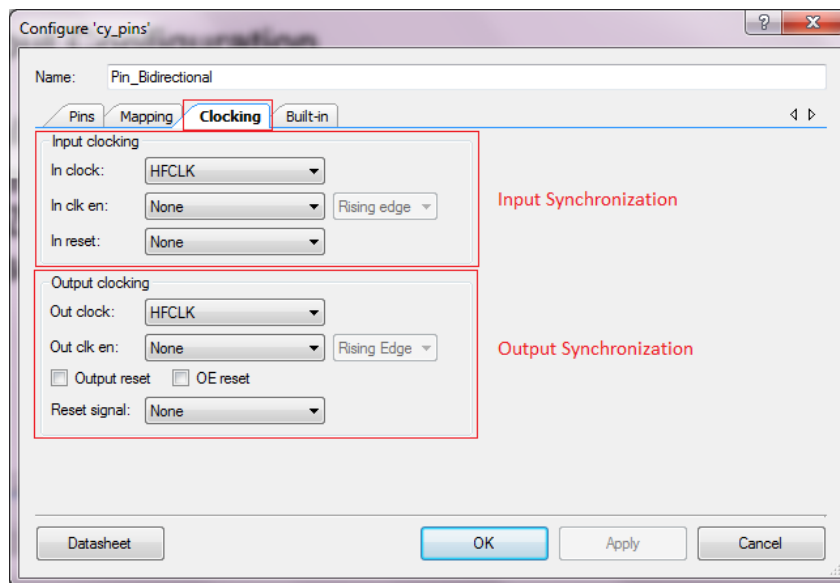




Figure 44. Clock Setting



Pin signals are synchronized using a combination of the UDB port adapter and GPIO blocks. Also, the clock is shared for all the pins of a port for internal clock synchronization. For more information, refer to the [PSoC 4 Architecture TRM](#).

**Note:** The signals at port 4 and higher port pins cannot be synchronized because these ports do not have a UDB port adapter. Therefore, these port pins should always be used in the Transparent mode to avoid an error during build.

The next two examples demonstrate how to set the input/output synchronization.

#### 10.4.1 GPIO Input Synchronization

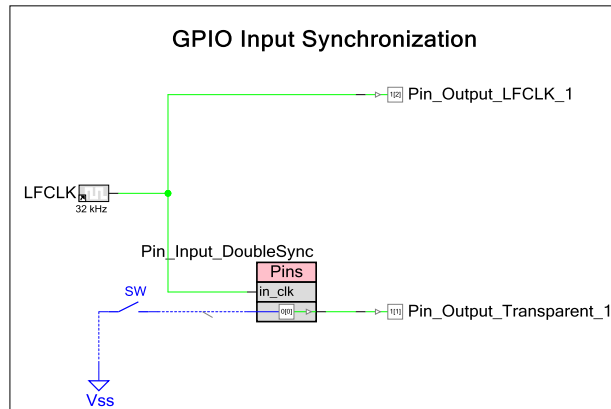
- Place one Digital Input Pin, two Digital Output Pins, and one Clock Component in the project schematic. Configure them as [Table 6](#) shows.

Table 6. Components Configuration

Component	Name	Configuration
Digital Input Pin	Pin_Input_DoubleSync	Drive Mode: Resistive Pull-Up Sync Mode: Double-Sync In Clock (for input sync): External
Digital Output Pin	Pin_Output_LFCLK_1	Output Mode: Transparent
Digital Output Pin	Pin_Output_Transparent_1	Output Mode: Transparent
Clock	LFCLK	Clock Type: Existing Source: LFCLK

- Connect the pins and add the Off-Chip Components, as [Figure 45](#) shows.

Figure 45. GPIO Input Synchronization Example Schematic

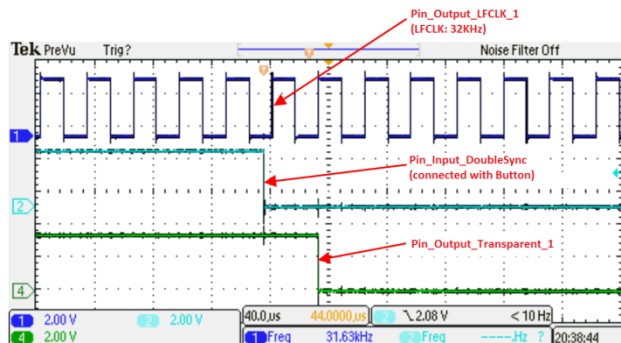


**Note:** Clocks in PSoC 4 cannot be directly connected to a pin terminal, except for SYSCLK and LFCLK. See the "Clocking System" section of the [PSoC 4 Architecture TRM](#) for more information.

3. Assign the pins, and connect the Pin\_Input\_DoubleSync pin to a switch connected to GND.
4. Build the project and program the PSoC 4 device.

When the button connected to Pin\_Input\_DoubleSync is pressed, the signal waveforms occur as [Figure 46](#) shows. The Pin\_Output\_Transparent\_1 pin becomes LOW at the second rising edge of LFCLK because the input is double-synchronized with LFCLK.

Figure 46. Input/Output Signal Waveforms



### 10.4.2 GPIO Output Synchronization

- Place one digital input pin, three digital output pins, and one Clock Component in the project schematic, as [Figure 47](#) shows, and configure them as described in [Table 7](#).

Figure 47. GPIO Output Synchronization Example Schematic

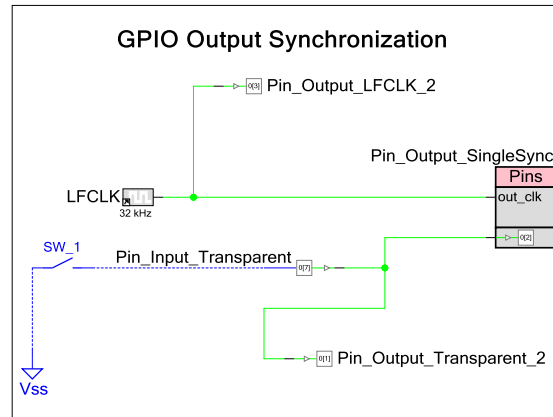


Table 7. Pins Configuration

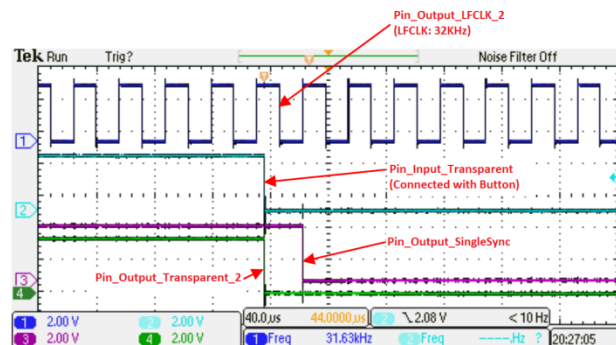
Component	Name	Configuration
Digital Input Pin	Pin_Input_Transparent	Drive Mode: Resistive Pull-Up Sync Mode: Transparent
Digital Output Pin	Pin_Output_LFCLK_2	Output Mode: Transparent
Digital Output Pin	Pin_Output_SingleSync	Output Mode: Single-Sync Out Clock (for output sync): External
Digital Output Pin	Pin_Output_Transparent_2	Output Mode: Transparent
Clock	LFCLK	Clock Type: Existing Source: LFCLK

- Connect the pins as [Figure 47](#) shows.
- Assign the pins, and connect the Pin\_Input\_Transparent to a switch connected to GND. Note that you cannot select Port 4 and higher port pins for Pin\_Output\_SingleSync as they don't support synchronization.
- Build the project and program the PSoC 4 device.

[Figure 48](#) shows the waveforms corresponding to a button press.

The Pin\_Output\_SingleSync pin becomes LOW at the next rising-edge of LFCLK because the output is synchronized with LFCLK. The Pin\_Output\_Transparent\_2 pin becomes LOW at the same time as the Input\_Transparent pin because there is no synchronization.

Figure 48. Input/Output Signal Waveform



## 10.5 Toggle GPIOs Faster with Data Registers

The Component API is the easiest way to control GPIO pins; however, it is not the fastest way.

Take an example of writing logic 1 to Pin\_1 mapped to pin P5[2]. Here is the API function call:

```
Pin_1_Write(1);
```

The equivalent assembly code can be seen in the listing file (main.lst) in the results tab of the project workspace.

```
mov    r0, #1      ;load the value in r0
bl    Pin_1_Write  ;call Pin_1_Write
```

The assembly code of component API Pin\_1\_Write function can also be seen in the listing file-

```
ldr    r3, .L2      ;load the address of Pin_1_DR into r3
ldr    r1, [r3]     ;load the value of Pin_1_DR into r1
mov    r2, #251    ;load 251 into r2 (value depends on location of pin
                  ;in 8 bit wide port, in this case, Pin_1 is on port P5[2])
and    r2, r1      ;AND the values of r2 and r1 and load result back in r2

lsl    r0, r0, #2   ;left shift r0 by two bits and load the result back in
                  ;r0 (this instruction is not present for the pin on LSB)
mov    r1, #4      ;load value of 4 into r1 (depends on the location of
                  ;pin in 8 bit wide port)
and    r0, r1      ;and the value of r0 (contains "value") and r1 and load
                  ;the result in r0
orr    r0, r2      ;or the value of r0 with r2 and load the result back in
                  ;r0
str    r0, [r3]    ;store the result back in Pin_1_DR
bx     lr          ;return to calling function
```

This code takes 20 CPU cycles to write logic 1 to Pin\_1.

Alternatively, you can use the register definitions and masks in the `<pin_name>.h` file that is created for each Pins Component to update the pins quickly.

The following statement sets logic 1 at Pin\_1 mapped to P5[2]. Pin\_1\_DR is the data register of Pin\_1.

```
Pin_1_DR |= Pin_1_MASK
```

In the listing file (main.lst), the above instruction translates into an assembly code as follows:

```
ldr    r3, .L3      ;load the address of Pin_1_DR into r3
ldr    r1, [r3]     ;load value of Pin_1_DR into r1
mov    r2, #4       ;move value of 4 (Pin_1_MASK) into r2
orr    r2, r1       ;Set the bit in Pin_1_DR
strb   r2, [r3]     ;Store it back into Pin_1_DR
```

This code takes eight cycles as against 20 cycles used by the component API.

The component API had firmware overhead for the following actions, which are not required in direct register writes:

- Function call
- Checking the function argument to set the pin to logic 1 or logic 0
- Return from the function

To set, reset, and read the pin using direct register writes, the following macros are provided in PSoC Creator:

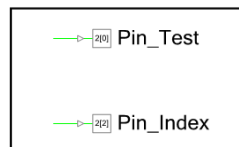
Macro	Description
<code>CY_SYS_PINS_SET_PIN(portDR, pin)</code>	Sets the output value for the pin to logic high <code>portDR</code> is the address of the port data register <code>pin</code> is the pin number (0 to 7)
<code>CY_SYS_PINS_CLEAR_PIN(portDR, pin)</code>	Clears the output value for the pin to logic low <code>portDR</code> is the address of the port data register <code>pin</code> is the pin number (0 to 7)
<code>CY_SYS_PINS_READ_PIN(portPS, pin)</code>	Reads the pin value <code>portPS</code> is the address of the port status register <code>pin</code> is the pin number (0 to 7)

Refer to the System Reference Guide (available from the PSoC Creator Help Menu) for more details on these macros.

Follow these instructions to create a PSoC Creator project that can be used to compare the performance of the API function call and direct register write:

1. Place two digital output pins, with hardware connection disabled, in the project schematic and name them "Pin\_Test" and "Pin\_Index," as Figure 49 shows.

Figure 49. Toggle GPIOs with Data Registers Example Schematic



2. Assign the pins in .cydwr.
3. Add the following code to the `main.c` file. The code sets the Pin\_Index HIGH and toggles Pin\_Test using the Component API. It then sets the Pin\_Index LOW and toggles Pin\_Test using the data register (DR).

```

for (;;)
{
    /* Set IndexPin */
    Pin_Index_Write(1);

    /****** API Call *****/

    /* Set TestPin */
    Pin_Test_Write(1u);

    /* Clear TestPin */
    Pin_Test_Write(0u);

    /* do it again */
    Pin_Test_Write(1u);
    Pin_Test_Write(0u);

    /****** */

    /* Clear IndexPin */
    Pin_Index_Write(0);
}

```

```

/***** Direct Register Writes *****/

/* Set TestPin */
CY_SYS_PINS_SET_PIN(Pin_Test_DR, Pin_Test_SHIFT);

/* Clear TestPin */
CY_SYS_PINS_CLEAR_PIN(Pin_Test_DR, Pin_Test_SHIFT);

/* do it again */
CY_SYS_PINS_SET_PIN(Pin_Test_DR, Pin_Test_SHIFT);
CY_SYS_PINS_CLEAR_PIN(Pin_Test_DR, Pin_Test_SHIFT);

/*****
}

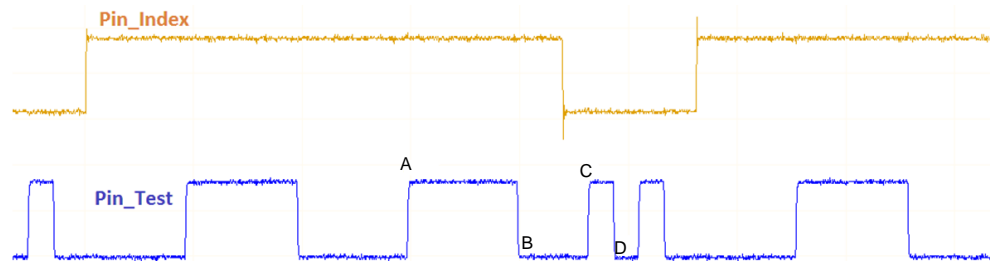
```

**Note** `Pin_Test__DR` is the address of the data register; whereas, `Pin_Test_DR` is the value of the data register. See the pin component .h file in the Source Files of the project workspace (in this case `Pin_Test.h`) to know the macro for the data register address.

The code that writes to the data register is also portable similar to the API call, so that if the pin assignment changes during development, you do not have to change the code.

4. Observe the waveform of the two pins using an oscilloscope, as shown in [Figure 50](#).

Figure 50. Output Signals Waveform



- A. `Pin_Test_Write(1u)`
- B. `Pin_Test_Write(0u)`
- C. `CY_SYS_PINS_SET_PIN(Pin_Test__DR, Pin_Test_SHIFT)`
- D. `CY_SYS_PINS_CLEAR_PIN(Pin_Test__DR, Pin_Test_SHIFT)`

[Figure 50](#) shows that a pin can toggle faster by directly writing to the data register, as opposed to calling the API functions.

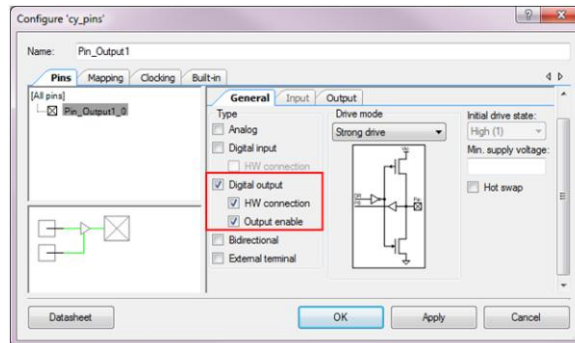
To know about coding techniques to achieve time efficiency, refer to the application note, [AN89610 – PSoC 4 and PSoC 5LP ARM Cortex Code Optimization](#).

## 10.6 Configure GPIO Output Enable Logic

This example demonstrates how to configure and use the output enable logic of a GPIO pin. This project is applicable only for PSoC 4200, PSoC 42xx\_BL, and PSoC 4200M parts.

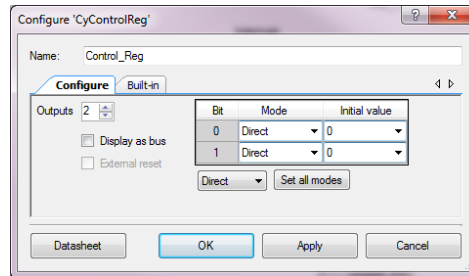
1. Place two Digital Output Pins in the project schematic.
2. Open the configuration dialog for each pin and select the **Output Enable** option, as [Figure 51](#) shows.

Figure 51. Output Enable Selection



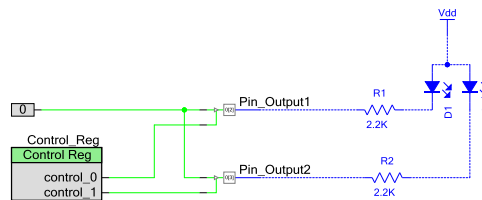
3. Place a Control Register in the schematic.
4. Configure the Control Register for two outputs, as [Figure 52](#) shows.

Figure 52. Control Register Configured With Two Outputs



5. Add one Logic Low '0' Component.
6. Connect the Logic Low to the pins and add the Off-Chip Components for the LEDs, as [Figure 53](#) shows.

Figure 53. Control Register Driving Pins' Output Enable



7. Assign the pins and connect the pins to LEDs.
8. Add the following code to the *main.c* file.

```

uint8 count;

for(;;)
{
    for(count = 0u; count < 4u; count++)
    {
        /* Set Control_Reg Value */
        Control_Reg_Write(count);

        /* Delay for 500ms */
        CyDelay(500u);
    }
}

```



9. Build the project and program the PSoC 4 device.

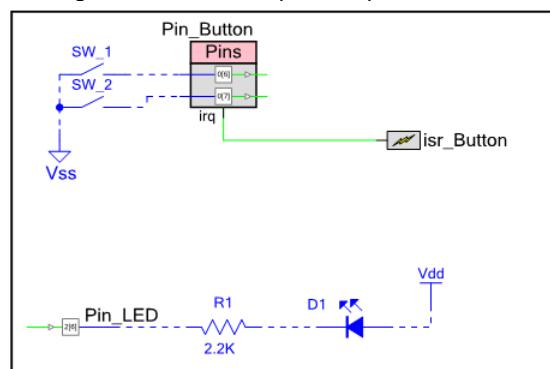
The result is the output of the two pins gated by the state of Control\_Reg, which causes the LEDs to “count” to 3.

## 10.7 Pin Interrupt

This example demonstrates how to use an interrupt generated from two pins in the same port, using the Component APIs. These two pins can use only one IRQ terminal. Thus, the interrupt source must be identified in the ISR.

1. Place two pins in the project schematic: one Digital Input Pin named Pin\_Button and one Digital Output Pin named Pin\_LED.
2. Set Pin\_Button’s number of pins to 2, Drive mode as Resistive Pull Up, and Interrupt as Falling-Edge. This exposes the IRQ terminal.
3. Connect the interrupt component to the irq terminal as shown in Figure 54.

Figure 54. Pin Interrupt Example Schematic



4. Assign the pins in .cywdr
5. Use the Component APIs to set the state of the LED Pin based on Pin\_Button. Copy the following *main.c* code:

```

#define LED_ON (0u)
#define LED_OFF (1u)

/* The flag to enter ISR_Button */
uint8 isrFlag = 0u;

/* The LED state */
uint8 ledState = LED_OFF;

/* ISR for ISR_Button */
CY_ISR(INT_ISR_Button)
{
    /* Set the flag */
    isrFlag = 1u;

    /* Check which pin caused interrupt by reading interrupt status register */
    if(Pin_Button_INTSTAT & (0x01u << Pin_Button_SHIFT))
    {
        /* Triggered by Pin_Button_0 */
        ledState = LED_OFF;
    }
    else
    {
        /* Triggered by Pin_Button_1 */
        ledState = LED_ON;
    }
}

```

```

    }

    /* Clear interrupt */
    Pin_Button_ClearInterrupt();
}

int main()
{
    /* Start Pin ISR */
    isr_Button_StartEx(INT_ISR_Button);

    /* Enable global interrupt */
    CyGlobalIntEnable;

    for(;;)
    {
        /* Check the flag */
        if(0u != isrFlag)
        {
            /* Clear the flag */
            isrFlag = 0u;

            /* Drive the LED with ledState. Led State is updated in ISR */
            Pin_LED_Write(ledState);
        }

        /* Delay 1ms */
        CyDelay(1u);
    }
}

```

In the `main.c` code, `CY_ISR(INT_ISR_Button)` is the interrupt service routine for the pin interrupt.

#### 6. Build the project and program the PSoC 4 device.

The result is that the LED turns OFF when you press the button connected to `Pin_Button_0` and turns ON when you press the button connected to `Pin_Button_1`, but not when you release the buttons. (Note that the switch bounce in the buttons may cause several interrupts on a single button press; see [AN60024 – Switch Debouncer and Glitch Filter with PSoC 3, PSoC 4, and PSoC 5LP](#) for details).

In the `main.c` code, `Pin_Button_INTSTAT` and `Pin_Button_SHIFT` are the functions and constant macros provided by the Pins Component. These are used to check which pin caused an interrupt.

The function `Pin_Button_ClearInterrupt()` clears the interrupt status register.

For more information on interrupts and writing interrupt handlers, refer to [AN90799 – PSoC 4 Interrupts](#).

## 10.8 Configure GPIO Interrupt Settings with Firmware

The GPIO interrupt is configured dynamically by writing to the two bits of the Interrupt Configuration register.

- For PSoC 4000: `GPIO_PRTx_INTR_CFG[2y+1:2y]`
- For other PSoC 4 parts: `PRTx_INTCFG[2y+1:2y]`

where “x” corresponds to the port number and “y” corresponds to the pin number (see [Table 8](#)). You can change the configuration at any time to enable or disable pin interrupts.

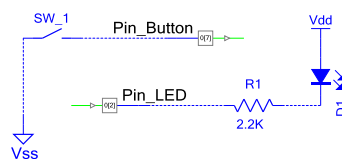
Table 8. GPIO Interrupt Types and Bit Settings

PRTx_INTCFG [2y+1:2y]	Edge Type	Description
0	Disable	Interrupts disabled
1	Rising-Edge	Trigger on rising edge
2	Falling-Edge	Trigger on falling edge
3	Both Edges	Trigger on either edge

In this example, Pin\_Button is configured with a rising-edge interrupt. Once the interrupt occurs, it is configured as a falling-edge interrupt. An LED is toggled whenever the interrupt is triggered.

1. Place a Digital Input Pin and a Digital Output Pin in the project schematic. Add the Off-Chip Components for the LED and button, as Figure 55 shows.

Figure 55. Example Schematic



2. Assign the pins to Pin\_Button and Pin\_LED in cydwr.
3. Configure Pin\_Button as a resistive pull-up pin and connect it to a button.
4. Configure Pin\_LED as a strong drive pin and connect it to an external LED.
5. Add the following code to the *main.c* file. Note that instead of using the device register name, this project uses Pin\_Button\_\_INTCFG, provided by PSoC Creator (in the *cyfitter.h* file) for the interrupt configuration. You do not need to worry about the exact register name in the selected device. This helps to port the project to a different PSoC 4 device without changing anything in the code.

```

#define INTERRUPT_MASK 0x03
#define RISING_EDGE 0x01
#define FALLING_EDGE 0x02

int main()
{
    /* Variable to save temporary data */
    uint32 regVal = 0x00u;

    /* Flag to switch interrupt type */
    uint8 edgeFlag = 0x00u;

    for(;;)
    {
        /* Get value of port interrupt configuration register */
        regVal = CY_GET_REG32(Pin_Button__INTCFG);

        /* Clear the configuration bits for the Pin_Button.
        Pin_Button_SHIFT is multiplied by 2 as two bits of the interrupt
        configuration register sets the configuration for one pin */
        regVal &= ~(INTERRUPT_MASK << (Pin_Button_SHIFT * 2));

        if(edgeFlag)
        {
            /* Set P0[7] to GPIO interrupt rising-edge trigger.
            Pin_Button_SHIFT is multiplied by 2 as two bits of the
  
```

```

        interrupt configuration register sets the configuration for
        one pin */
        CY_SET_REG32(Pin_Button__INTCFG, regVal | (RISING_EDGE <<
(Pin_Button_SHIFT * 2)));
    }
    else
    {
        /* Set P0[7] to GPIO interrupt falling-edge trigger.
        Pin_Button_SHIFT is multiplied by 2 as two bits of the
        interrupt configuration register sets the configuration for
        one pin */
        CY_SET_REG32(Pin_Button__INTCFG, regVal | (FALLING_EDGE <<
(Pin_Button_SHIFT * 2)));
    }

    /* Toggle edgeFlag */
    edgeFlag ^= 0x01u;

    /* Wait for Interrupt */
    while(!(CY_GET_REG32(Pin_Button__INTSTAT) & (0x01u <<
Pin_Button_SHIFT))) {};

    /* Clear interrupt */
    CY_SET_REG32(Pin_Button__INTSTAT, (0x01u << Pin_Button_SHIFT));

    /* Toggle LED */
    Pin_LED_Write(~Pin_LED_Read());
}
}

```

6. Build the project and program the PSoC 4 device.

The LED toggles whenever the button is pressed or released. When the button is pressed, the falling-edge triggers the interrupt, and when it is released, the rising-edge triggers the interrupt.

The [PSoC 4 Architecture TRM](#) contains more information about the GPIO interrupt, including block diagrams and functional descriptions. Another good resource is the application note [AN90799 – PSoC 4 Interrupts](#).

## 10.9 Using Both Analog and Digital on a GPIO

This example demonstrates how to configure and use a pin for analog and digital functions. In this example, an output pin is controlled alternately by an IDAC and the firmware. When controlled by the firmware, the LED blinks. When controlled by the IDAC, the LED gradually brightens.

This type of multiplexing is useful when you need analog and digital functionality from a single pin. It can also reduce the number of GPIOs used in a design.

You can use a hardware connection instead of firmware to control the digital output. See the description at the end of this section for the required modifications to the project.

To configure the pin signal source, the HSIOM\_PORT\_SELx register is updated. Like the previous example, this project uses the register name as defined in the Pins Component for easy portability across the PSoC 4 device family.

Follow these steps to create the schematic and firmware:

1. Place an Analog Pin and a Current DAC in the schematic.
2. Assign the Pins Component to a physical pin (this example uses P0[2]).
3. Configure the pin with both **Analog** and **Digital Output** settings, as [Figure 56](#) shows.
4. Set the IDAC's **Polarity** as **Negative (Sink)**, as [Figure 57](#) shows. Connect the IDAC to the analog terminal, as [Figure 58](#) shows.
5. Build the project to create the necessary APIs.

Figure 56. LED Pin Configured as both Analog and Digital

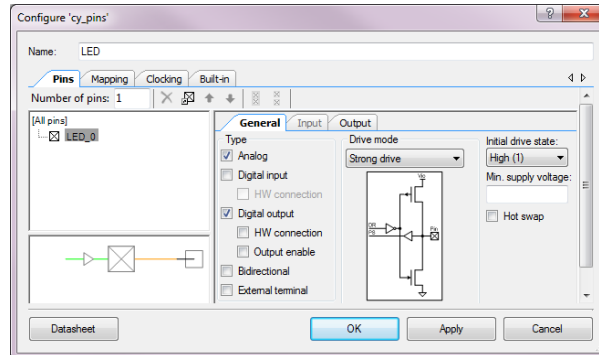


Figure 57. IDAC Setting

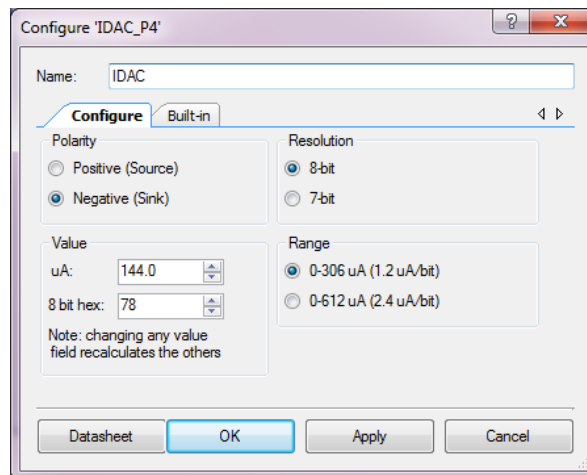
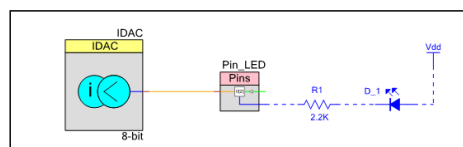


Figure 58. PSoC Creator Schematic of Analog and Digital Switching Scheme



6. Add the following code to the *main.c* file and build the project again. Program the device with the hex file generated. Note that in this code, the macros defined in the Pins Component and *Cyfitter.h* are used.

```

#define HSIOM_SW_GPIO 0x00
#define HSIOM_AMUX_BUS_A 0x06

int main()
{
    uint32 i = 0u;
    uint32 regVal = 0x00u;

    /* Disable Input Buffer */
    Pin_LED_INP_DIS |= (0x01u << Pin_LED_SHIFT);
    
```

```

/* Start IDAC */
IDAC_Start();

for(;;)
{
    /* Get the current value of HSIOM_PORT_SEL0 register */
    regVal = CY_GET_REG32(Pin_LED__0__HSIOM);
    regVal &= ~Pin_LED__0__HSIOM_MASK;

    /* Set LED Pin as GPIO controlled by firmware */
    regVal = CY_SET_REG32(Pin_LED__0__HSIOM, regVal | (HSIOM_SW_GPIO <<
Pin_LED__0__HSIOM_SHIFT));

    /* Set LED Pin to Strong Drive Mode */
    Pin_LED_SetDriveMode(Pin_LED_DM_STRONG);

    for(i= 0u; i < 5u; i++)
    {
        /* Toggle LED with 100-ms delay */
        Pin_LED_Write(0u);
        CyDelay(100u);
        Pin_LED_Write(1u);
        CyDelay(100u);
    }

    /* Get the current value of HSIOM_PORT_SEL0 register */
    regVal = CY_GET_REG32(Pin_LED__0__HSIOM);
    regVal &= ~Pin_LED__0__HSIOM_MASK;

    /* Connect LED Pin to AMUXBUS-A */
    CY_SET_REG32(Pin_LED__0__HSIOM, regVal | (HSIOM_AMUX_BUS_A <<
Pin_LED__0__HSIOM_SHIFT));

    /* Set LED Pin to High Impedance-Analog Drive Mode */
    Pin_LED_SetDriveMode(Pin_LED_DM_ALG_HIZ);

    for(i = 0u; i < 0x7fu; i++)
    {
        /* Adjust LED brightness */
        IDAC_SetValue(i);

        /* Delay 20 ms */
        CyDelay(20u);
    }
}
}

```

The result is an output that alternates control by the firmware and IDAC.

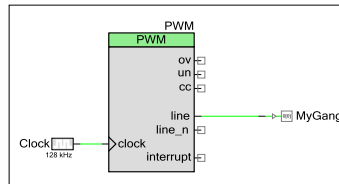
You can easily modify this project to use a hardware connection for the digital output instead of the firmware control. To do so, in step 3, enable the **HW connection** in the pin configuration window. You can then wire a digital resource to the pin. To select this digital resource as the pin output, set the pin as a DSI-controlled GPIO or a pin-specific digital resource connection, using the HSIOM\_PORT\_SEL register. See the [PSoC 4 Architecture TRM](#) for more details.

## 10.10 Gang Pins for More Drive/Sink Current

To increase the total source or sink capabilities of the circuit, GPIO pins can be ganged (shorted together). This example demonstrates driving a PWM signal with four GPIO pins. Note that the project is applicable only for PSoC 4200, PSoC 42xx\_BL, and PSoC 4200M parts.

1. Place and configure a PWM (TCPWM mode) and a Clock Component in the schematic.
2. Place a single Digital Output Pins Component.
3. Connect the Components as [Figure 59](#) shows.

Figure 59. PWM Driven to Single Pin



4. Open the pins configuration dialog and set the number of pins accordingly, as [Figure 60](#) shows. This example uses four GPIO pins. Set the **Output Mode** to **Single-Sync** and **Out Clock** to **External**, as [Figure 61](#) and [Figure 62](#) show.

**Note:** Synchronize the output to avoid different output signal delays for the different pins.

Figure 60. Configure Multiple Pins in the Component

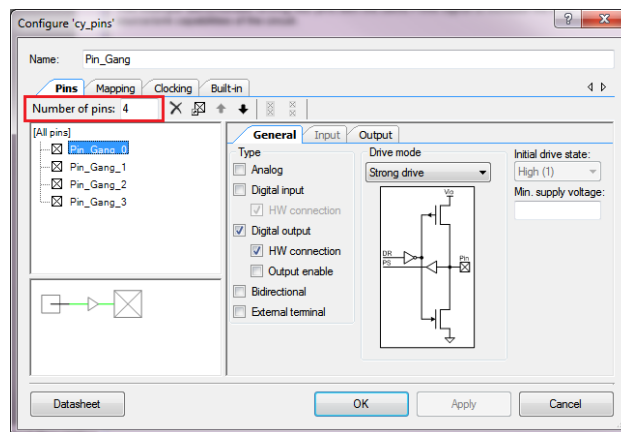


Figure 61. Output Mode Setting

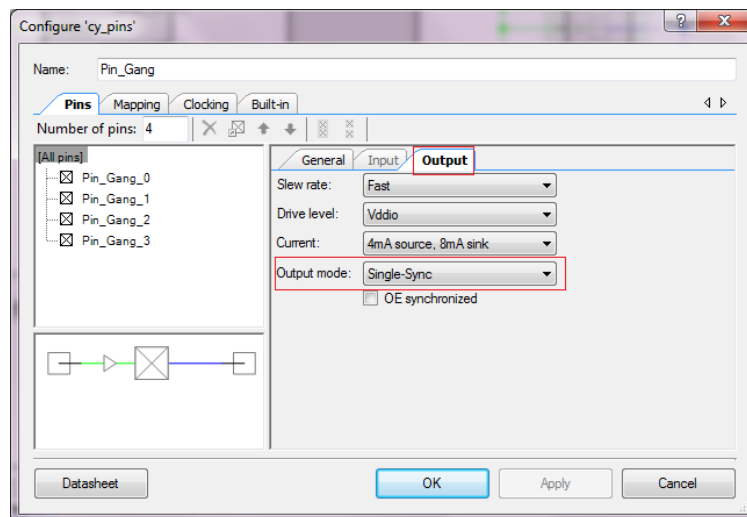
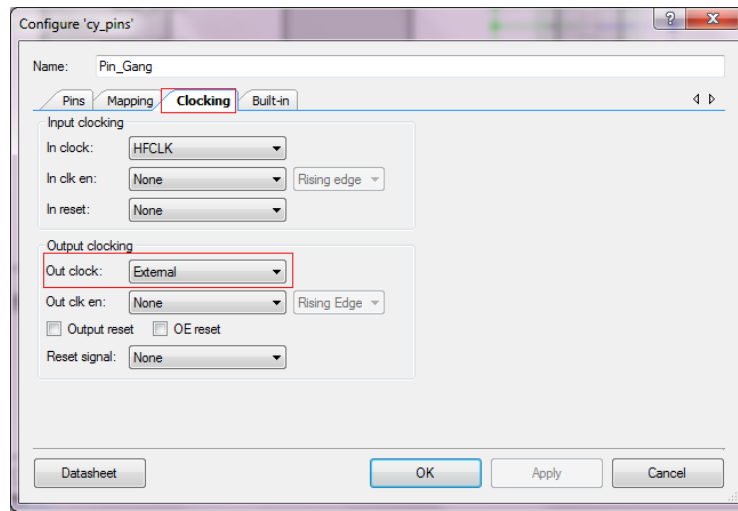
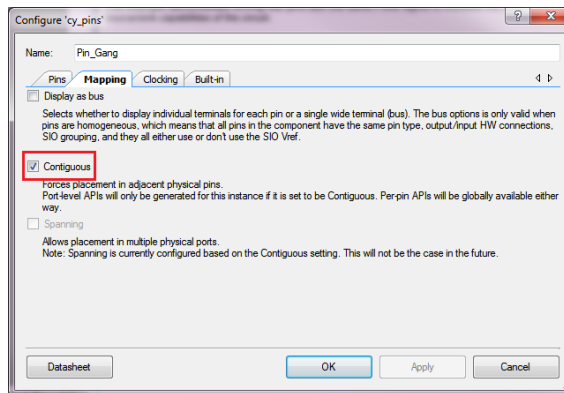


Figure 62. Out Clock Setting



- (Optional) Set the pin mapping to **Contiguous** for easier PCB routing, as Figure 63 shows.

Figure 63. Enable Contiguous Mapping

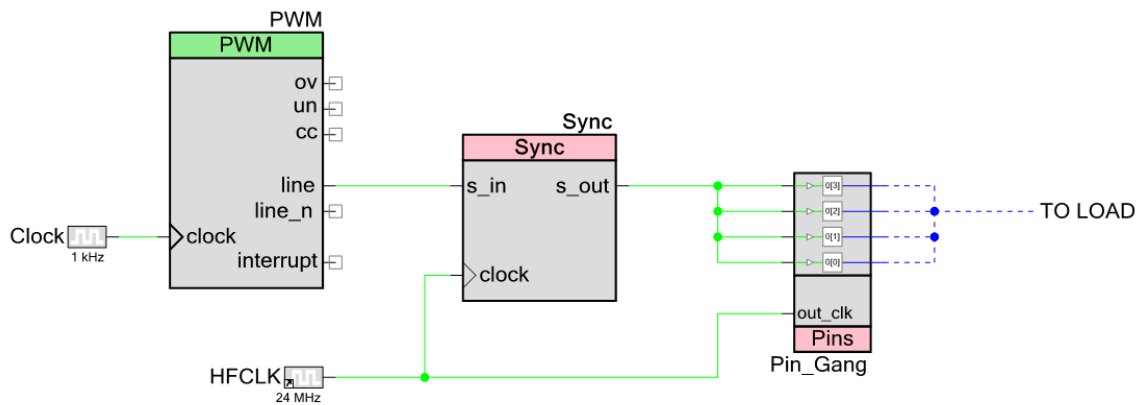


- Assign the Pins Component to physical pins.
- Place a Sync Component and connect the signal source (PWM, in this example) to each of the pin terminals via the Sync Component. Place another Clock Component and set its source to High Frequency Clock (HFCLK). Connect the out\_clk terminal of the Pin Component and the Clock terminal of the Sync Component to the HFCLK.

It is important to select a high-frequency synchronization clock to reduce the difference in pin signal delays. The Sync Component is required to synchronize the signal crossing from one clock domain to another. In this case, the PWM output is going to cross from Clock (1 kHz) domain to HFCLK.



Figure 64. PWM Driving Four Pins



- Build the project and program the PSoC 4 device.

The output of the PWM is driven on all four GPIOs. The pins can be shorted externally on the PCB and connected to the external circuit as needed.

## 10.11 Control Register Handling in Deep-Sleep

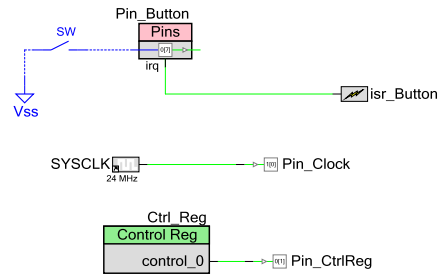
This example demonstrates freezing of GPIO pins to avoid glitches at the output while using the low-power modes. As an example, consider a Control Register driving a pin. When the device enters the Deep-Sleep mode, all I/Os are frozen without any user intervention. When the device wakes up, all the I/Os are automatically restored to their original configuration. However, the Control Register loses its data in the Deep-Sleep mode. It needs to be restored before the I/Os are unfrozen. Otherwise, there is a glitch at the output. PSoC 4 provides alternate control for the user to freeze and unfreeze the GPIOs in the form of `CySysPmFreezeIo()` and `CySysPmUnfreezeIo()` APIs. Follow these steps to create the PSoC Creator project. Note that this project is applicable only for PSoC 4200, PSoC 42xx\_BL, and PSoC 4200M.

- Place one Digital Input Pin, two Digital Output Pins, a Clock, a Control Register, and an Interrupt Component in the schematic.
- Configure the Components as Table 9 lists. Connect the Components as Figure 65 shows.

Table 9. Component Configurations

Component	Name	Configuration
Digital Input Pin	Pin_Button	Drive Mode: Resistive Pull-Up Interrupt: Rising-Edge
Digital Output Pin	Pin_Clock	Default Configuration
Digital Output Pin	Pin_CtrlReg	Default Configuration
Clock	SYSCLK	Source: SYSCLK
Interrupt	isr_Button	Default Configuration
Control Register	Ctrl_Reg	Output: 1 Initial value: 1

Figure 65. Avoiding Glitch While Exiting Deep-Sleep



3. Add the following code to the *main.c* file.

```

/* Set FREEZE_IO to 0x01 to avoid glitch by enabling the GPIO freeze */
/* else set it to 0 */
#define FREEZE_IO 0x01

/* The flag to enter ISR */
uint8 isrFlag = 0u;
CY_ISR(ISR_Handle)
{
    /* Set the flag */
    isrFlag = 1u;

    /* Clear pin interrupt */
    Pin_Button_ClearInterrupt();
}

int main()
{
    /* This variable is used as backup for Control register value */
    uint8 ctrlRegVal = 0u;

    /* Clear the flag */
    isrFlag = 0u;

    /* Start the ISR */
    isr_Button_StartEx(ISR_Handle);

    CyGlobalIntEnable;

    /* Set Control register output as high */
    Ctrl_Reg_Write(1u);

    for(;;)
    {
        /* If freeze flag is set */
        if(0u != isrFlag)
        {
            /* Clear isr flag set in GPIO Interrupt Handler */
            isrFlag = 0u;

            /* Rewrite the value */
            Ctrl_Reg_Write(ctrlRegVal);

            #if(FREEZE_IO)
                /* Unfreeze I/O */
                CySysPmUnfreezeIo();
            #endif
        }
    }
}

```

```

        #endif
    }

    /* Delay 200us */
    CyDelayUs(200u);

    /* Store the value of Control register */
    ctrlRegVal = Ctrl_Reg_Read();

    #if(FREEZE_IO)
        /* Freeze I/O */
        CySysPmFreezeIo();
    #endif

    /* Enter Deep-Sleep mode */
    CySysPmDeepSleep();
}

```

To disable the freeze option, set FREEZE\_IO to 0. Build and program the device. When the button is pressed and released, the glitch can be seen on Pin\_CtrlReg as [Figure 66](#) shows. To enable the freeze option, set FREEZE\_IO to 1. Build and program the device. In this case, I/O is frozen and no glitch is observed, as [Figure 67](#) shows.

Figure 66. Output Signal Waveform (No Freeze I/O)

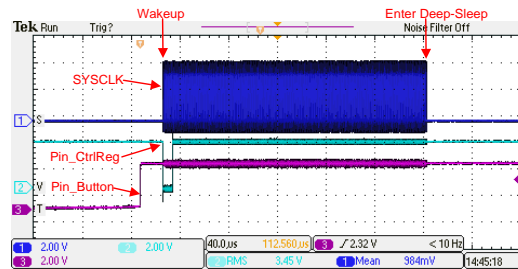
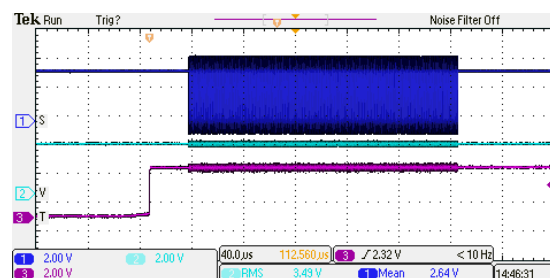


Figure 67. Output Signal Waveform (Freeze I/O)



## 11 Related Application Notes

[AN79953 – Getting Started with PSoC 4](#)

[AN86233 – PSoC 4 Low-Power Modes and Power Reduction Techniques](#)

[AN60024 – Switch Debouncer and Glitch Filter with PSoC 3, PSoC 4, and PSoC 5LP](#)

[AN72382 – Using PSoC 3 and PSoC 5LP GPIO Pins](#)

[AN90799 – PSoC 4 Interrupts](#)

[AN2094 – PSoC 1 Getting Started with GPIO](#)

[AN89610 - PSoC® 4 and PSoC 5LP ARM Cortex Code Optimization](#)

## 12 Summary

This application note explains the basic use cases of the GPIO modes in PSoC 4 using the Pin Component in PSoC Creator.

## 13 About the Authors

Name: Charles Cheng  
Title: Application Engineer  
Background: Charles is an application engineer in the Cypress Semiconductor Programmable Systems Division focused on PSoC applications.

Name: Rajiv Badiger  
Title: Application Engineer Staff  
Background: Rajiv is an application engineer in the Cypress Semiconductor Programmable Systems Division focused on PSoC applications.

Contact: [rjvb@cypress.com](mailto:rjvb@cypress.com)

## A Appendix A: PSoC 4 GPIO Compared to PSoC 1, PSoC 3, and PSoC 5LP GPIO

The PSoC 4 GPIO is different from that of PSoC 1, PSoC 3, and PSoC 5LP; see [Table 10](#) for details.

Table 10. PSoC 4 GPIO versus PSoC 1, PSoC 3, and PSoC 5LP GPIO

GPIO Features	PSoC 1	PSoC 3	PSoC 4	PSoC 5LP
CapSense	√	√	√	√
LCD segment drive	√	√	√*	√
Eight drive modes	√	√	√	√
POR state configuration	×	√	×	√
Separate port DR and PS	×	√	√	√
Input/output synchronization	×	Bus_clk	HFCLK, External*	Bus_clk

\* Not available in PSoC 4000

## B Appendix B: PSoC 4 Development Boards

You can test the PSoC Creator projects provided with this application note on the following Cypress development boards.

Device Family	Development Board
PSoC 4000	<a href="#">CY8CKIT-040 PSoC 4000 Pioneer Development Kit</a>
PSoC 4200 / PSoC 4100	<a href="#">CY8CKIT-042 PSoC® 4 Pioneer Kit</a> <a href="#">PSoC 4 CY8CKIT-049 4xxx Prototyping Kits</a>
PSoC 42x7_BL	<a href="#">CY8CKIT-042-BLE Bluetooth® Low Energy (BLE) Pioneer Kit</a>
PSoC 4200M	<a href="#">CY8CKIT-044 PSoC® 4 M-Series Pioneer Kit</a>

## Document History

Document Title: AN86439 - PSoC® 4 – Using GPIO Pins

Document Number: 001-86439

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4288100	CLSC	03/27/2014	New application note
*A	4381956	RJVB	05/16/2014	Updated for PSoC 4000
*B	4739860	RJVB	07/02/2015	Updated for PSoC 4 BLE and PSoC 4 M-Series Updated component customizer screenshots Added information on latency in GPIO update Added example projects Added Appendix B – PSoC 4 Development Boards Updated information on GPIO architecture

## 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	<a href="http://cypress.com/go/automotive">cypress.com/go/automotive</a>
Clocks & Buffers	<a href="http://cypress.com/go/clocks">cypress.com/go/clocks</a>
Interface	<a href="http://cypress.com/go/interface">cypress.com/go/interface</a>
Lighting & Power Control	<a href="http://cypress.com/go/powerpsoc">cypress.com/go/powerpsoc</a>
Memory	<a href="http://cypress.com/go/memory">cypress.com/go/memory</a>
PSoC	<a href="http://cypress.com/go/psoc">cypress.com/go/psoc</a>
Touch Sensing	<a href="http://cypress.com/go/touch">cypress.com/go/touch</a>
USB Controllers	<a href="http://cypress.com/go/usb">cypress.com/go/usb</a>
Wireless/RF	<a href="http://cypress.com/go/wireless">cypress.com/go/wireless</a>

### PSoC<sup>®</sup> Solutions

[psoc.cypress.com/solutions](http://psoc.cypress.com/solutions)

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#)

### Cypress Developer Community

[Community](#) | [Forums](#) | [Blogs](#) | [Video](#) | [Training](#)

### Technical Support

[cypress.com/go/support](http://cypress.com/go/support)

CapSense and PSoC are registered trademarks and PSoC Creator is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.



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

© Cypress Semiconductor Corporation, 2014-2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.