PSoC™ 4 MCU - Using GPIO pins

About this document

Scope and purpose

AN86439 explains how to effectively use PSoC™ 4 MCU GPIO pins, with various use case examples to demonstrate their features. Major topics in this application note include GPIO basics, configuration options, mixed-signal use, interrupts, and low-power behavior.

Intended audience

This document is intended for anyone who uses the PSoC™ 4 MCU GPIO pins.

Table of contents

1 Introduction

PSoC™ has a flexible general-purpose I/O (GPIO) architecture that provides more features than traditional MCUs. PSoC™ GPIOs are controlled not only by configuring the registers in 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 shows techniques for using them effectively for different functions.

This application note assumes that you are familiar with PSoC™ Creator or ModusToolbox™ and the PSoC™ 4 architecture. If you are new to PSoC™ 4, read AN79953 – Getting started with PSoC™ 4 MCU. If you are new to PSoC™ Creator, visit the PSoC™ Creator. If you are new to ModusToolbox™, visit the ModusToolbox™ software. For information on device packages or GPIO specifications, see the PSoC™ 4 datasheet. If you are already familiar with the device and development environments, you can jump to the GPIO tips and tricks in PSoC™ Creator or ModusToolbox™ software GPIO tips and tricks section.

This application note describes how to use the PSoC™ peripheral driver library (PDL) and ModusToolbox™ to develop with PSoC™ 4 devices. This is currently supported only when using PSoC™ 4 S-series devices.

Note: References to 'PSoC™' or 'device' henceforth refer to PSoC™ 4, unless specified otherwise.

2 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).

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

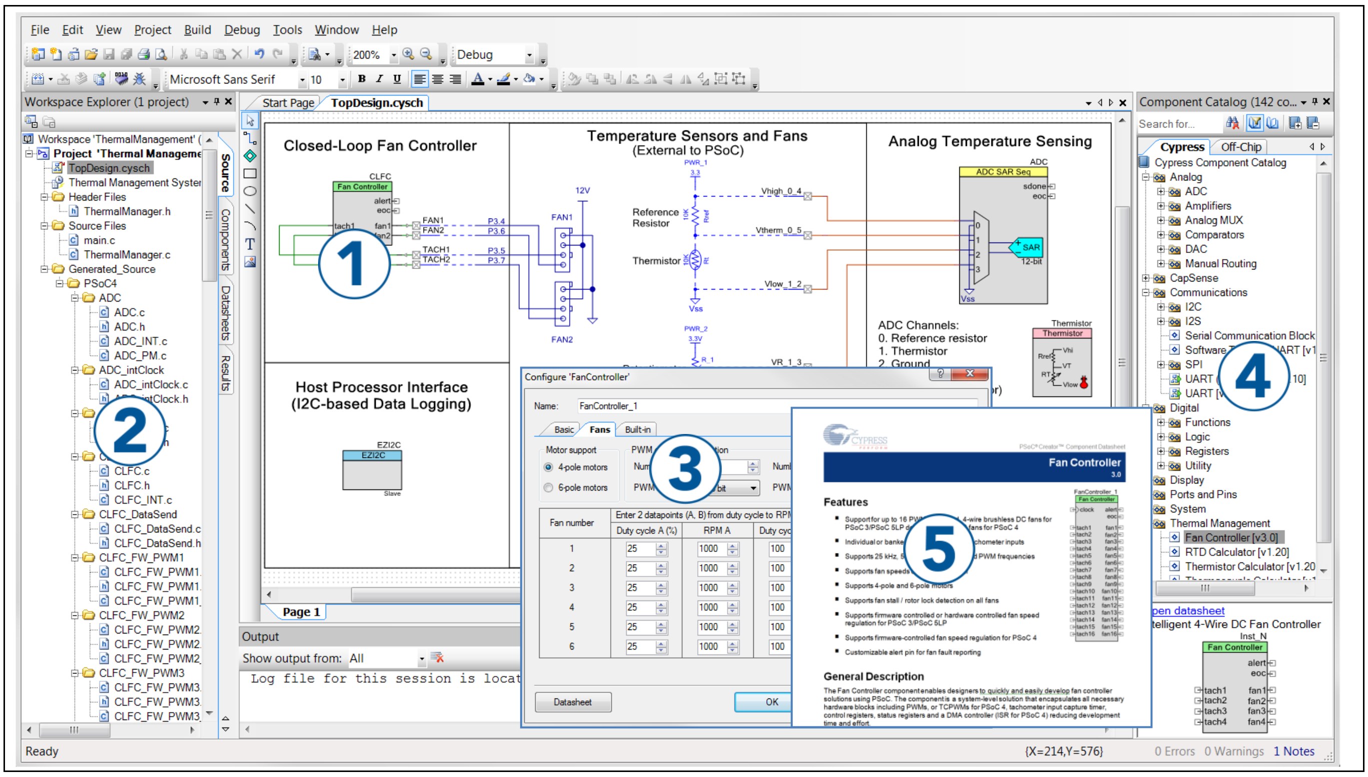

1
2
3
4
5
Closed-Loop Fan Controller
Temperature Sensors and Fans (External to PSoC)
Analog Temperature Sensing
Host Processor Interface (I2C-based Data Logging)
Workspace Explorer (1 project)
Component Catalog
Configure 'FanController'
Fan Controller
Output
Ready

Figure 1 **PSoC™ Creator features**

2.1 PSoC™ Creator code examples

PSoC™ Creator includes a large number of code example projects. These projects are available from the PSoC™ Creator start page, as shown in Figure 2. PSoC™ Creator code examples can also be downloaded from infineon.com.

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. The code examples and datasheets are included, as shown in Figure 3.

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

- Filter for examples based on 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.

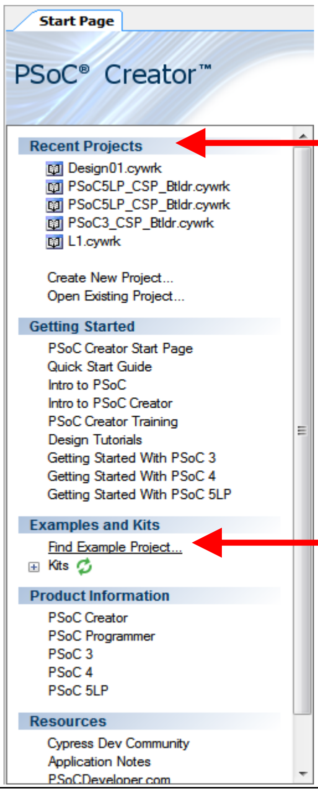
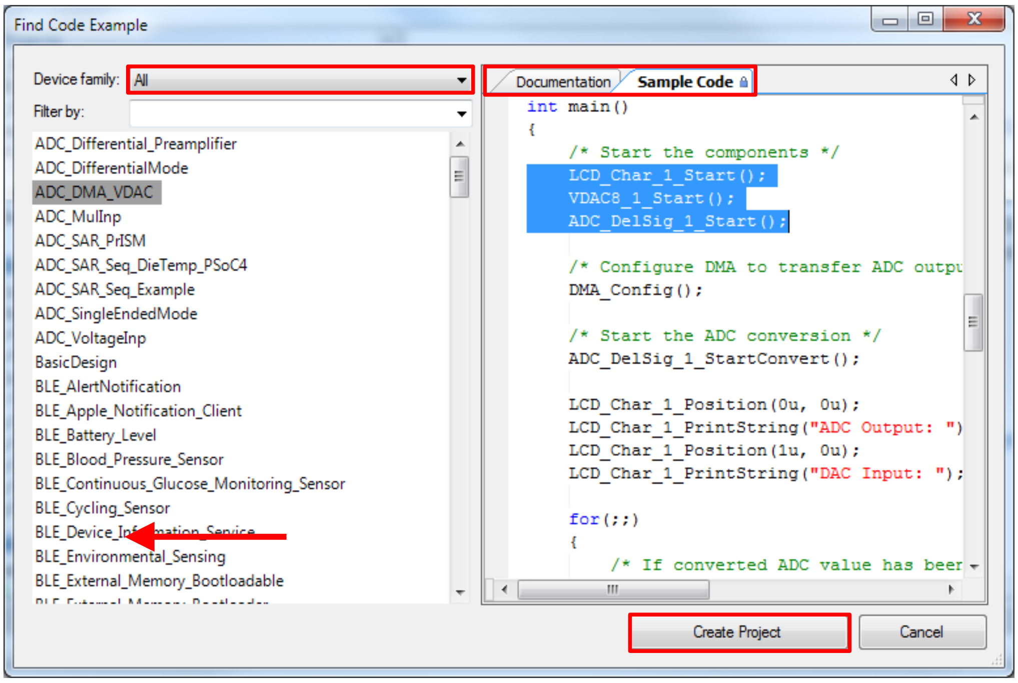


Figure 2 **Code examples in PSoC™ Creator**



```
int main()
{
    /* Start the components */
    LCD_Char_1_Start();
    VDAC8_1_Start();
    ADC_DelSig_1_Start();

    /* Configure DMA to transfer ADC outpu
    DMA_Config();

    /* Start the ADC conversion */
    ADC_DelSig_1_StartConvert();

    LCD_Char_1_Position(0u, 0u);
    LCD_Char_1_PrintString("ADC Output: ")
    LCD_Char_1_Position(1u, 0u);
    LCD_Char_1_PrintString("DAC Input: ");

    for(;;)
    {
        /* If converted ADC value has beer
```

Figure 3 **Code example projects with sample code**

2.1.1 PSoC™ Creator help

Visit the PSoC™ Creator 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.
- **System reference guide**: Choose **Help** > **System Reference Guides**. 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**.

2.2 ModusToolbox™ software

ModusToolbox™ is a set of multi-platform development tools and a comprehensive suite of GitHub-hosted firmware libraries. Together, they enable an immersive development experience for customers creating converged MCU and wireless systems.

The firmware libraries comprise easily customizable board support packages (BSP) for Infineon PSoC™ 6 MCU, PSoC™ 4, and Bluetooth® SoC (20xxx) kits and a comprehensive set of middleware libraries enabling industry-leading features:

- CAPSENSE™
- Bluetooth® Low Energy and Mesh
- Lowest-power, most reliable Wi-Fi on the market
- Impressive set of thoroughly tested and helpful code example applications

Visit the ModusToolbox™ software to download the latest version of ModusToolbox™. The following are helpful items for getting started with ModusToolbox™:

- Quick start guide: This is a short step-by-step guide specifically for using the Eclipse-based IDE to create and build applications for ModusToolbox™.
- ModusToolbox™ tools package user guide: This guide focuses on the Eclipse IDE, covering more details about the IDE and software features.
- Documentation: Refer to Quick Panel section in ModusToolbox™ IDE user guide.

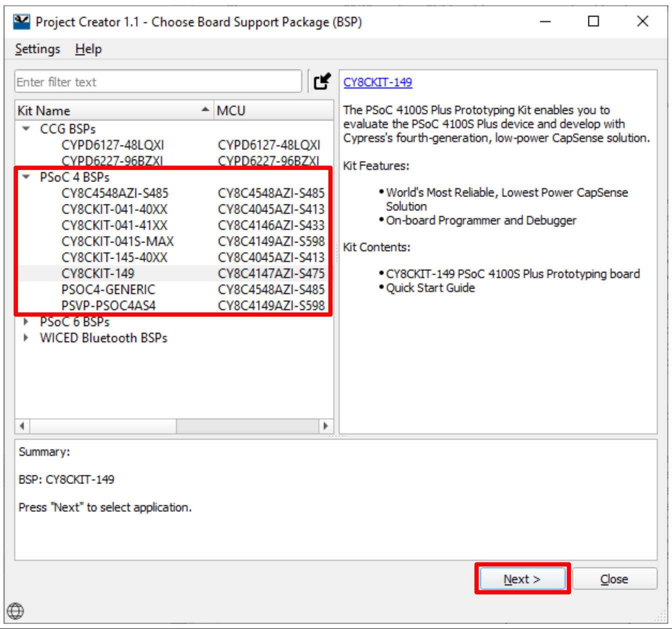
Note: ModusToolbox™ is compatible with KitProg3 and MiniProg4 (CY8CKIT-005-A) programming devices.

2.2.1 ModusToolbox™ code examples

ModusToolbox™ includes a growing number of code example projects. These code example projects are available in the new application wizard in ModusToolbox™ or Infineon GitHub.

Example projects can speed up your design process by starting you off with a complete design instead of a blank page.

In the New Application Wizard, as Figure 4 shows, you can choose your board support package (BSP). The BSP corresponds to the specific kit that is being used. The code examples can be viewed when a BSP is chosen, as shown in Figure 5.



Kit Name	MCU
CCG BSPs	
CYPD6127-48LQXI	CYPD6127-48LQXI
CYPD6227-96BZXI	CYPD6227-96BZXI
PSoC 4 BSPs	
CY8C4548AZI-S485	CY8C4548AZI-S485
CY8CKIT-041-40XX	CY8C4045AZI-S413
CY8CKIT-041-41XX	CY8C4146AZI-S433
CY8CKIT-041S-MAX	CY8C4149AZI-S598
CY8CKIT-145-40XX	CY8C4045AZI-S413
CY8CKIT-149	CY8C4147AZI-S475
PSOC4-GENERIC	CY8C4548AZI-S485
PSVP-PSOC4AS4	CY8C4149AZI-S598
PSoC 6 BSPs	
WICED Bluetooth BSPs	

CY8CKIT-149

The PSoC 4100S Plus Prototyping Kit enables you to evaluate the PSoC 4100S Plus device and develop with Cypress's fourth-generation, low-power CapSense solution.

Kit Features:

- World's Most Reliable, Lowest Power CapSense Solution
- On-board Programmer and Debugger

Kit Contents:

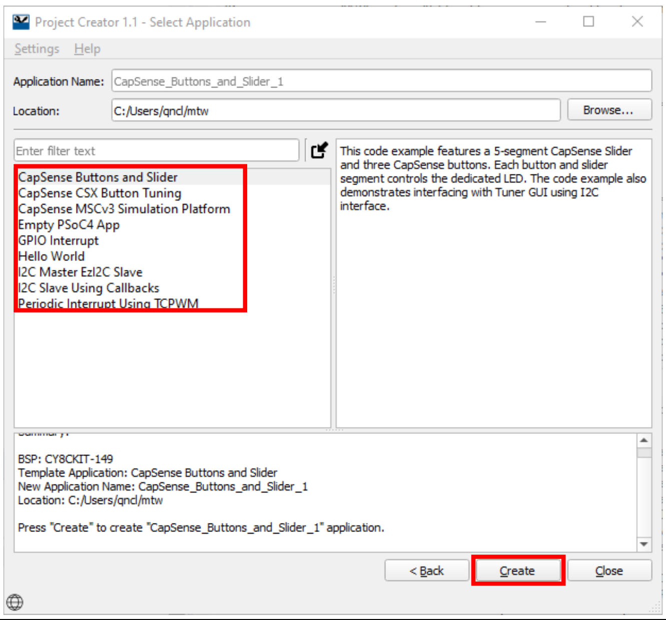
- CY8CKIT-149 PSoC 4100S Plus Prototyping board
- Quick Start Guide

Summary:

BSP: CY8CKIT-149

Press "Next" to select application.

Figure 4 New application wizard



Application Name: CapSense_Buttons_and_Slider_1

Location: C:/Users/qncl/mtw

- CapSense Buttons and Slider
- CapSense CSX Button Tuning
- CapSense MSCv3 Simulation Platform
- Empty PSoC4 App
- GPIO Interrupt
- Hello World
- I2C Master EzI2C Slave
- I2C Slave Using Callbacks
- Periodic Interrupt Using TCPWM

This code example features a 5-segment CapSense Slider and three CapSense buttons. Each button and slider segment controls the dedicated LED. The code example also demonstrates interfacing with Tuner GUI using I2C interface.

BSP: CY8CKIT-149
Template Application: CapSense Buttons and Slider
New Application Name: CapSense_Buttons_and_Slider_1
Location: C:/Users/qncl/mtw

Press "Create" to create "CapSense_Buttons_and_Slider_1" application.

Figure 5 Code examples

2.3 Technical support

For further assistance and clarification, create a support request on the Infineon technical support page.

- Use the support resources Self-help, for quick assistance

3 GPIO pin basics

The PSoC™ GPIO pins offer the following features:

- Analog and digital input and output capability
- LCD segment drive support (not available in PSoC™ 4000 and PSoC™ 4200DS)
- CAPSENSE™ support
- Interrupt on level, rising-edge, falling-edge, or both edges
- Slew-rate control
- Input threshold select (CMOS / LVTTL)
- Overvoltage-tolerant pins (available only in PSoC™ 4 Bluetooth® LE, PSoC™ 4 M-Series, and PSoC™ 4 L-Series) with hot-swap capability

The GPIO functionality depends on the peripherals available in the PSoC™ 4 device. For a side-by-side comparison of the features available in different PSoC™ 4 families, see Table1 in AN79953 – Getting started with PSoC™ 4 MCU.

3.1 Physical structure of GPIO pins

Figure 6 shows the pin connections with the resources in the PSoC™ device.

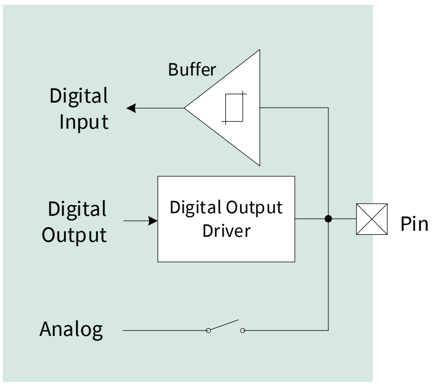


Figure 6 Simplified GPIO block diagram

A detailed block diagram of the GPIO structure is available in the “I/O System” chapter of the PSoC™ 4 architecture reference manual. Each pin can act as an input or an output to the CPU and the digital peripheral such as the Timer, PWM, or I^{2}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, only-digital output, only-analog, or even combinations of these three. For example, if you enable both digital output and input, it provides a digital bidirectional pin. The input buffer provides high impedance to the external input. It is configurable to CMOS, LVTTL.

For the input threshold values, see the device datasheet.

The digital output driver supports different drive modes and slew-rate control (see Figure 7).

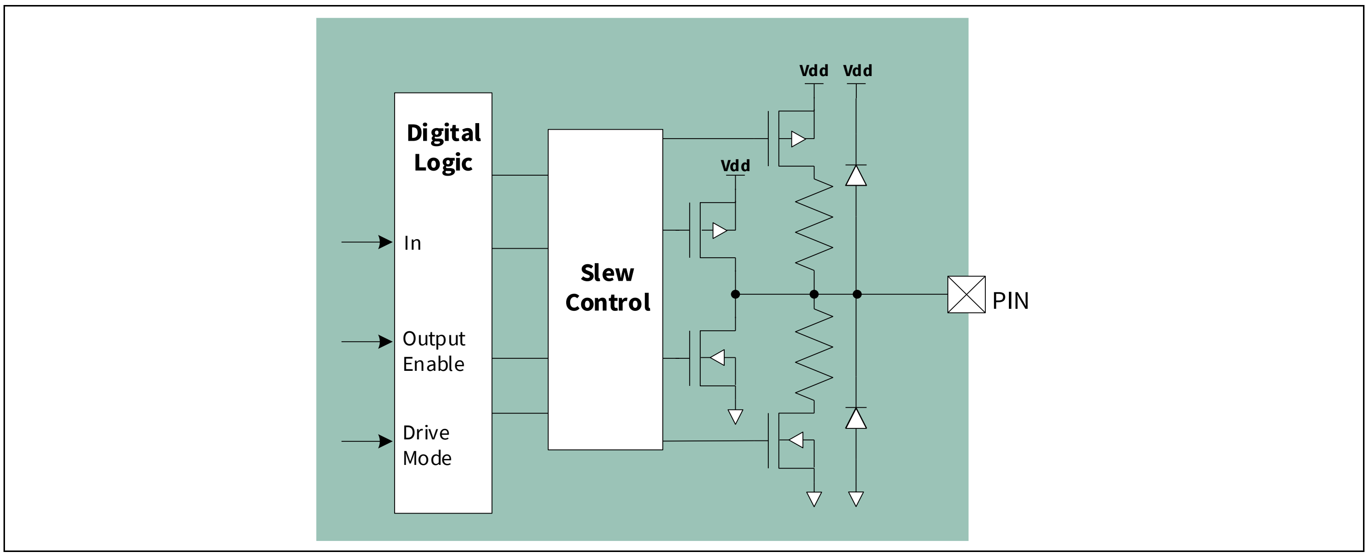


Figure 7 Digital output driver

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 7 supports eight drive modes, as listed in Table 1.

Table 1 Drive modes and applications

#	Drive mode	Application examples
1	High-impedance (High-Z) analog	Analog input/output
2	High-impedance (HIGH-Z) digital	Digital input
3	Resistive pull-up (~5 kΩ)	Interface to open-drain LOW input, such as the 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^2C 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 states
8	Resistive pull-up and resistive pull-down (~5 kΩ)	Adds a series resistor in both HIGH and LOW states

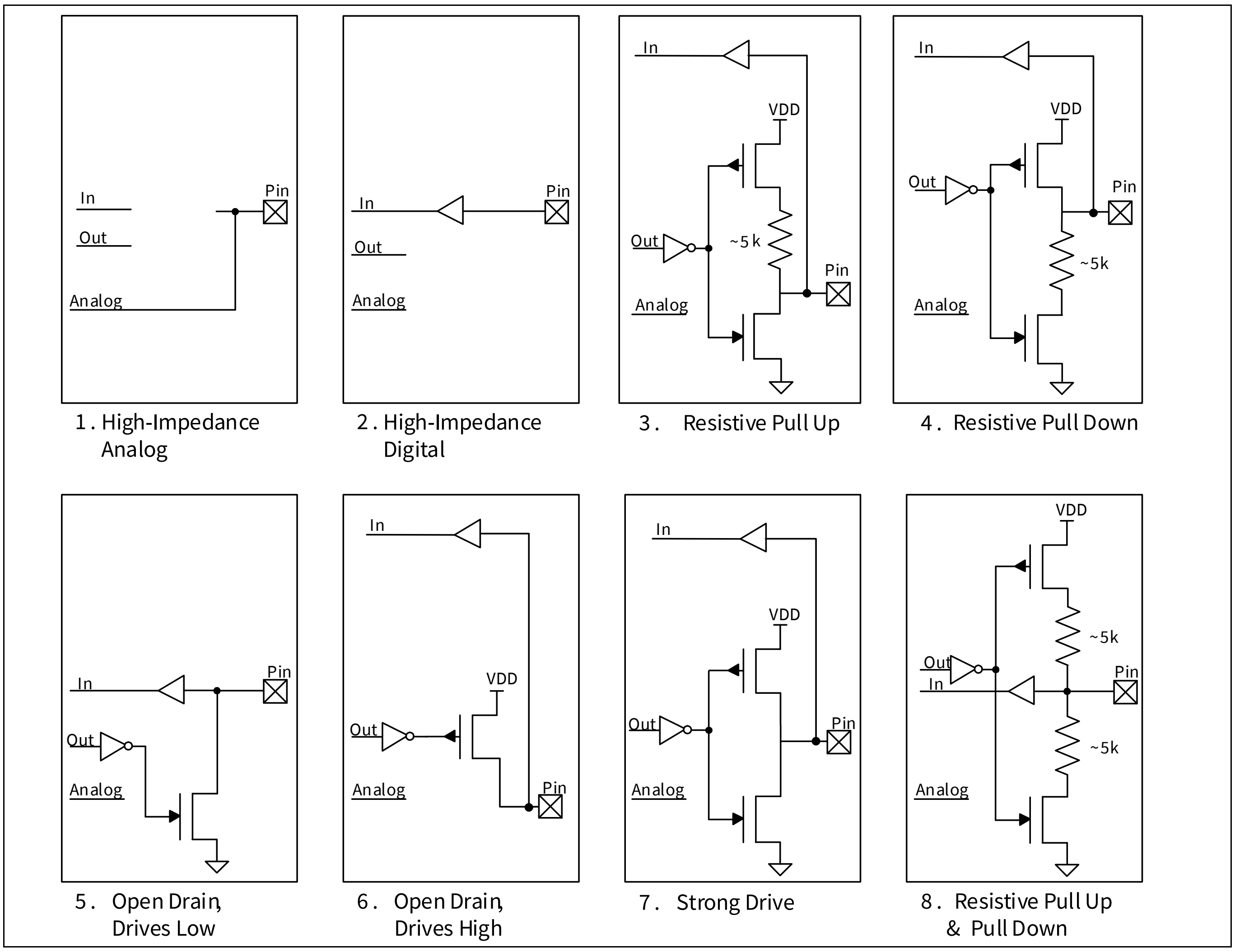

In
Out
Analog
Pin
1. High-Impedance Analog
2. High-Impedance Digital
VDD
~5 k
3. Resistive Pull Up
~5k
4. Resistive Pull Down
5. Open Drain, Drives Low
6. Open Drain, Drives High
7. Strong Drive
8. Resistive Pull Up & Pull Down

Figure 8 **Drive modes**

Note: *The resistor values for pull-up and pull-down drive modes, shown in Figure 8, are approximate values; see the device datasheet for resistor value specifications. Use an external resistor if a higher accuracy is required. In this case, the pin must be configured as Open Drain Drive High or Open Drain Drive Low.*

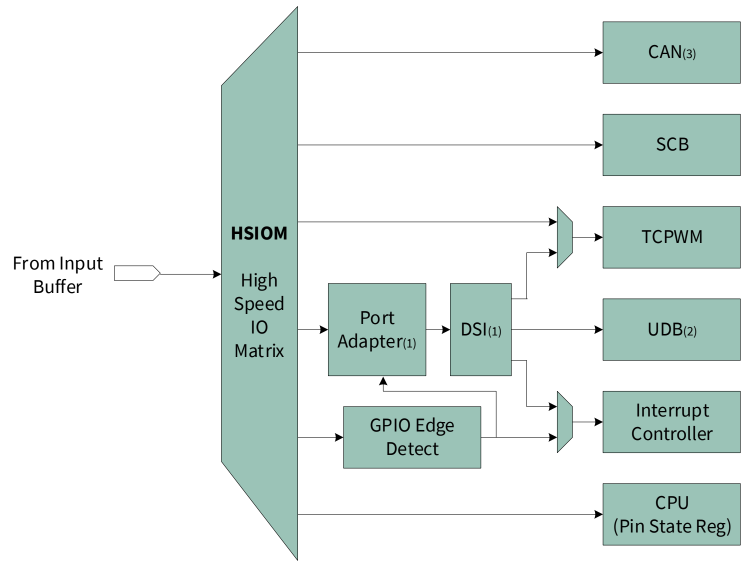
Note: *At all times, avoid the device VDD getting powered from an external voltage at the pin through ESD clamp diodes. This can happen if the PSoC™ 4 device is not powered and an external voltage is applied at the GPIO or when an external voltage at the GPIO is greater than the device VDD. This is,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

3.2 Pin routing

3.2.1 Digital routing

A pin can be routed to different digital peripherals, such as universal digital block (UDB), Serial Communication Block (SCB), Timer/Counter/Pulse-Width Modulator (TCPWM) block, LCD driver, CAN block, and interrupt controller; as well as the data register which is read/written by the CPU. Figure 9 shows the routing for an input pin and Figure 10 shows the routing for an output pin. As shown in these figures, 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™, 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.

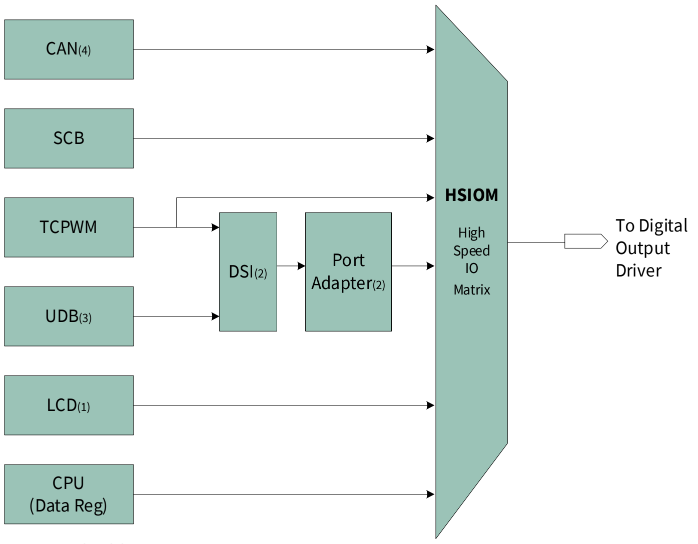


1. Not applicable to PSoC™ 4000, PSoC™ 4000S, PSoC™ 4100S, 4100S Plus, PSoC™ 4100PS
2. Not applicable to PSoC™ 4000, PSoC™ 4000S, PSoC™ 4100PS, PSoC™ 4100, PSoC™ 4100S, PSoC™ 4100S Plus, PSoC™ 41xx-BL, or PSoC™ 4100M
3. Only available in PSoC™ 4200M , PSoC™ 4200L, and 4100S Plus devices

Figure 9 Digital pin input path

SCB (I^{2}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™ parts (except PSoC™ 4000 and PSoC™ 4200DS), with any I/O acting as a segment or a common driver for the LCD.

The GPIO Edge Detect block enables pin interrupts on rising-edge, falling-edge, and both edges. See the GPIO interrupt section for details.



1. Not applicable to PSoC™ 4000
2. Not applicable to PSoC™ 4000, PSoC™ 4000S, PSoC™ 4100S, PSoC™ 4100S Plus, PSoC™ 4100PS
3. Not applicable to PSoC™ 4000, PSoC™ 4000S, PSoC™ 4100PS, PSoC™ 4100, PSoC™ 4100S, PSoC™ 4100S Plus, PSoC™ 41xx-BL, or PSoC™ 4100M
4. Only available in PSoC™ 4200M, PSoC™ 4200L, and PSoC™ 4100S Plus.

Figure 10 Digital pin output path

Note: *PSoC™ 4 has multiple ports with a maximum of 8 pins per port. For PSoC™ 4200L devices, Ports 7, 8, and 9 pins do not have the port adapter; for other devices, 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), Low-Power Comparator (applicable only to PSoC™ 4100, PSoC™ 4100PS, and PSoC™ 4200), and Continuous Time Block (applicable only to PSoC™ 4100PS)
- 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 CAPSESNE™ pins
- Interrupt generation

Note: *Pins of the PSoC™ 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.*

3.2.2 Analog routing

GPIO pins configured in the High-impedance analog (HIGH-Z) mode are connected to the analog resources by direct connections or through the analog switches and the analog mux (AMUX) bus, as shown in Figure 11 through Figure 16 .

The following are the key highlights of the analog routing in the PSoC™ 4000 parts shown in Figure 11:

- 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.*

The following are the key highlights of analog routing in other PSoC™ 4 parts — see Figure 13 through Figure 17.

- 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 by using the DSI signal. Note that in the 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 13 through Figure 17 to know about the pins.
- CAPSENSE™ IDAC0 is connected to AMUXBUS_A; 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 14 through Figure 17. This is useful if the AMUX buses are required for non-CAPSENSE™ applications such as opamp/comparator input and output routing along with CAPSENSE™ in the system.
- The SAR sequencer connects the SAR ADC input to:
  - Port 2 in PSoC™ 4100, PSoC™ 4100S, PSoC™ 4100S Plus, PSoC™ 4200, PSoC™ 4100M, PSoC™ 4200M, and PSoC™ 4200L
  - Port 3 in PSoC™ 41xx-BL, PSoC™ 42xx-BL, and PSoC™ 41xxPS
  - CTBm, CTB outputs
  - Temperature sensor output

Multiplexing is done by controlling the switches shown in red in Figure 11 through Figure 15. 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-numbered pin with the negative input as the adjacent odd-numbered 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. This is shown using rings* ◯ *in analog routing diagrams.*

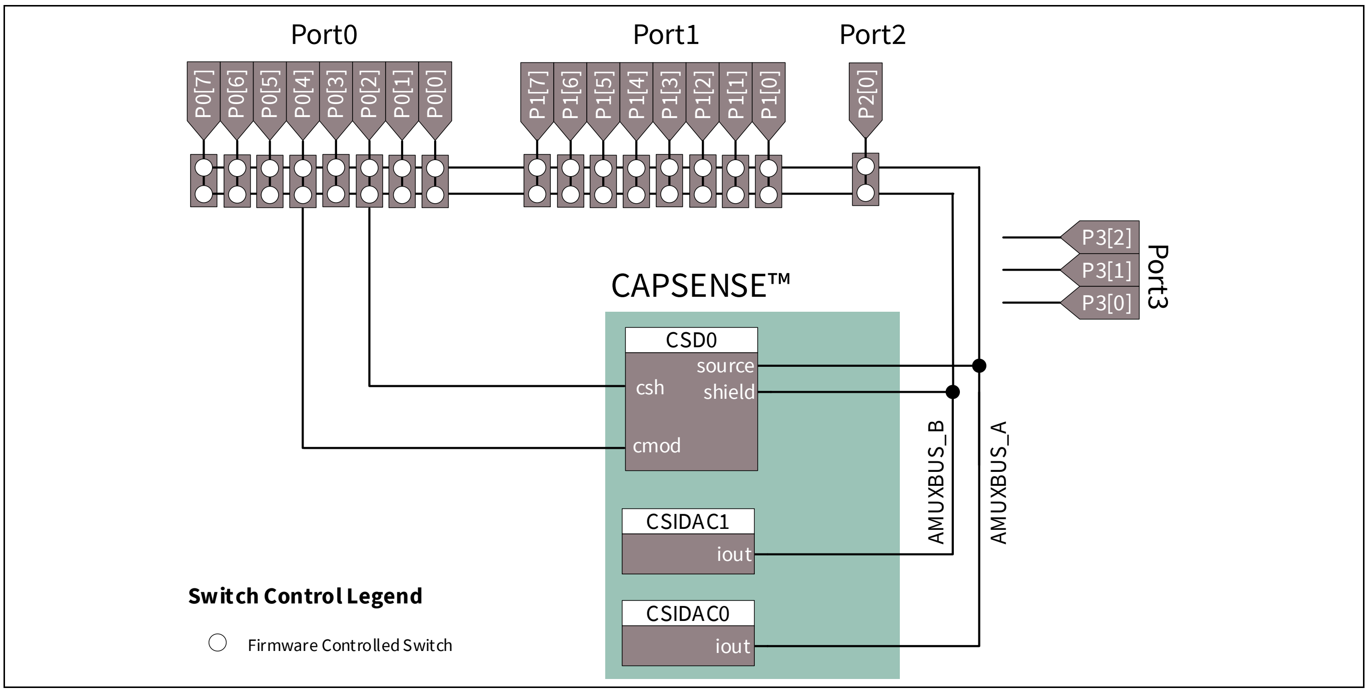


Figure 11 **PSoC™ 4000 analog routing diagram**

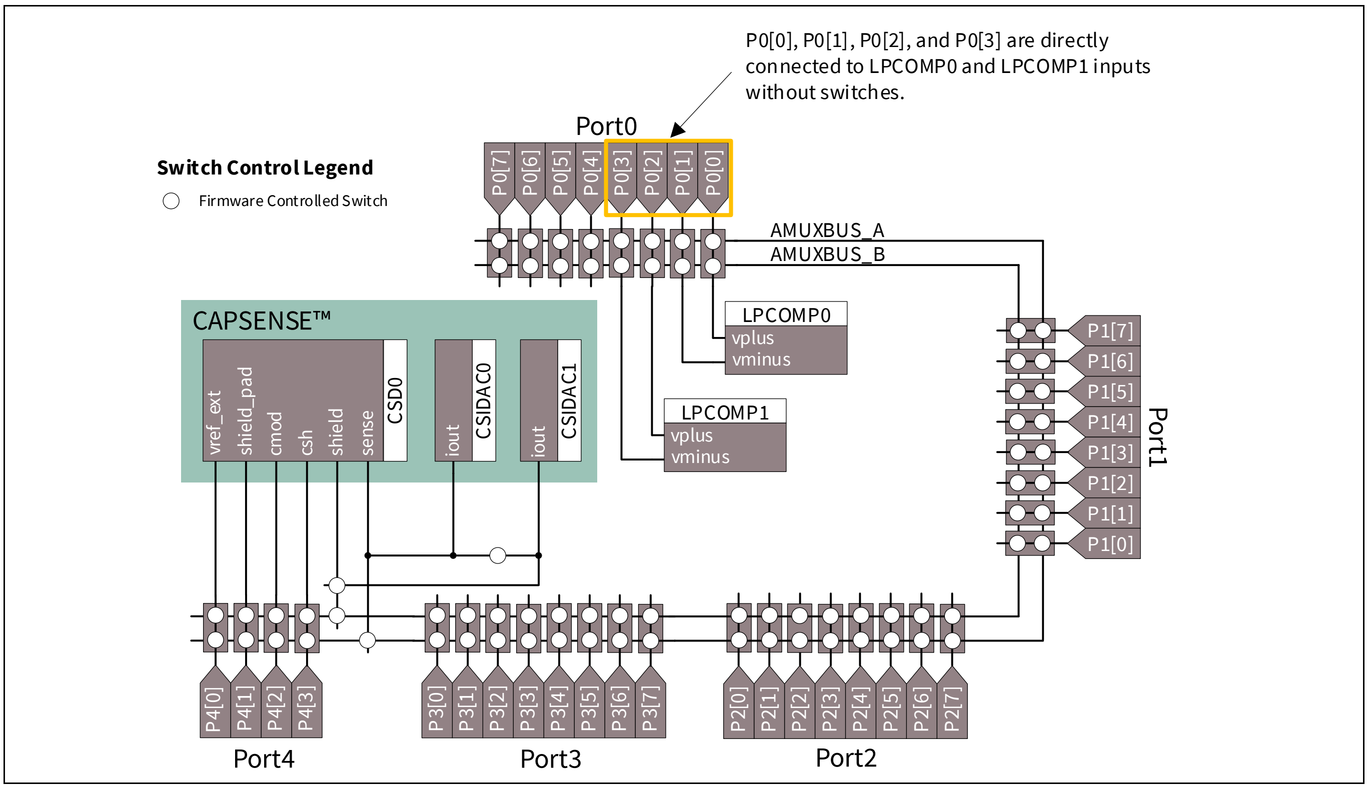


Figure 12 **PSoC™ 4000S analog routing diagram**

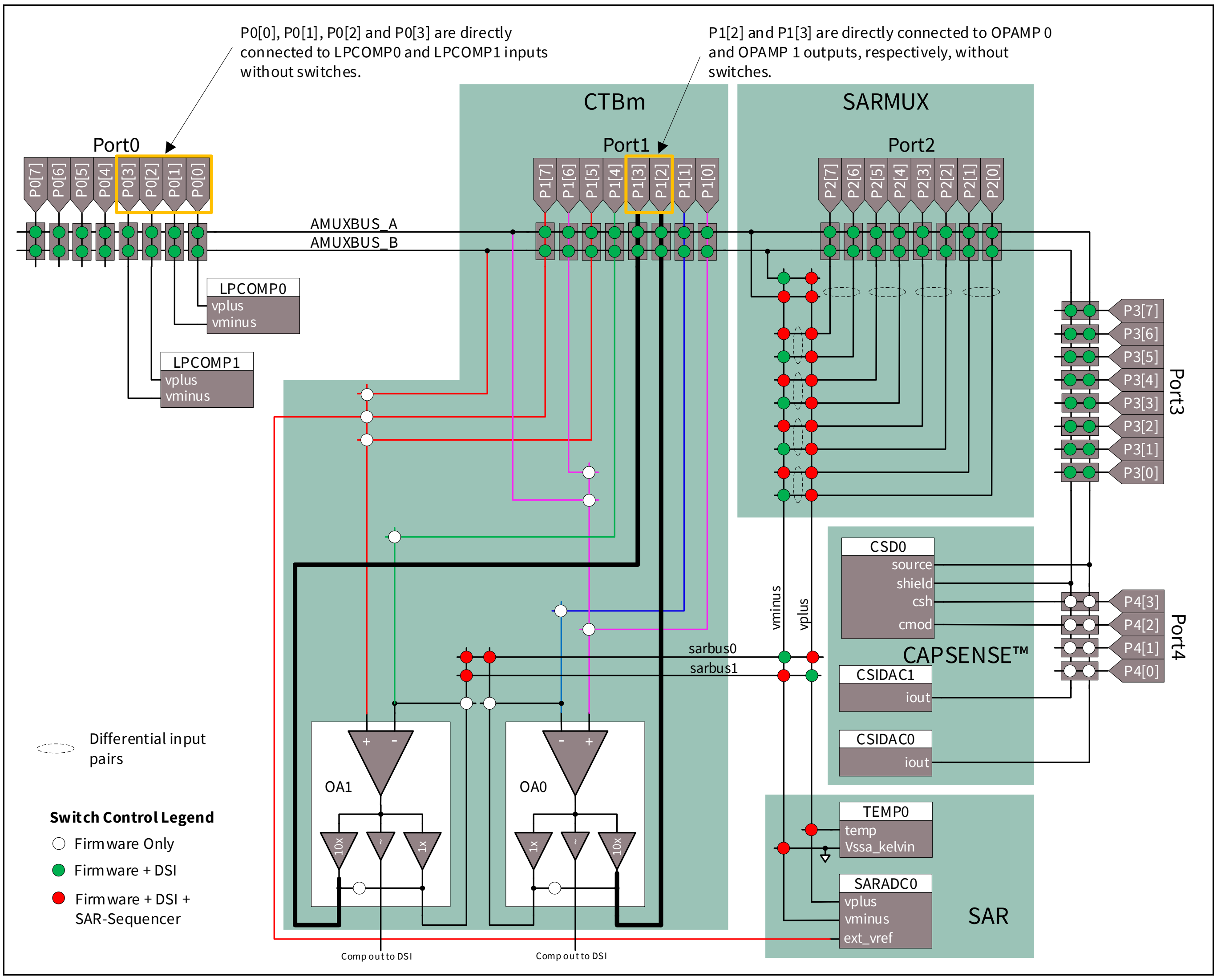


Figure 13 **PSoC™ 4200/PSoC™ 4100 analog routing diagram**

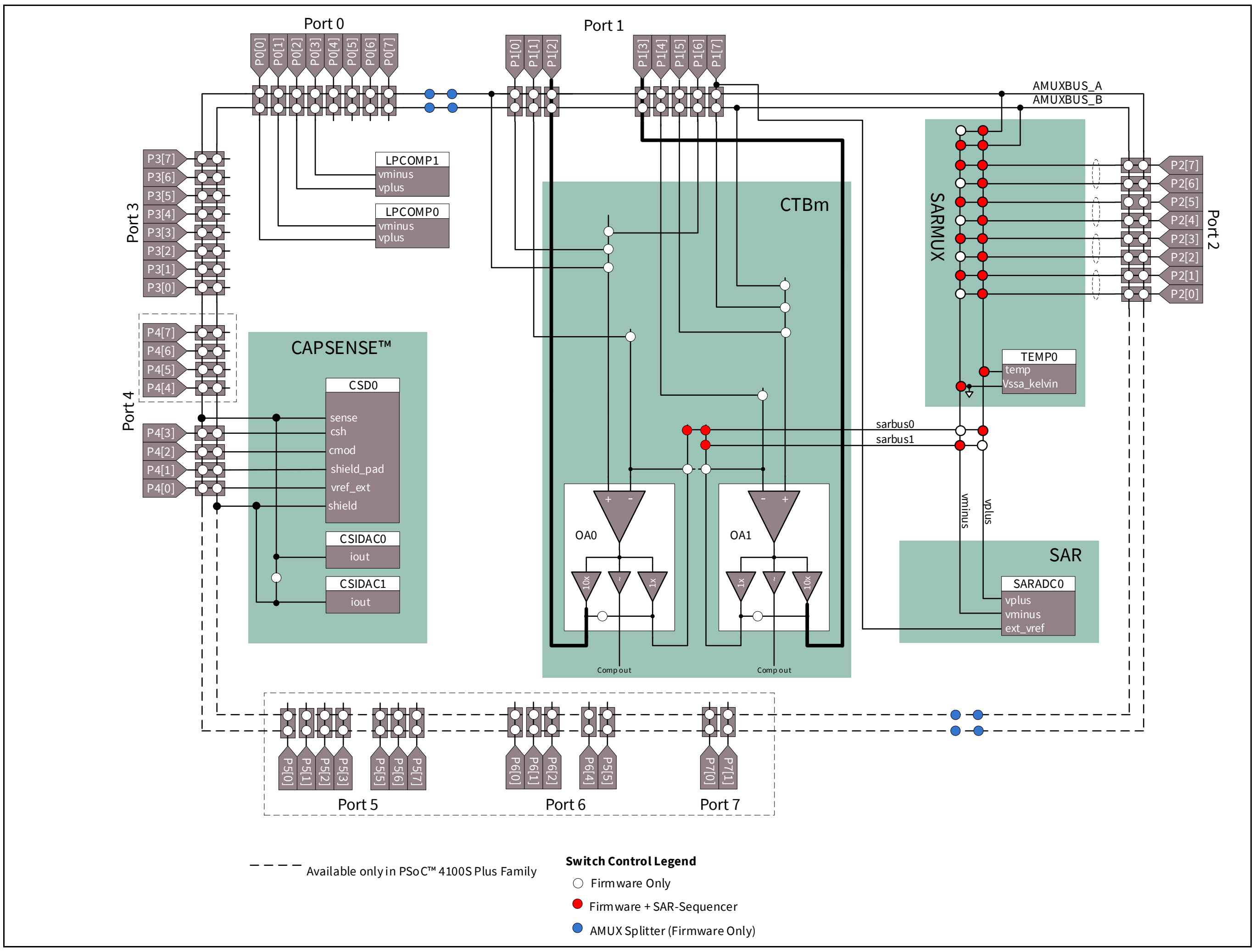


Figure 14 PSoC™ 4100S/4100S Plus analog routing diagram

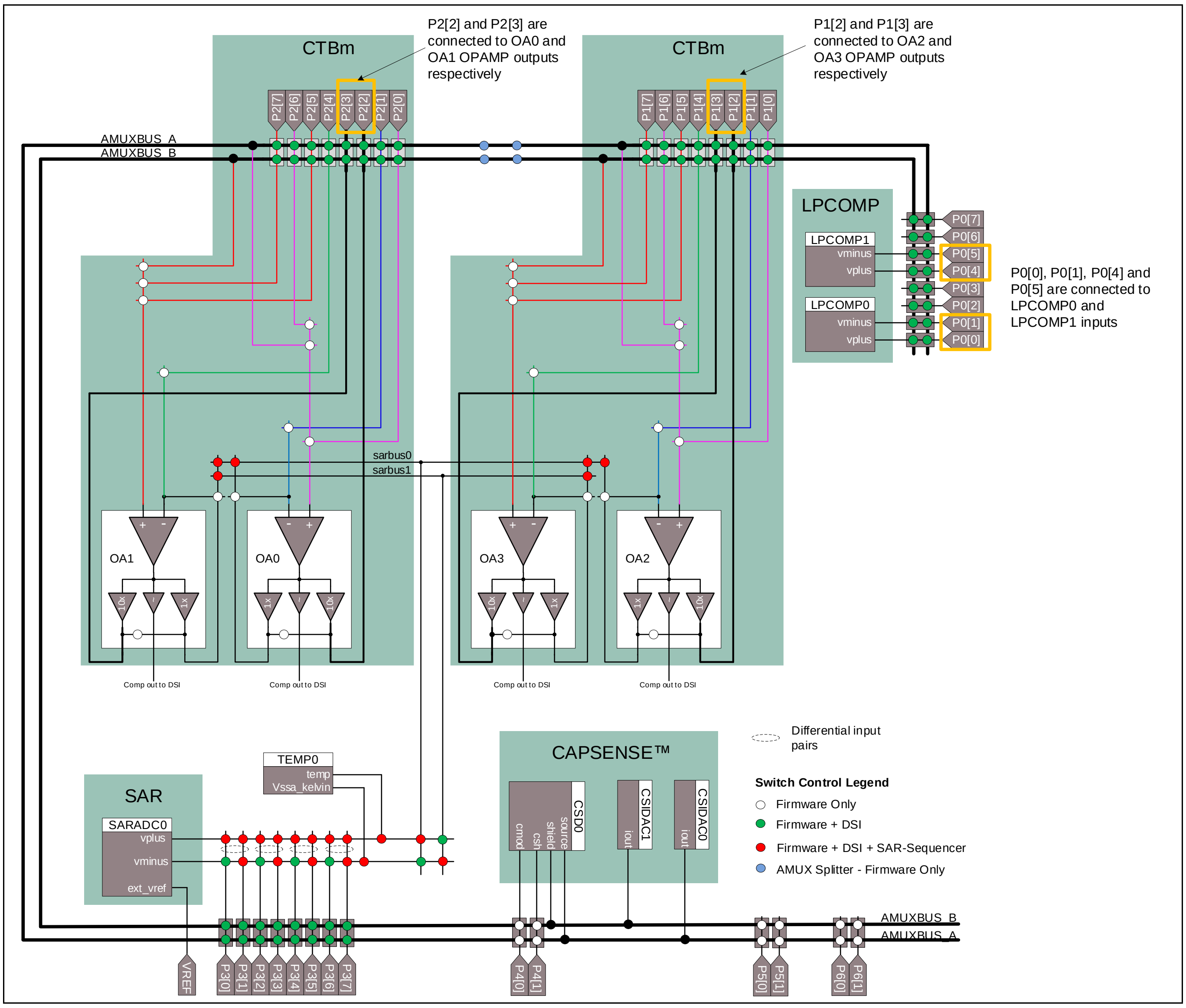

P2[2] and P2[3] are connected to OA0 and OA1 OPAMP outputs respectively
P1[2] and P1[3] are connected to OA2 and OA3 OPAMP outputs respectively
P0[0], P0[1], P0[4] and P0[5] are connected to LPCOMP0 and LPCOMP1 inputs
CTBm
AMUXBUS_A
AMUXBUS_B
LPCOMP
LPCOMP1
LPCOMP0
vminus
vplus
sarbus0
sarbus1
OA0
OA1
OA2
OA3
Comp out to DSI
SAR
SARADC0
ext_vref
TEMP0
temp
Vssa_kelvin
CAPSENSE™
CSD0
CSIDAC1
CSIDAC0
cmod
source
shield
csh
iout
VREF
Differential input pairs
Switch Control Legend
○ Firmware Only
● Firmware + DSI
● Firmware + DSI + SAR-Sequencer
● AMUX Splitter - Firmware Only

Figure 15 PSoC™ 41xx-BL/PSoC™ 42xx-BL analog routing diagram

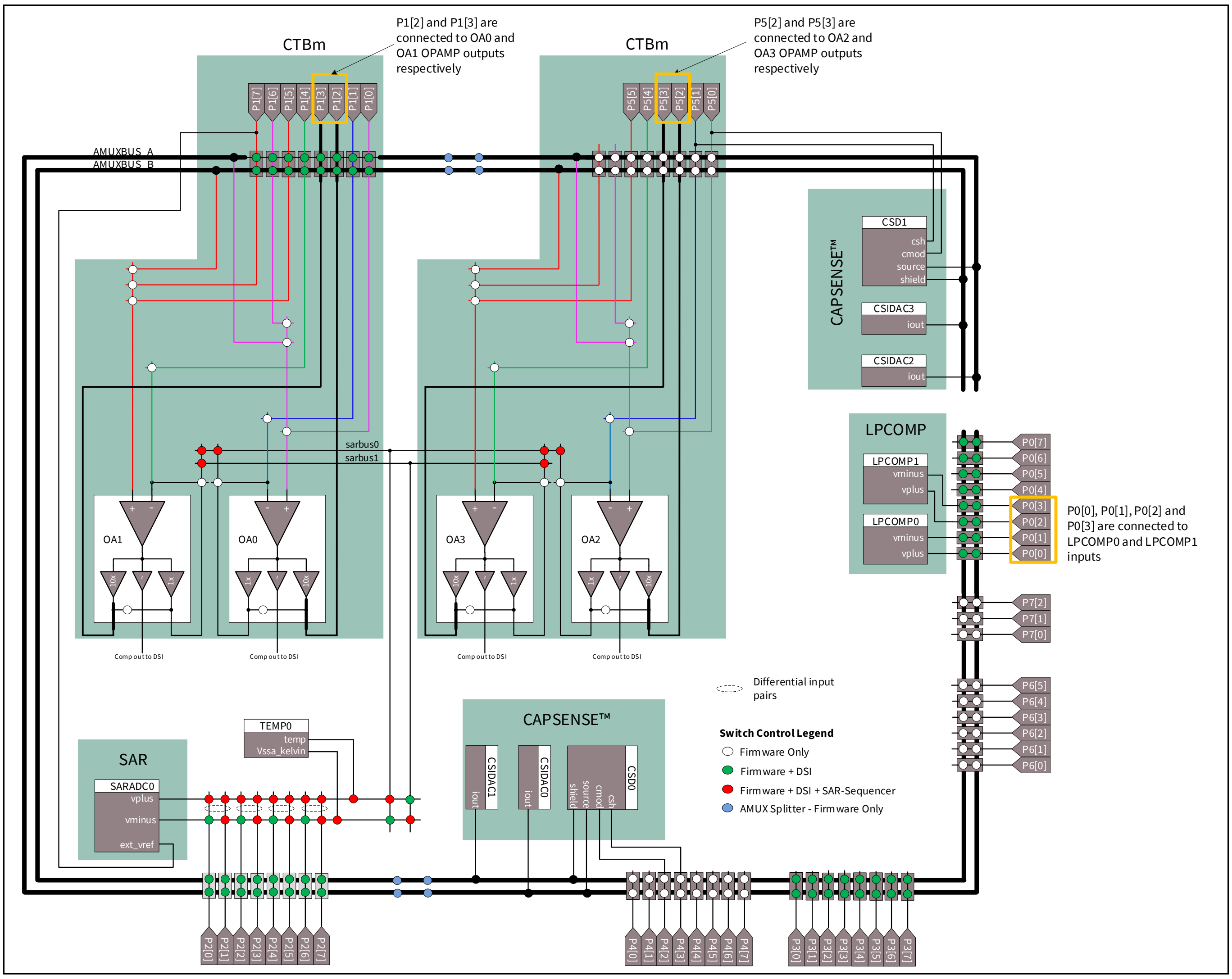

CTBm
CTBm
P1[2] and P1[3] are connected to OA0 and OA1 OPAMP outputs respectively
P5[2] and P5[3] are connected to OA2 and OA3 OPAMP outputs respectively
AMUXBUS_A
AMUXBUS_B
CAPSENSE™
CSD1
CSIDAC3
CSIDAC2
LPCOMP
LPCOMP1
LPCOMP0
P0[0], P0[1], P0[2] and P0[3] are connected to LPCOMP0 and LPCOMP1 inputs
OA1
OA0
OA3
OA2
sarbus0
sarbus1
Comp out to DSI
Differential input pairs
Switch Control Legend
Firmware Only
Firmware + DSI
Firmware + DSI + SAR-Sequencer
AMUX Splitter - Firmware Only
SAR
SARADC0
TEMP0
CAPSENSE™
CSIDAC1
CSIDAC0
CSD0

Figure 16 PSoC™ 4100M/PSoC™ 4200M analog routing diagram

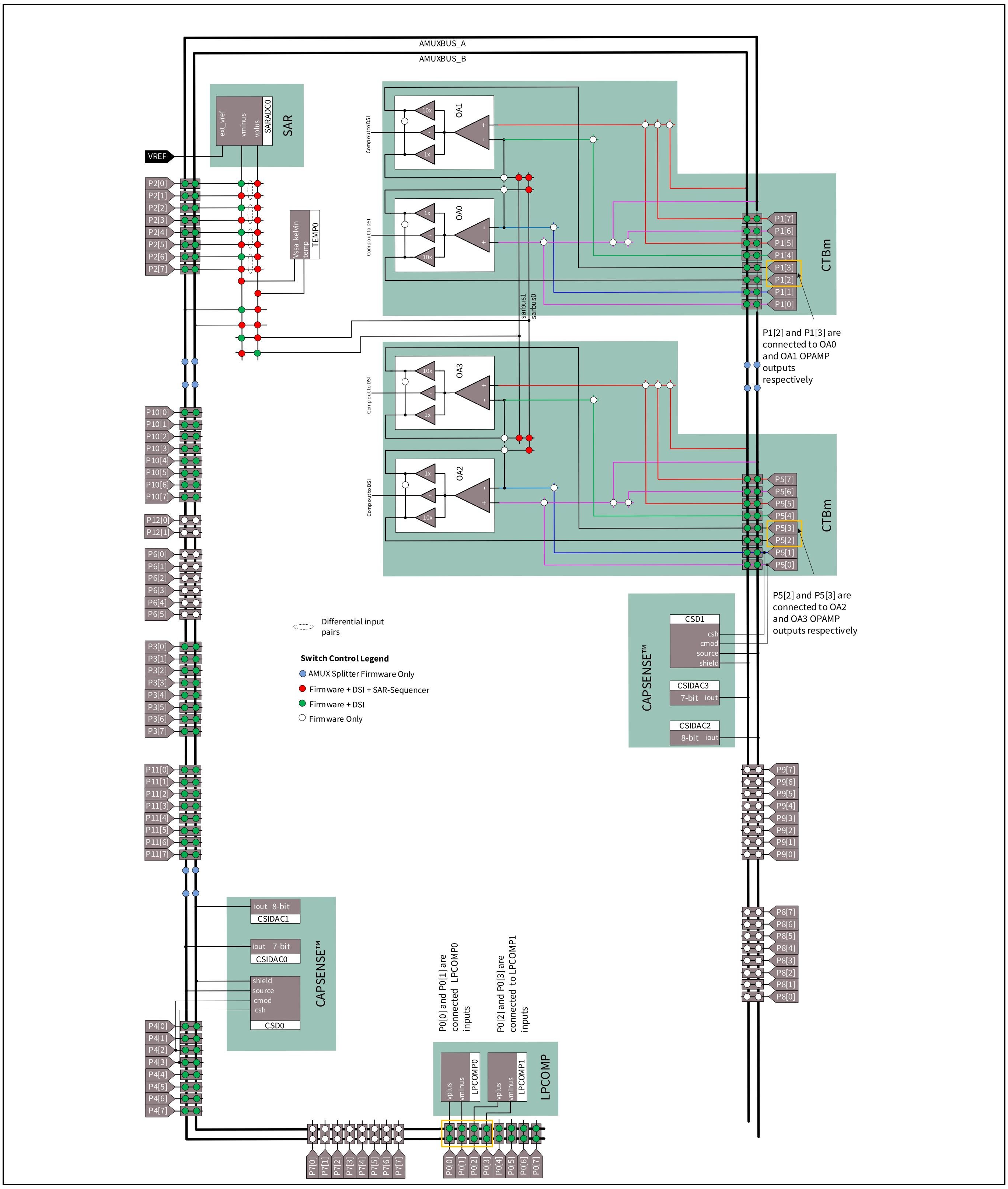

AMUXBUS_A
AMUXBUS_B
VREF
SAR
SARADC0
ext_vref
vminus
vplus
TEMP0
Vssa_kelvin
temp
OA0
OA1
OA2
OA3
CTBm
sarbus0
sarbus1
P1[2] and P1[3] are connected to OA0 and OA1 OPAMP outputs respectively
P5[2] and P5[3] are connected to OA2 and OA3 OPAMP outputs respectively
Differential input pairs
Switch Control Legend
AMUX Splitter Firmware Only
Firmware + DSI + SAR-Sequencer
Firmware + DSI
Firmware Only
CAPSENSE™
CSD1
csh
cmod
source
shield
CSIDAC3
7-bit iout
CSIDAC2
8-bit iout
CSIDAC1
CSIDAC0
CSD0
P0[0] and P0[1] are connected LPCOMP0 inputs
P0[2] and P0[3] are connected to LPCOMP1 inputs
LPCOMP
LPCOMP0
LPCOMP1
vplus
vminus

Figure 17 PSoC™ 4200L analog routing diagram

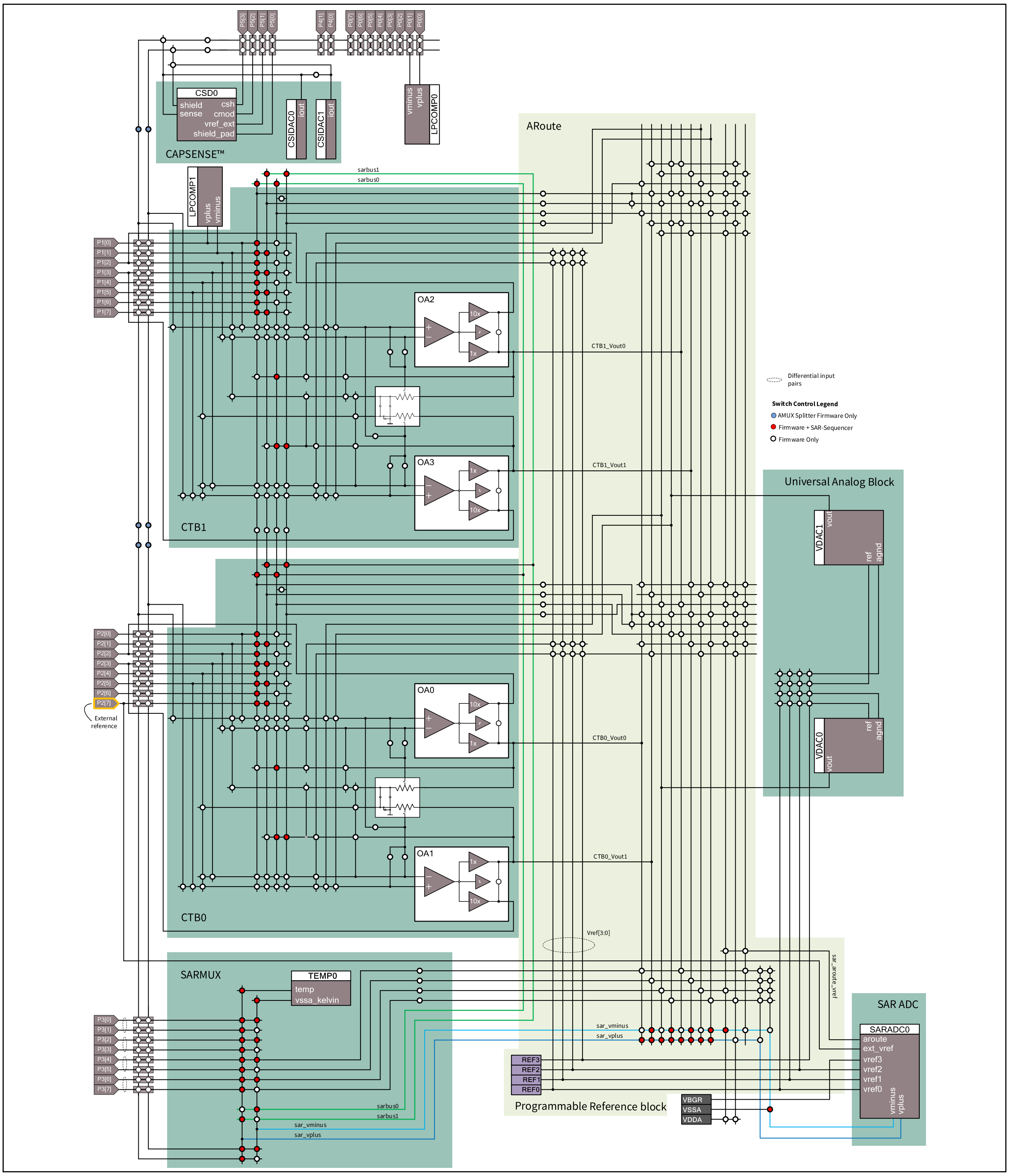


Figure 18 **PSoC™ 4100PS analog routing diagram**

Note: *The PSoC™ Creator IDE tool provides an analog routing diagram for a design similar to those illustrated in Figure 11 through Figure 18. See the Analog tab in the .cydwr file of the project in PSoC™ Creator.*

3.3 Startup and low-power behavior

On reset/power-up, all GPIO pins start up in the high-impedance analog mode, that is, with the input buffer and output driver disabled. These GPIO pins remain in this mode until the reset is released; then the initial operating configuration of the associated registers of each GPIO pin is loaded during boot and takes effect at that time. During run time, GPIOs can be configured by writing to the associated registers.

Note: *In the PSoC™ 4000 parts, 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. Note that the reset provision via P1[6] is only for production test purpose and not intended for user applications.*

See I/O system restrictions in the PSoC™ 4000 family – KBA91258 for more information.

PSoC™ has up to four power modes as follows:

Table 2 Low power modes in PSoC™ 4 families

Device	Sleep	Deep Sleep	Hibernate	Stop
PSoC™ 4000	✓	✓	✗	✗
PSoC™ 4000S	✓	✓	✗	✗
PSoC™ 4100/4200	✓	✓	✓	✓
PSoC™ 4100S	✓	✓	✗	✗
PSoC™ 4100S Plus	✓	✓	✗	✗
PSoC™ 4100S Max	✓	✓	✗	✗
PSoC™ 4100S Plus 256KB	✓	✓	✗	✗
PSoC™ 4200DS	✓	✓	✗	✗
PSoC™ 4500S	✓	✓	✗	✗
PSoC™ 4700S	✓	✓	✗	✗
PSoC™ analog coprocessor	✓	✓	✗	✗
PSoC™ 4 Bluetooth® LE	✓	✓	✓	✓
PSoC™ 4 M	✓	✓	✓	✓
PSoC™ 4 L	✓	✓	✓	✓
PSoC™ 4100PS	✓	✓	✗	✗

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 peripherals such as I^{2}C, LCD driver, opamp, and comparator are functional. The I^{2}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.

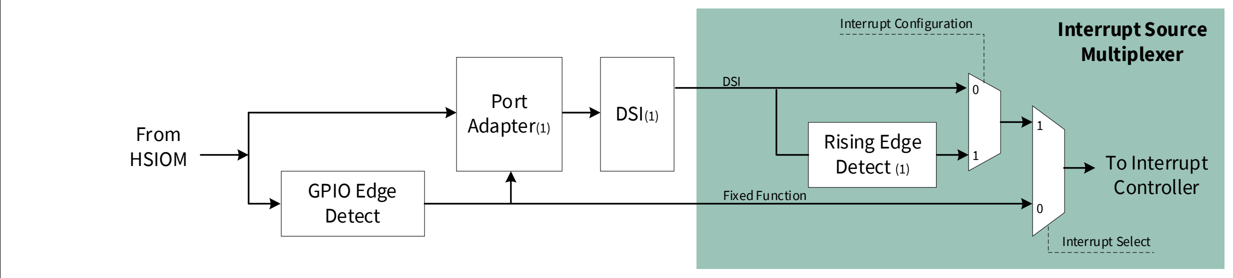
The PSoC™ 4 parts (except PSoC™ 4000) have an additional feature that freezes the GPIOs in Deep Sleep, Hibernate, and Stop modes. Unfreezing of GPIOs also happens automatically when the low-power mode is exited. However, note that the GPIOs driven by Deep Sleep peripherals are active in Deep Sleep mode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

In the case of Hibernate and Stop modes, wakeup happens with a device reset. This clears the GPIO configuration and the pin state. To retain the pin state, use the `CySysPmFreezeIo()` and `CySysPmUnfreezeIo()` API functions. Note that you do not need to call `CySysPmFreezeIo()` for Stop mode because it is automatically called when the user invokes Stop mode using the `CySysPmStop()` API function. However, you should call `CySysPmFreezeIo()` just before the function call to enter Hibernate mode. The GPIOs are unlocked by calling `CySysPmUnfreezeIo()`. A call to this function 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. UDB-based Components such as 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 a '1'. To avoid this glitch, the GPIO should be frozen before entering a Low-power mode.

For information on Low-power modes, see AN86233 – PSoC™ 4 MCU low-power modes and power reduction techniques.

3.4 GPIO interrupt



(1) Not applicable to PSoC™ 4000, PSoC™ 4000S, PSoC™ 4100S, PSoC™ 4100S Plus, PSoC™ 4100PS

Figure 19 GPIO interrupt signal routing to the interrupt controller

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 the interrupt and provides an option of rising-edge detection or 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/Cortex® M0+ CPU. The 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 or direct connection.

Note: The interrupt vector table is available in the Interrupts chapter of the Reference manual.

The use of Interrupt Source Multiplexer is not limited to the GPIO interrupts; it is also used for all other sources. To know more about other interrupt sources, see the "Interrupt Sources" section of AN90799 - PSoC™ 4 interrupts.

The GPIO interrupt, in addition to the resources present in the Interrupt Source Multiplexer, uses its own GPIO Edge Detect block as showin in Figure 19.

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

See Pins component interrupts to know how different routes are configured.

The following figure 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.

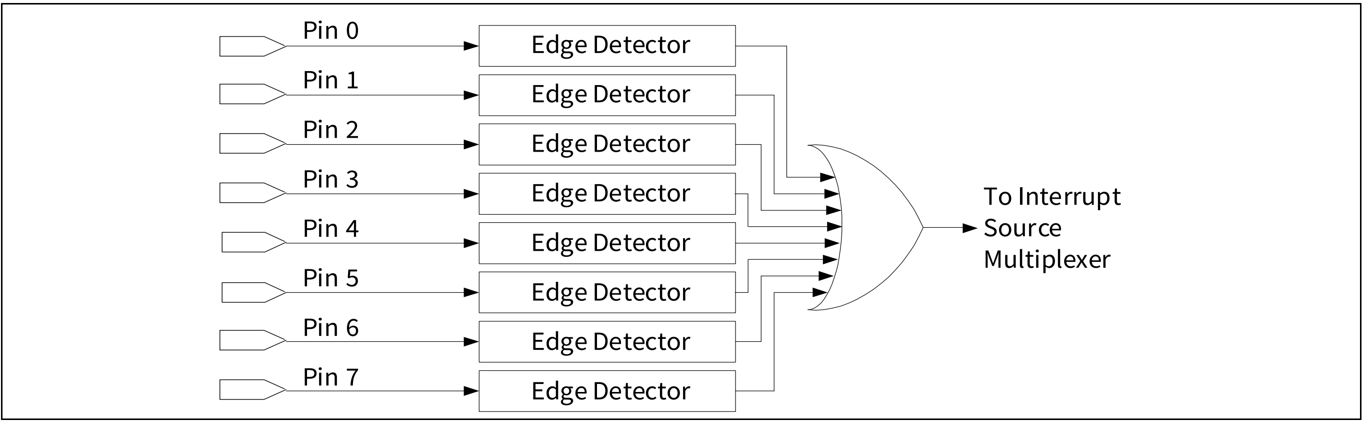


Figure 20 GPIO edge detect

As it is clear from Figure 20, when an 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:

1. Single interrupt trigger and nonresponsive to further interrupts when the Interrupt Source Multiplexer is configured to rising-edge. This scenario occurs if Route 2 is used.
2. Repetitive interrupts for a single request when the Interrupt Source Multiplexer is configured to direct connection. This scenario occurs if Route 1 or Route 3 is used.
3. 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 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, PSoC™ 4M, and PSoC™ 4L parts. For a PSoC™ 4 Bluetooth® LE device, the wakeup pin is P2[2].*

3.4.1 Limitations in GPIO interrupt

- Port 4 and higher ports do not have a port adapter. Thus, pin interrupt via DSI routing is not possible for these port pins.
- PSoC™ 4000 and PSoC™ 4100/PSoC™ 4200 have one interrupt vector for each port. PSoC™ 4 Bluetooth® LE does not have a dedicated interrupt vector for the ports beyond Port 5, while PSoC™ 4M 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.
- See the "Interrupts" chapter in the respective device architecture reference manual to learn about ports that have a common interrupt vector.

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.

4 Overvoltage-tolerant (OVT) pins

Pins P5[0] and P5[1] in PSoC™ 4 Bluetooth® LE, and Port 6 in PSoC™ 4M are the OVT pins. For PSoC™ 4L, Port 6 and Port 8 have OVT pins. These are similar to regular GPIOs with the following additional features:

1. Overvoltage-tolerant: There is no ESD clamp diode between OVT pin and the supply rail. This enables the OVT pin to withstand an external voltage higher than VDDIO, VDDD, or VDDA voltage, up to 5.5 V.
2. Provides better pull-down drive strength compared to a regular GPIO
3. Serial Communication Block (SCB): When configured as I^{2}C and its lines routed to OVT pins, an SCB meets the following I^{2}C specifications:
   a) Fast Mode Plus LOW-level output current (IOL) specification
   b) Fast Mode and Fast Mode Plus hysteresis and minimum fall-time specifications

For more details on the I/O hardware, see the I/O system chapter of the Reference manual.

5 GPIO pins in PSoC™ Creator

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

5.1 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 Infineon 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 follows:

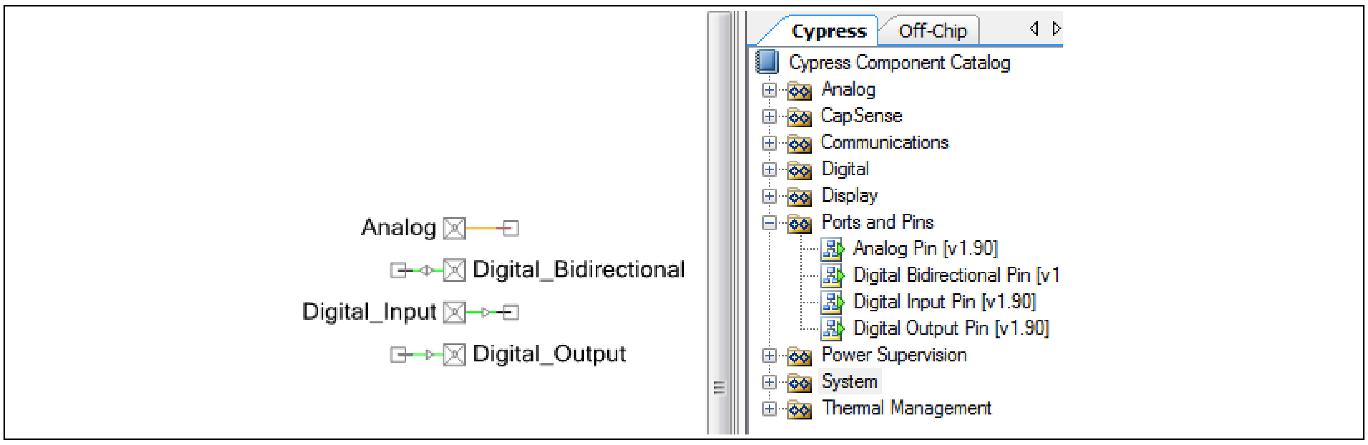


Figure 21 Pins component symbol types in PSoC™ Creator

5.2 Pins component customizer

Each component in PSoC™ Creator comes with a customizer to configure the component. Figure 22 shows the Pin component customizer, which is accessed by double-clicking the component.

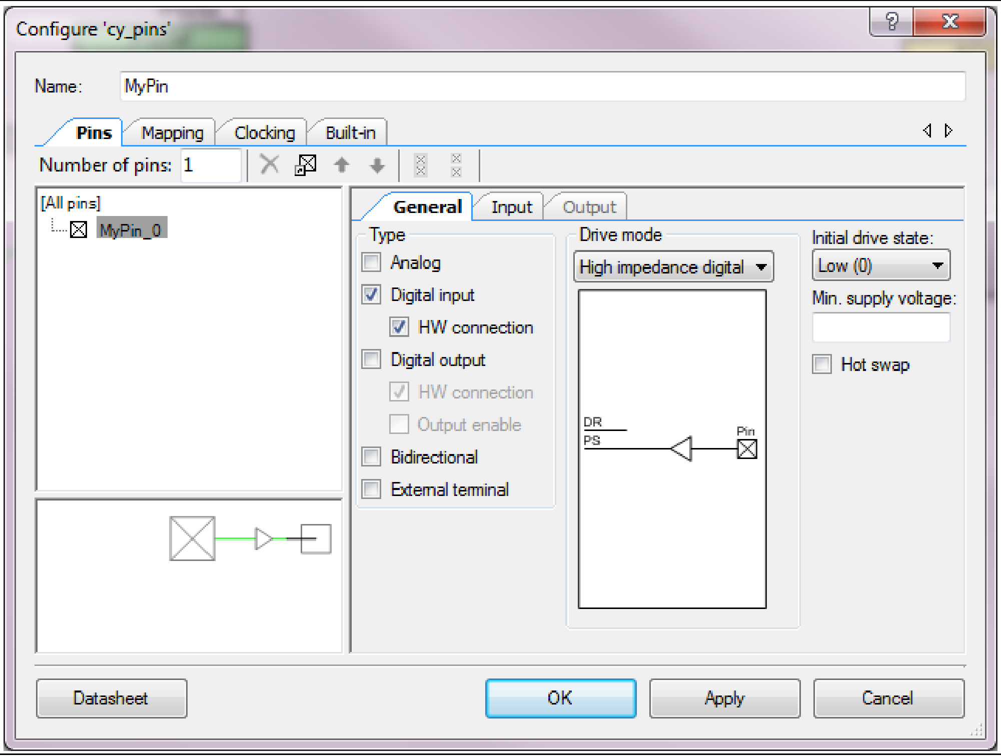


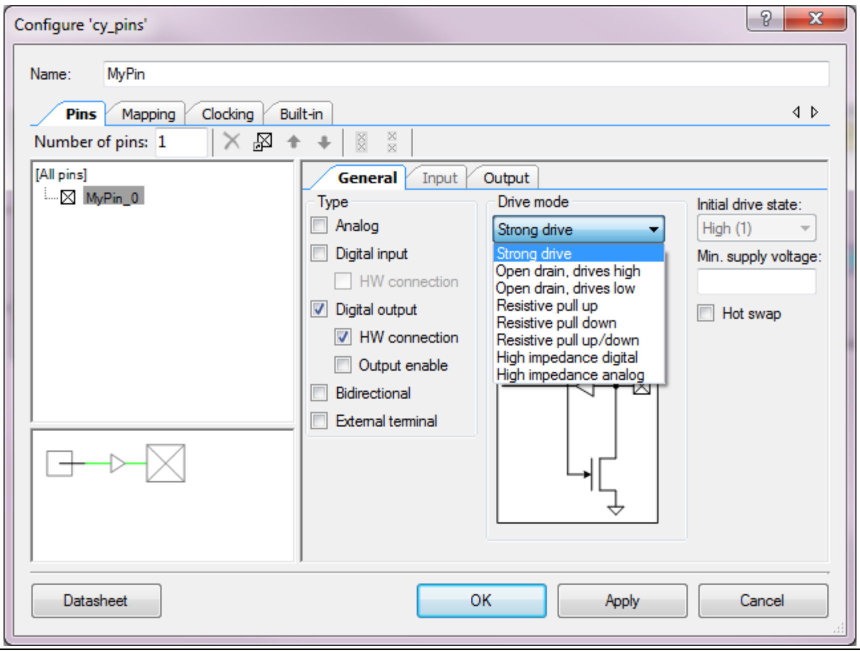
Figure 22 **Pin component customizer**

The following table describes some of the parameters in the Pin component 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 that 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 GPIO pin basics section. Figure 23 shows 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 the 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 the default value by PSoC™ Creator, as shown in Figure 23. 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

Setting	Description
	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 shown in Figure 24. 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 GPIO interrupt section. For more details on this setting, see Pins component interrupts.


Configure 'cy_pins'

Name: MyPin

Number of pins: 1

Type: Analog [ ], Digital input [ ], HW connection [ ], Digital output [✓], HW connection [✓], Output enable [ ], Bidirectional [ ], External terminal [ ]

Drive mode: Strong drive

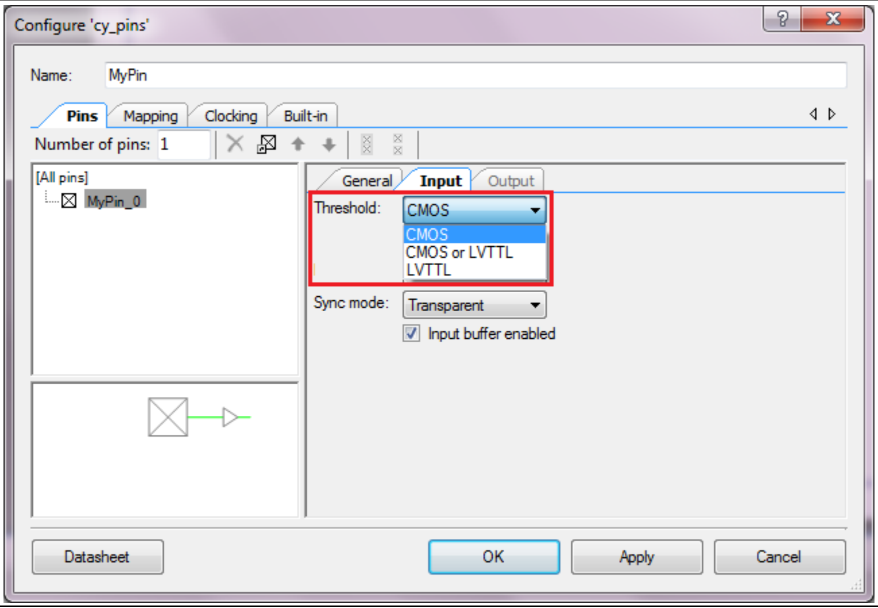
Options: Strong drive; Open drain, drives high; Open drain, drives low; Resistive pull up; Resistive pull down; Resistive pull up/down; High impedance digital; High impedance analog

Initial drive state: High (1)

Min. supply voltage:

Hot swap [ ]

Figure 23 Pin Drive mode setting and initial drive state


Configure 'cy_pins'

Name: MyPin

Number of pins: 1

Threshold: CMOS

Options: CMOS; CMOS or LVTTL; LVTTL

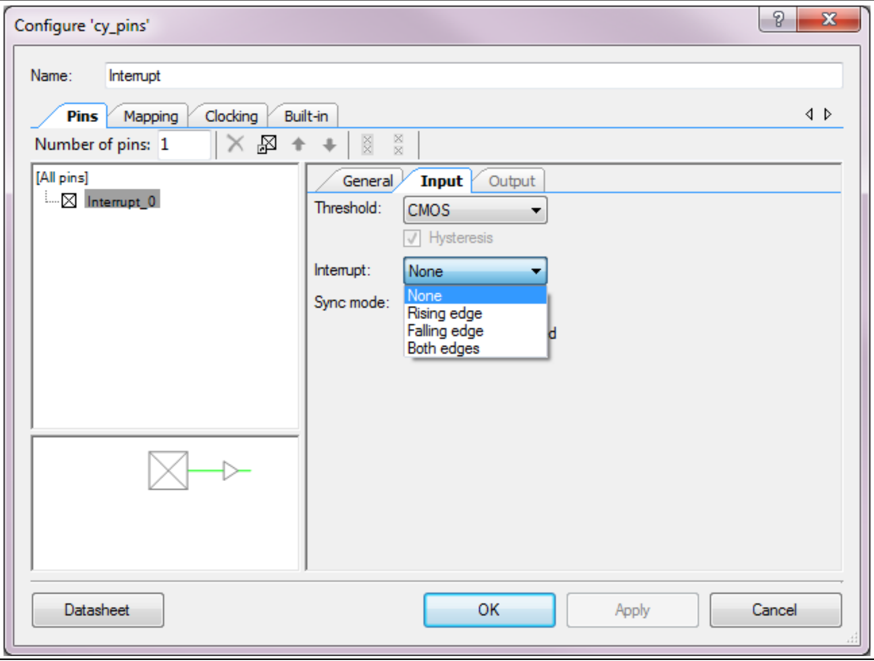
Sync mode: Transparent

Input buffer enabled [✓]

Figure 24 Pin input threshold selection

5.3 Pins component interrupts

The Interrupt parameter in the pin customizer configures the GPIO Edge Detect block described in the GPIO interrupt section.


Configure 'cy_pins'

Name: Interrupt

Pins | Mapping | Clocking | Built-in

Number of pins: 1

[All pins]
Interrupt_0

General | Input | Output

Threshold: CMOS

Hysteresis

Interrupt: None

Sync mode:

None
Rising edge
Falling edge
Both edges

Figure 25 Interrupt configuration in PSoC™ Creator

The Pins Component symbol changes when interrupts are enabled, as shown in Figure 26.

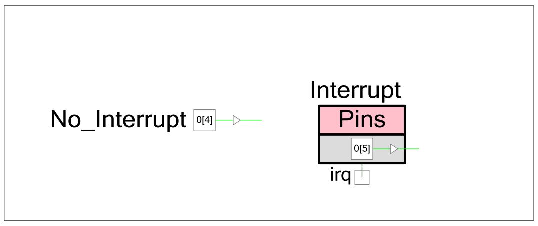


Figure 26 Pins component symbol with interrupts enabled

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. These components cannot be mapped to pins in the same physical port.

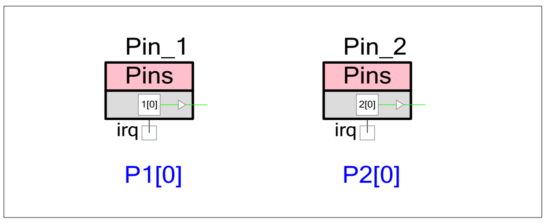


Figure 27 Two pins components with interrupts enabled

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. 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.

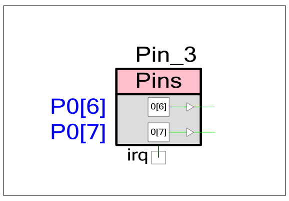


Figure 28 Multiple pins in the same port with interrupts enabled

The IRQ of the Pin Ccomponent should be connected to the interrupt component. This routes the GPIO interrupt signal to the interrupt controller.

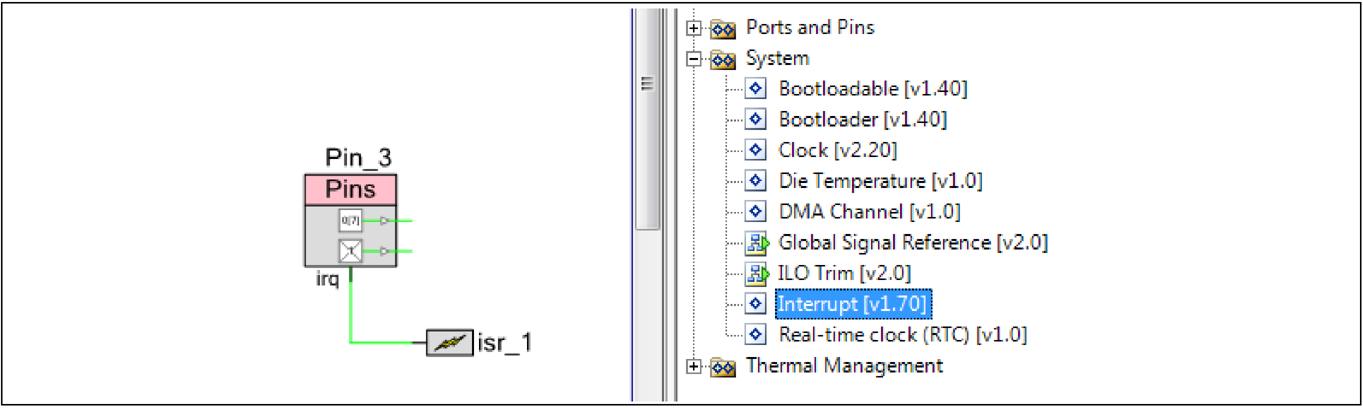
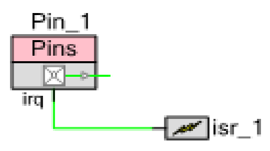
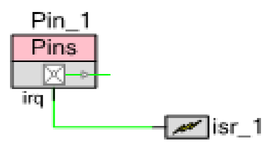
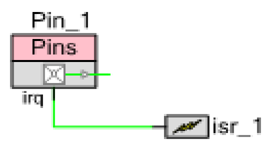
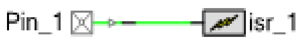
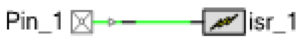


Figure 29 Interrupt component in the catalog

The interrupt component configures the interrupt source multiplexer to either direct connection (shown as "level" in the Interrupt component customizer) or 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 the help of the pin component and the interrupt component customizer settings summarized as follows:

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. This

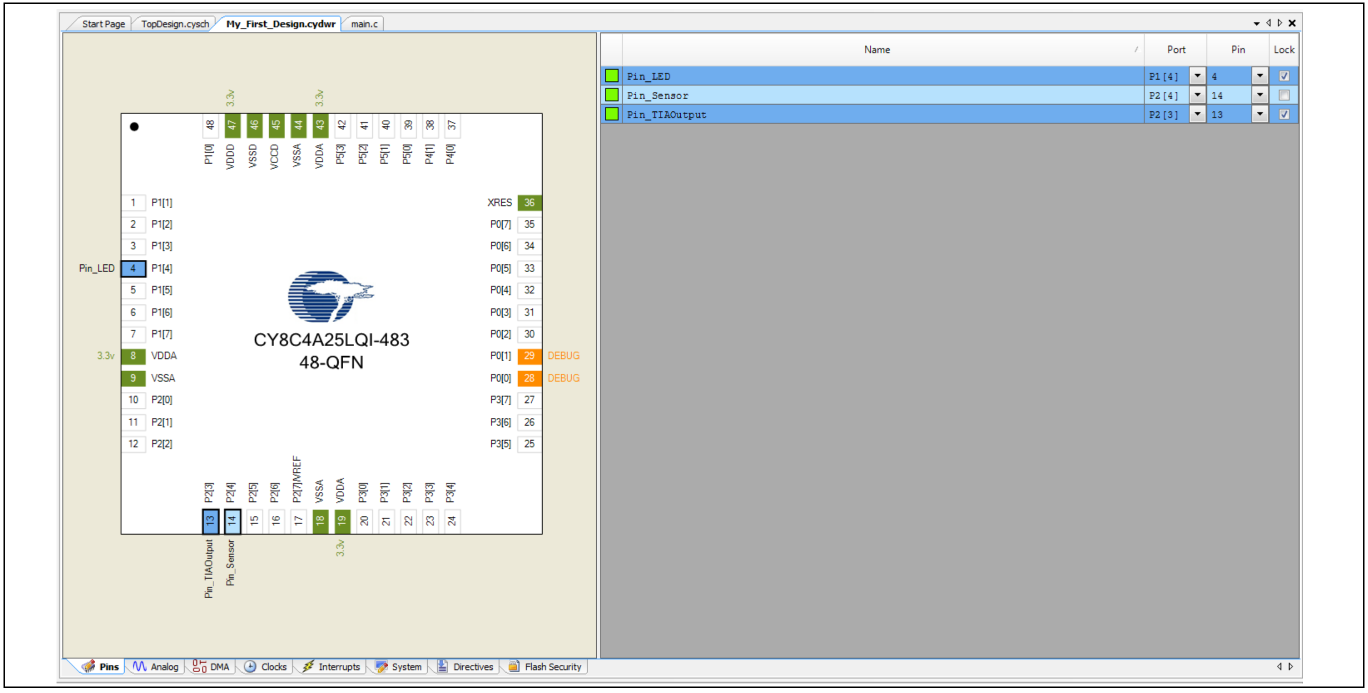
Schematic	Interrupt setting in pin component	Interrupt component setting	Route	Details
				configuration can be used to wake up the device from any low-power mode. However, note that wakeup from Stop mode requires the use of a specific pin depending on the selected device.
	Rising-edge or falling-edge 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. This configuration can wake up the CPU only from Sleep mode; it will not work in other Low-power modes.
	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 AN90799 – PSoC™ 4 interrupts to know how to force the interrupt vector. This configuration can wake up the CPU from only Sleep mode; it will not work in other Low-power modes.
	Disabled	Rising-edge	Route 4	This configuration provides rising-edge interrupt. In this case, there is no need to clear the interrupt. This configuration can wake up the CPU only from Sleep mode; it will not work in other Low-power modes.
	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. This configuration can wake up the CPU from only Sleep mode; it will not work in other Low-power modes.

5.4 Manual pin assignments

You can use the **Pins** tab of the Design-Wide Resources (DWR) window to assign a Pin 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 30 shows three assigned pins. The pins highlighted in dark blue are manually assigned and the pin highlighted in light blue 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.



	Name	Port	Pin	Lock
🟩	Pin_LED	P1[4]	4	✓
🟩	Pin_Sensor	P2[4]	14	
🟩	Pin_TIAOutput	P2[3]	13	✓

Figure 30 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. See PSoC™ Creator Help for more details.*

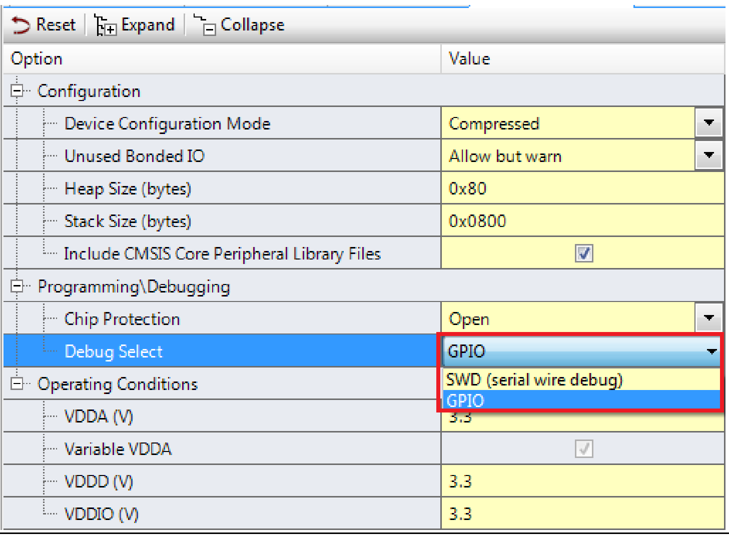
5.5 PSoC™ Creator APIs

Infineon provides a set of API functions that you can use to control GPIOs dynamically through 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.

5.6 Debug logic on GPIO pins

The PSoC™ 4 serial wire debug (SWD) pins are shared on the port pins. See 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.



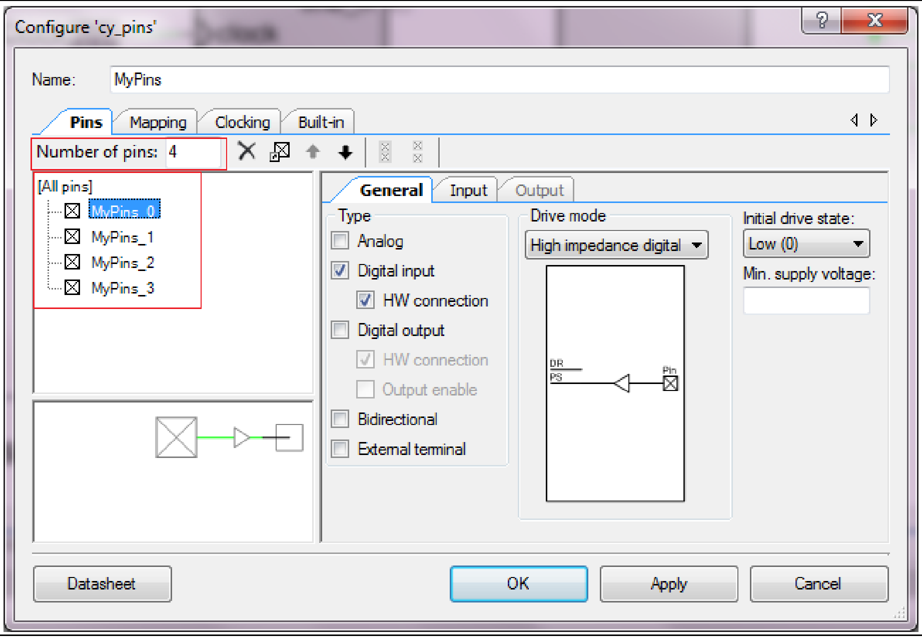
Option	Value
Configuration	
Device Configuration Mode	Compressed
Unused Bonded IO	Allow but warn
Heap Size (bytes)	0x80
Stack Size (bytes)	0x0800
Include CMSIS Core Peripheral Library Files	✓
Programming\Debugging	
Chip Protection	Open
Debug Select	GPIO
Operating Conditions	
VDDA (V)	3.3
Variable VDDA	✓
VDDD (V)	3.3
VDDIO (V)	3.3

Figure 31 Debug port disabled

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

5.7 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 shown in Figure 32. Each pin can be configured independently. Select **[All Pins]** to configure every pin in the component with the same settings.


Configure 'cy_pins'

Name: MyPins

Pins | Mapping | Clocking | Built-in

Number of pins: 4

[All pins]
- MyPins_0
- MyPins_1
- MyPins_2
- MyPins_3

General | Input | Output

Type
- [ ] Analog
- [✓] Digital input
  - [✓] HW connection
- [ ] Digital output
  - [✓] HW connection
  - [ ] Output enable
- [ ] Bidirectional
- [ ] External terminal

Drive mode: High impedance digital

Initial drive state: Low (0)

Min. supply voltage:

Datasheet | OK | Apply | Cancel

Figure 32 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 33.

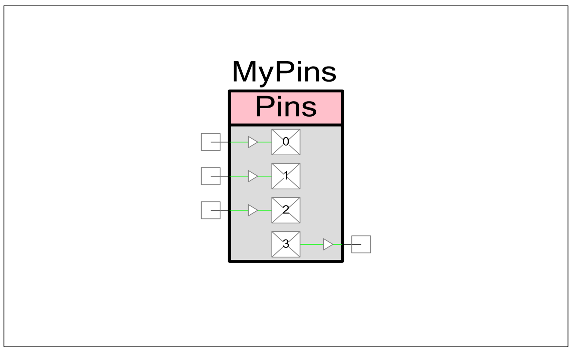
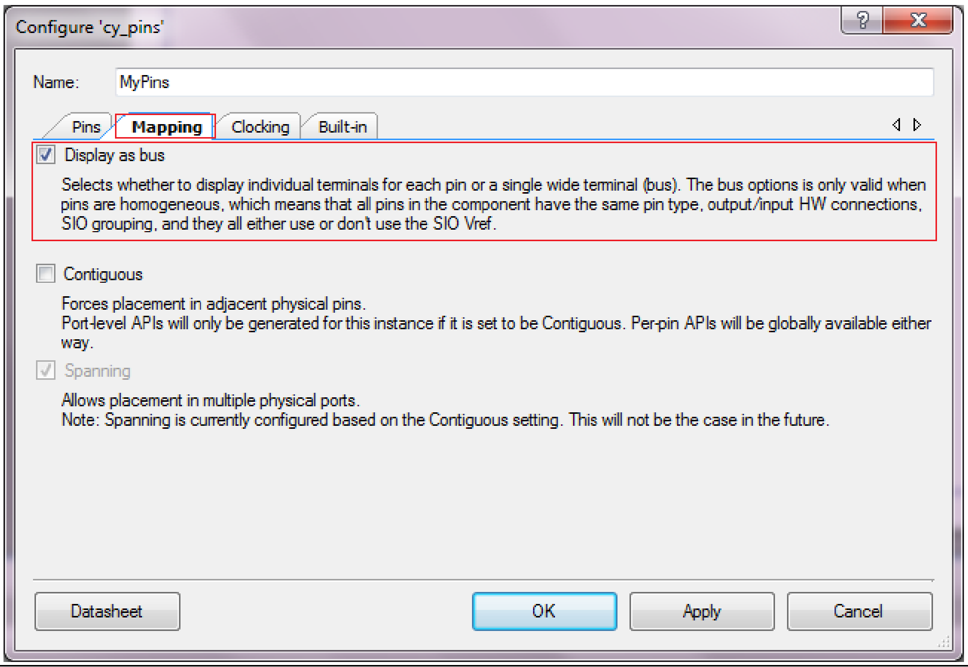


Figure 33 Pins Component in port configuration

There is an option to display the port with a bus instead of individual pin terminals. Select **Display as Bus** in the **Mapping** tab of the Pin Configuration window to display the port as a bus. Note that all pins must be of the same type to display as a bus.



Name: MyPins

☑ Display as bus

Selects whether to display individual terminals for each pin or a single wide terminal (bus). The bus options is only valid when pins are homogeneous, which means that all pins in the component have the same pin type, output/input HW connections, SIO grouping, and they all either use or don't use the SIO Vref.

☐ Contiguous

Forces placement in adjacent physical pins.
Port-level APIs will only be generated for this instance if it is set to be Contiguous. Per-pin APIs will be globally available either way.

☑ Spanning

Allows placement in multiple physical ports.
Note: Spanning is currently configured based on the Contiguous setting. This will not be the case in the future.

Figure 34 Display as bus option

If the **Number of Pins** is configured to four digital outputs, the schematic symbol appears:

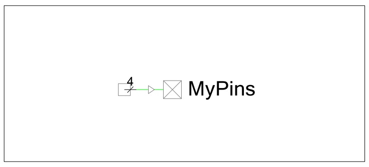
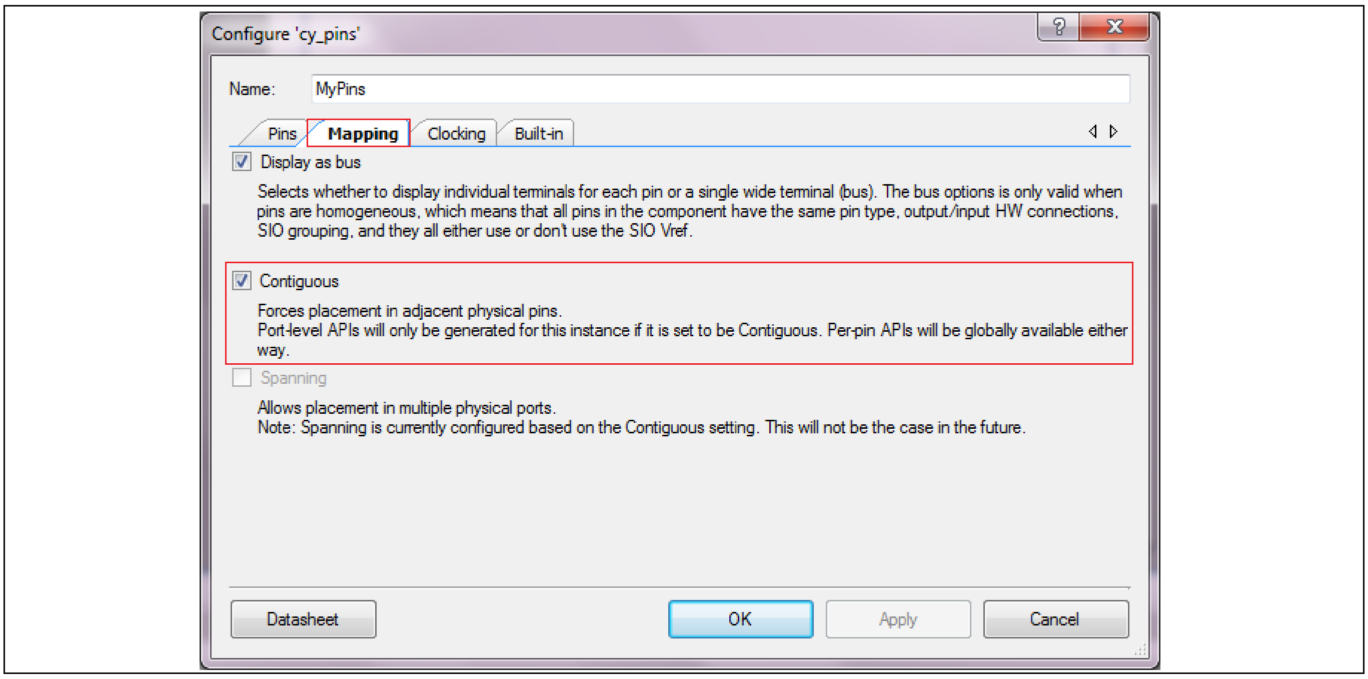


Figure 35 Four pins displayed with bus

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



☑ Display as bus

Selects whether to display individual terminals for each pin or a single wide terminal (bus). The bus options is only valid when pins are homogeneous, which means that all pins in the component have the same pin type, output/input HW connections, SIO grouping, and they all either use or don't use the SIO Vref.

☑ Contiguous

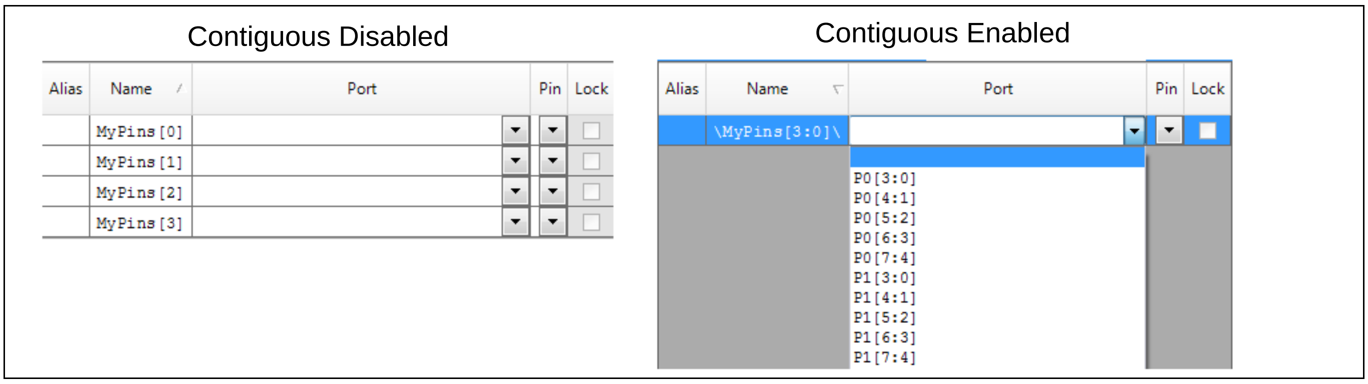
Forces placement in adjacent physical pins.
Port-level APIs will only be generated for this instance if it is set to be Contiguous. Per-pin APIs will be globally available either way.

☐ Spanning

Allows placement in multiple physical ports.
Note: Spanning is currently configured based on the Contiguous setting. This will not be the case in the future.

Figure 36 **Contiguous pin placement option**

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


Contiguous Disabled

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

Contiguous Enabled

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]
```

Figure 37 **Pin placement with contiguous disabled/enabled**

These features are described in more detail in the Pin Configuration window and the Pins Component datasheet.

5.8 Represent Off-chip components

The off-chip components catalog provides a way to mix external and internal components on the same schematic. 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.

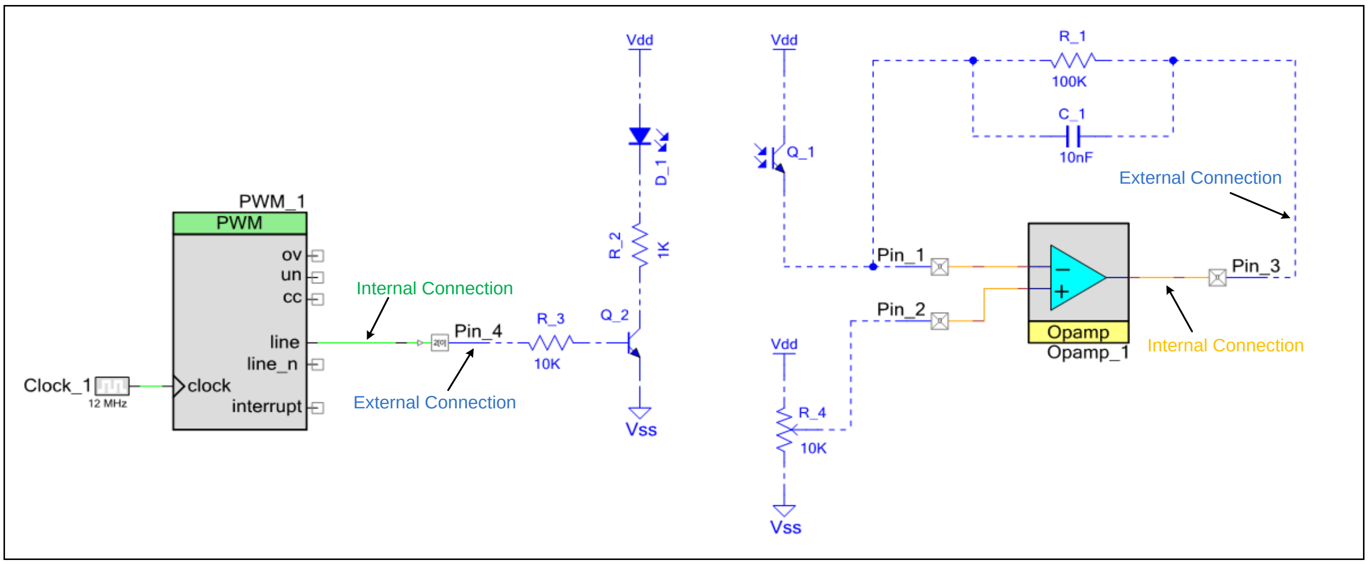
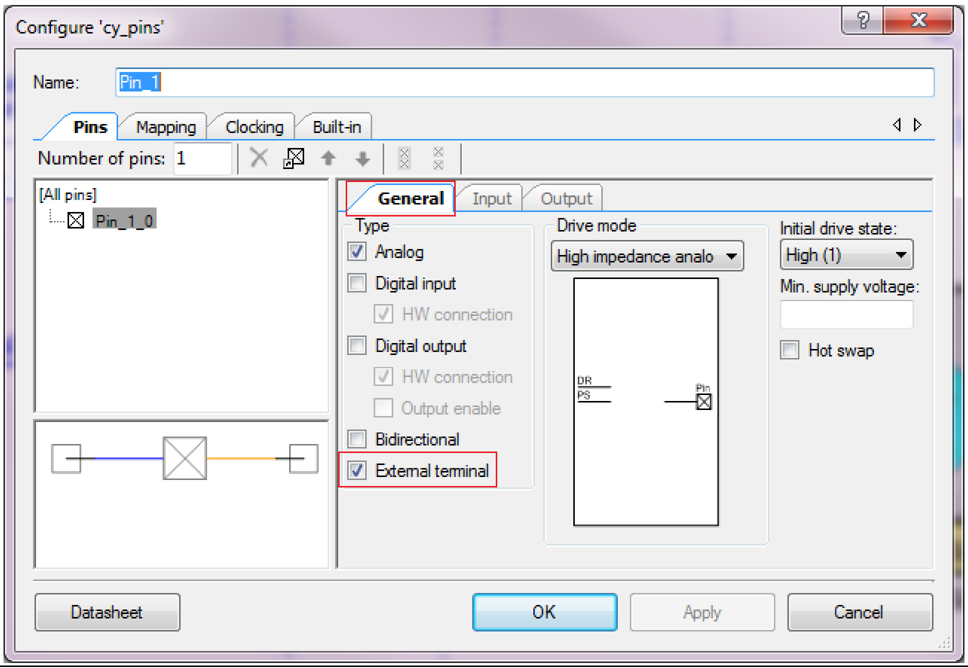


Figure 38 **Design with off-chip components**

In the design shown above, 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. This brings out an additional terminal on the schematic.


Configure 'cy_pins'

Name: Pin_1

Number of pins: 1

[All pins]
Pin_1_0

Type
- [✓] Analog
- [ ] Digital input
- [✓] HW connection
- [ ] Digital output
- [✓] HW connection
- [ ] Output enable
- [ ] Bidirectional
- [✓] External terminal

Drive mode: High impedance analo

Initial drive state: High (1)

Min. supply voltage:

- [ ] Hot swap

Figure 39 **Enabling external terminal**

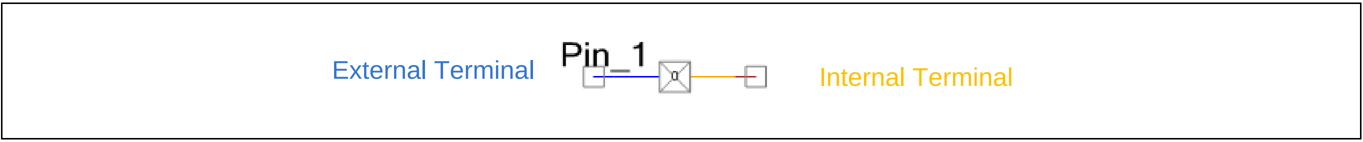


Figure 40 Pin component with internal and external terminal

The components, to connect to the external terminals on the schematic, are available in the off-chip tab in the components catalog.

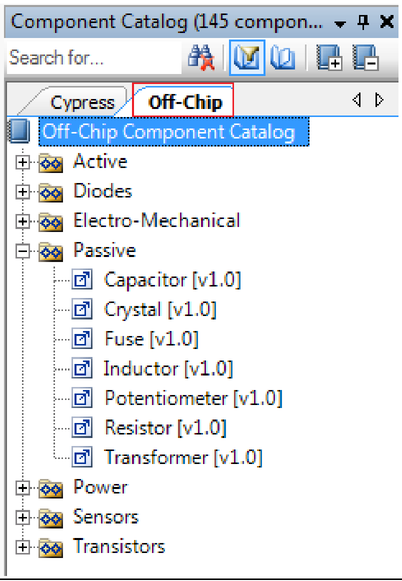


Figure 41 Off-chip Component Catalog

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.

6 GPIO pins in ModusToolbox™ software

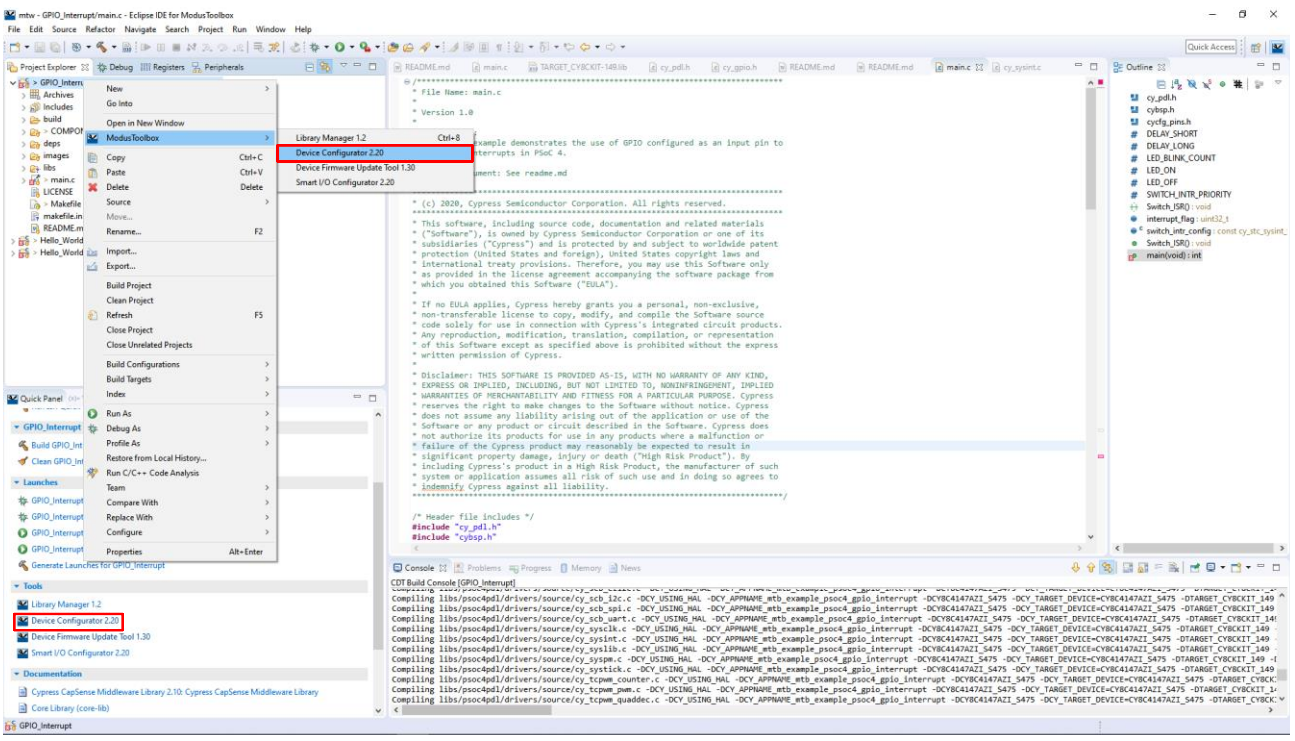
This section describes how to use ModusToolbox™ to configure and use GPIO pins.

6.1 Configuring GPIO pins using ModusToolbox™ Device Configurator

6.1.1 Using the Device Configurator

Initialization of GPIO pins can be done using two methods: using the Device Configurator as shown in this section or using GPIO using the peripheral driver library (PDL) in the next section. If you are familiar with PSoC™ Creator, the Device Configurator is similar to the configurations that can be made to components in PSoC™ Creator Top Design. The **Pins** tab in the Device Configurator allows the user to initialize GPIO pins and set the parameters for individual pins.

The Device Configurator is accessed by right-clicking on the project in ModusToolbox™ , selecting ModusToolbox™ and then Device Configurator 2.20. The Device Configurator can also be accessed in the ModusToolbox™ **Quick Panel** in the bottom left.

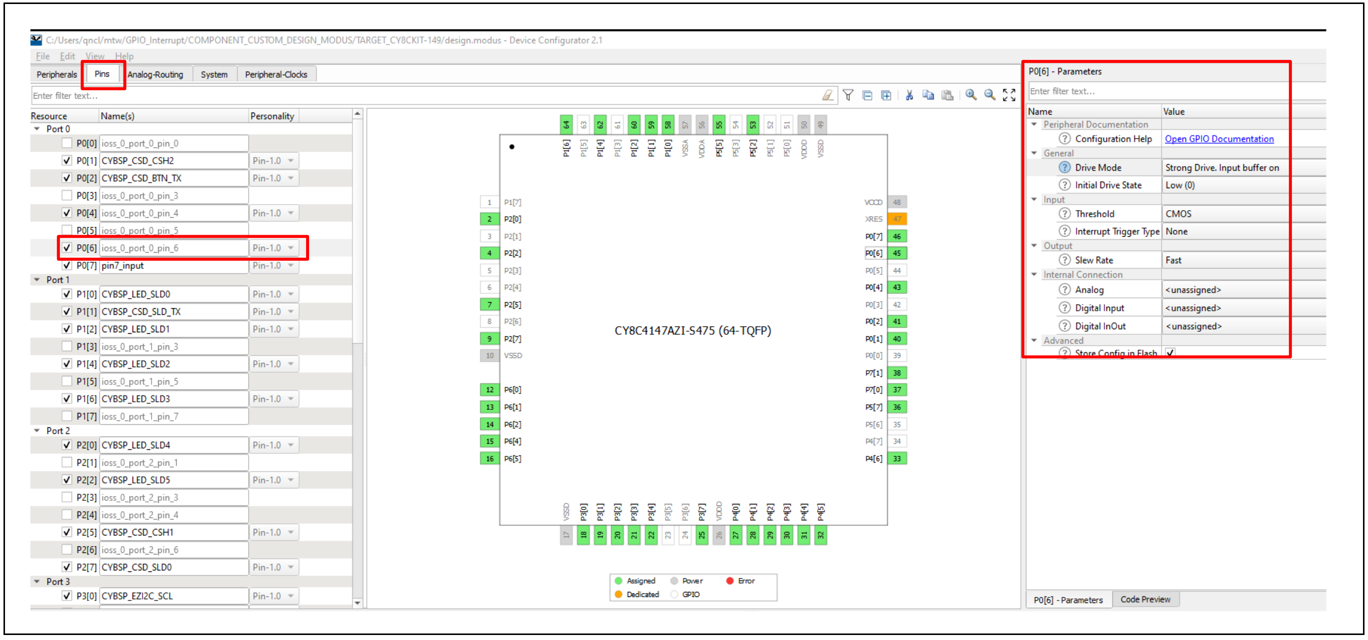


```
/*******************************************************************************
* File Name: main.c
*
* Version 1.0
*
          xample demonstrates the use of GPIO configured as an input pin to
          nterrupts in PSoC 4.

          ument: See readme.md

*******************************************************************************
* (c) 2020, Cypress Semiconductor Corporation. All rights reserved.
*******************************************************************************
* This software, including source code, documentation and related materials
* ("Software"), is owned by Cypress Semiconductor Corporation or one of its
* subsidiaries ("Cypress") and is protected by and subject to worldwide patent
* protection (United States and foreign), United States copyright laws and
* international treaty provisions. Therefore, you may use this Software only
* as provided in the license agreement accompanying the software package from
* which you obtained this Software ("EULA").
*
* If no EULA applies, Cypress hereby grants you a personal, non-exclusive,
* non-transferable license to copy, modify, and compile the Software source
* code solely for use in connection with Cypress's integrated circuit products.
* Any reproduction, modification, translation, compilation, or representation
* of this Software except as specified above is prohibited without the express
* written permission of Cypress.
*
* Disclaimer: THIS SOFTWARE IS PROVIDED AS-IS, WITH NO WARRANTY OF ANY KIND,
* EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, NONINFRINGEMENT, IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress
* reserves the right to make changes to the Software without notice. Cypress
* does not assume any liability arising out of the application or use of the
* Software or any product or circuit described in the Software. Cypress does
* not authorize its products for use in any products where a malfunction or
* failure of the Cypress product may reasonably be expected to result in
* significant property damage, injury or death ("High Risk Product"). By
* including Cypress's product in a High Risk Product, the manufacturer of such
* system or application assumes all risk of such use and in doing so agrees to
* indemnify Cypress against all liability.
*******************************************************************************/

/* Header file includes */
#include "cy_pdl.h"
#include "cybsp.h"
```

```
Compiling libs/psoc4pdl/drivers/source/cy_scb_i2c.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_149
Compiling libs/psoc4pdl/drivers/source/cy_scb_spi.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_149
Compiling libs/psoc4pdl/drivers/source/cy_scb_uart.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_14
Compiling libs/psoc4pdl/drivers/source/cy_sysclk.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_149
Compiling libs/psoc4pdl/drivers/source/cy_sysint.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_149
Compiling libs/psoc4pdl/drivers/source/cy_syslib.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_149
Compiling libs/psoc4pdl/drivers/source/cy_syspm.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_149 -I
Compiling libs/psoc4pdl/drivers/source/cy_systick.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_149
Compiling libs/psoc4pdl/drivers/source/cy_tcpwm_counter.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CK
Compiling libs/psoc4pdl/drivers/source/cy_tcpwm_pwm.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CKIT_1
Compiling libs/psoc4pdl/drivers/source/cy_tcpwm_quaddec.c -DCY_USING_HAL -DCY_APPNAME_mtb_example_psoc4_gpio_interrupt -DCY8C4147AZI_S475 -DCY_TARGET_DEVICE=CY8C4147AZI_S475 -DTARGET_CY8CK
```

Figure 42 Accessing the Device Configurator

The Pins section in the Device Configurator allows the configuration of individual GPIO pins on the device. The left panel shows the pins separated by their respective ports. To configure a pin, check the box next to the pin; this will create the initialization code. The parameters for the pin can be configured on the right of the Device Configuration window.


C:/Users/qncl/mtw/GPIO_Interrupt/COMPONENT_CUSTOM_DESIGN_MODUS/TARGET_CY8CKIT-149/design.modus - Device Configurator 2.1
File
Edit
View
Help
Peripherals
Pins
Analog-Routing
System
Peripheral-Clocks
CY8C4147AZI-S475 (64-TQFP)
Assigned
Power
Error
Dedicated
GPIO
P0[6] - Parameters
Code Preview

Resource	Name(s)	Personality
Port 0		
☐ P0[0]	ioss_0_port_0_pin_0	
✓ P0[1]	CYBSP_CSD_CSH2	Pin-1.0
✓ P0[2]	CYBSP_CSD_BTN_TX	Pin-1.0
☐ P0[3]	ioss_0_port_0_pin_3	
✓ P0[4]	ioss_0_port_0_pin_4	Pin-1.0
☐ P0[5]	ioss_0_port_0_pin_5	
✓ P0[6]	ioss_0_port_0_pin_6	Pin-1.0
✓ P0[7]	pin7_input	Pin-1.0
Port 1		
✓ P1[0]	CYBSP_LED_SLD0	Pin-1.0
✓ P1[1]	CYBSP_CSD_SLD_TX	Pin-1.0
✓ P1[2]	CYBSP_LED_SLD1	Pin-1.0
☐ P1[3]	ioss_0_port_1_pin_3	
✓ P1[4]	CYBSP_LED_SLD2	Pin-1.0
☐ P1[5]	ioss_0_port_1_pin_5	
✓ P1[6]	CYBSP_LED_SLD3	Pin-1.0
☐ P1[7]	ioss_0_port_1_pin_7	
Port 2		
✓ P2[0]	CYBSP_LED_SLD4	Pin-1.0
☐ P2[1]	ioss_0_port_2_pin_1	
✓ P2[2]	CYBSP_LED_SLD5	Pin-1.0
☐ P2[3]	ioss_0_port_2_pin_3	
☐ P2[4]	ioss_0_port_2_pin_4	
✓ P2[5]	CYBSP_CSD_CSH1	Pin-1.0
☐ P2[6]	ioss_0_port_2_pin_6	
✓ P2[7]	CYBSP_CSD_SLD0	Pin-1.0
Port 3		
✓ P3[0]	CYBSP_EZI2C_SCL	Pin-1.0

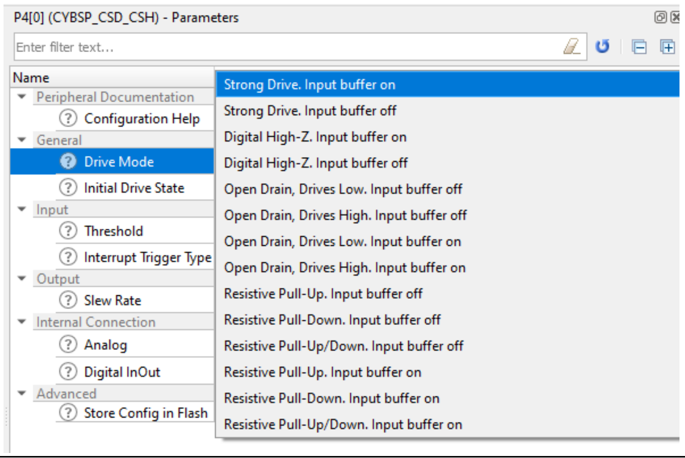
Name	Value
Peripheral Documentation	
Configuration Help	Open GPIO Documentation
General	
Drive Mode	Strong Drive. Input buffer on
Initial Drive State	Low (0)
Input	
Threshold	CMOS
Interrupt Trigger Type	None
Output	
Slew Rate	Fast
Internal Connection	
Analog	<unassigned>
Digital Input	<unassigned>
Digital InOut	<unassigned>
Advanced	
Store Config in Flash	✓

Figure 43 Device Configurator

Table 5 Pin component settings

Setting	Description
General > Drive Mode	This setting configures the pin with one of the eight drive modes as shown in Figure 44. For a more detailed description of these drive modes, see the GPIO pin basics section.
General > 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, and the pin is set with an appropriate drive mode, as shown in Figure 44. If the pin is configured as an input, the initial drive state can still be useful. For example, if resistive pull-up is required at the input pin, 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 > Threshold	CMOS and LVTTL input threshold setting is for an entire port. Note that an error will be displayed in the Notice List of the Device Configurator if all the pins on a port are not configured the same way. For more information, see the Pins datasheet.
Input > Interrupt Trigger Type	This setting configures the GPIO Edge Detect block described in the GPIO interrupt. For more details on interrupts, see the AN90799 – PSoC™ 4 Interrupts.
Output > Slew Rate	The slew rate parameter determines the rise and fall ramp rate for the pin as it changes output logic levels. For more details on this setting, see Physical structure of GPIO pins.
Internal Connection > Analog	This setting allows connecting of the pin to an analog signal.
Internal Connection > Digital Output	This setting allows connecting to a digital output signal.
Internal Connection > Digital InOut	This setting allows connecting to a digital input signal. Input signals are primarily used for I^{2}C interfaces.

Setting	Description
Advanced > Store Config in Flash	This setting controls whether the configuration structure is stored in flash (const, true) or SRAM (not const, false).


P4[0] (CYBSP_CSD_CSH) - Parameters

Enter filter text...

Name
- Peripheral Documentation
  - Configuration Help
- General
  - Drive Mode
  - Initial Drive State
- Input
  - Threshold
  - Interrupt Trigger Type
- Output
  - Slew Rate
- Internal Connection
  - Analog
  - Digital InOut
- Advanced
  - Store Config in Flash

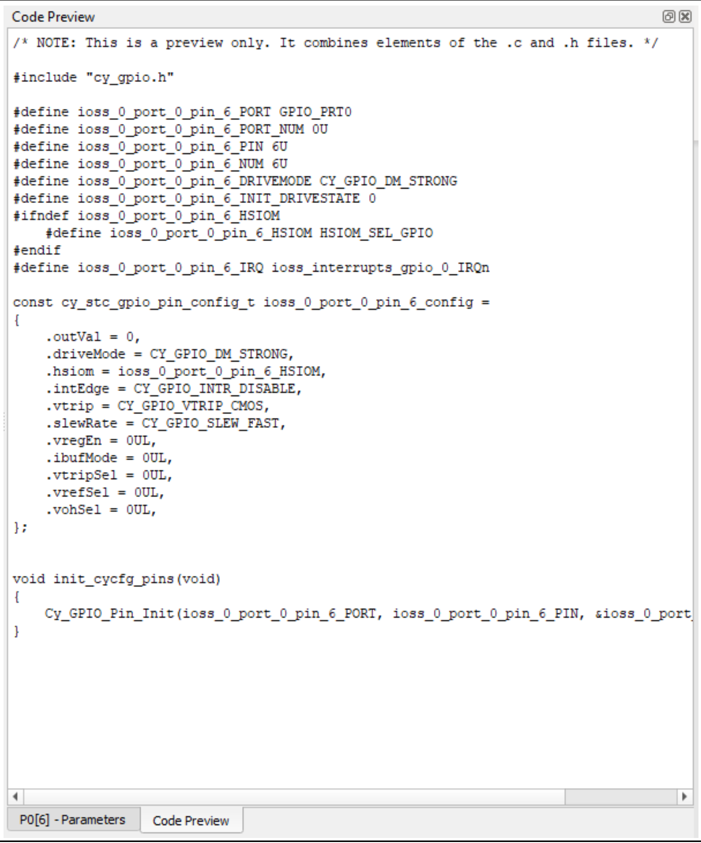
Strong Drive. Input buffer on
Strong Drive. Input buffer off
Digital High-Z. Input buffer on
Digital High-Z. Input buffer off
Open Drain, Drives Low. Input buffer off
Open Drain, Drives High. Input buffer off
Open Drain, Drives Low. Input buffer on
Open Drain, Drives High. Input buffer on
Resistive Pull-Up. Input buffer off
Resistive Pull-Down. Input buffer off
Resistive Pull-Up/Down. Input buffer off
Resistive Pull-Up. Input buffer on
Resistive Pull-Down. Input buffer on
Resistive Pull-Up/Down. Input buffer on

Figure 44 **Drive modes**

The Device Configurator sets up configurations of the pins, peripherals, and other device configurations. When closing the Device Configurator, go to **File** > **Save** to save the updates to the current project. After configuring the resources using the Device Configurator, PDL is used to interact with the pins and any interrupts. See the GPIO using Peripheral Driver Library (PDL) section and the PSoC™ 4 PDL API reference for more information on using GPIO pins after initialization.

Note: *The name of a GPIO pin can be changed from the default name provided by the BSP. This is helpful for naming the pins to correspond to their purpose in your design. The name of a pin can be changed by selecting the pin and typing the custom name into the text input in the "Name(s)" column as shown in Figure 43.*

6.1.2 Device Configurator code preview



```
/* NOTE: This is a preview only. It combines elements of the .c and .h files. */

#include "cy_gpio.h"

#define ioss_0_port_0_pin_6_PORT GPIO_PRT0
#define ioss_0_port_0_pin_6_PORT_NUM 0U
#define ioss_0_port_0_pin_6_PIN 6U
#define ioss_0_port_0_pin_6_NUM 6U
#define ioss_0_port_0_pin_6_DRIVEMODE CY_GPIO_DM_STRONG
#define ioss_0_port_0_pin_6_INIT_DRIVESTATE 0
#ifndef ioss_0_port_0_pin_6_HSIOM
    #define ioss_0_port_0_pin_6_HSIOM HSIOM_SEL_GPIO
#endif
#define ioss_0_port_0_pin_6_IRQ ioss_interrupts_gpio_0_IRQn

const cy_stc_gpio_pin_config_t ioss_0_port_0_pin_6_config =
{
    .outVal = 0,
    .driveMode = CY_GPIO_DM_STRONG,
    .hsiom = ioss_0_port_0_pin_6_HSIOM,
    .intEdge = CY_GPIO_INTR_DISABLE,
    .vtrip = CY_GPIO_VTRIP_CMOS,
    .slewRate = CY_GPIO_SLEW_FAST,
    .vregEn = 0UL,
    .ibufMode = 0UL,
    .vtripSel = 0UL,
    .vrefSel = 0UL,
    .vohSel = 0UL,
};


void init_cycfg_pins(void)
{
    Cy_GPIO_Pin_Init(ioss_0_port_0_pin_6_PORT, ioss_0_port_0_pin_6_PIN, &ioss_0_port
}
```

Figure 45 Device Configuration code preview

The Device Configurator features a Code Preview window that displays the PDL defines and functions that are created to configure the GPIO pin as shown in Figure 45. When the Device Configurator settings are saved, this code is automatically added to your project. This code can also be manually added to your projected by copying the definitions from this window.

6.2 GPIO using the peripheral driver library (PDL)

The PDL integrates device header files, startup code, and peripheral drivers into a single package. The PDL supports the PSoC™ 4 Series device family. The drivers abstract the hardware functions into a set of easy-to-use APIs. These are fully documented in the PSoC™ 4 PDL API reference. The PDL reduces the need to understand register usage and bit structures, thus easing software development for the extensive set of peripherals in the PSoC™ 4 series. You configure the driver for your application, and then use API function calls to initialize and use the peripheral.

The peripheral driver library (PDL) GPIO functions and other declarations used in this driver are located in *cy_gpio.h*. You can optionally include *cy_pdl.h* (ModusToolbox™ only) to get access to all the functions and declarations in the PDL.

Initialization can be performed either at the port level or by configuring the individual pins. See the product device header files for the list of supported ports and pins. It is recommended to use the Device Configurator for initialization of pins and other device resources. This reduces the risk of improper initialization and overflow of resources.

Single pin configuration is performed by using `Cy_GPIO_Pin_FastInit` (provide specific argument values) or `Cy_GPIO_Pin_Init` (provide a filled `cy_stc_gpio_pin_config_t` structure).

An entire port can be configured using `Cy_GPIO_Port_Init` by providing a filled `cy_stc_gpio_prt_config_t` structure. The values in the structure are bit fields representing the desired value for each pin in the port.

For the following example, the names defined in the device configurator can be used. For "base" use "`user_defined_name`"`_PORT` and for the "`pin_num`" use "`user_defined_name`"**_NUM.** The user-defined name can be easily configured and viewed by using the ModusToolbox™ Device Configurator.

6.2.1 GPIO pin initialization-full

Individual GPIO initialization starts with defining all pin configuration values in the format as shown below. This initialization is not needed if a pin is already configured using the ModusToolbox™ Device Configurator.

Code Listing 1 GPIO pin configuration structure

```
cy_stc_gpio_pin_config_t pinConfig = {
/*.outVal */ 1UL,                  /* Output = High */
/*.driveMode */ CY_GPIO_DM_PULLUP, /* Resistive pull-up, input buffer on */
/*.hsiom */ P0_3_GPIO,             /* Software controlled pin */
/*.intEdge */ CY_GPIO_INTR_RISING, /* Rising edge interrupt */
/*.vtrip */ CY_GPIO_VTRIP_CMOS,    /* CMOS voltage trip */
/*.slewRate */ CY_GPIO_SLEW_FAST,  /* Fast slew rate */
/*.vregEn */ 0UL,                  /* SIO-specific setting - ignored */
/*.ibufMode */ 0UL,                /* SIO-specific setting - ignored */
/*.vtripSel */ 0UL,                /* SIO-specific setting - ignored */
/*.vrefSel */ 0UL,                 /* SIO-specific setting - ignored */
/*.vohSel */ 0UL                   /* SIO-specific setting - ignored */
};
```

This pin configuration defines all the parameters for a GPIO pin. This is the third argument when initializing a pin.

To initialize a pin use Cy_GPIO_Pin_Init `(base,pinNum,config)` with the following arguments as shown below:

base	Pointer to the pin's port register base address
pinNum	Position of the pin bit field within the port register
config	Pointer to the pin config structure base address

Code Listing 2 Single GPIO pin initialization-full example

```
/* Initialize pin P0.3 */
if(CY_GPIO_SUCCESS != Cy_GPIO_Pin_Init(P0_3_PORT, P0_3_NUM, &pinConfig))
{
     /* Insert error handling */
}
```

6.2.2 GPIO pin initialization- fast

The GPIO Pin fast initialization Initializes the most common configuration settings for all pin types. These include, drive mode, initial output value, and HSIOM connection. Initialize a pin with `CY_GPIO_PIN_FastInit(base, pinNum, driveMode, outVal, hsiom)` using the following arguments as shown below This initialization is not needed if a pin is already configured using the ModusToolbox™ Device Configurator.

base	Pointer to the Pin's Port register base address
pinNum	Position of the pin bit field within the Port register
driveMode	Pin drive mode. Options are detailed in Pin drive mode macros.
outVal	Logic state of the output buffer driving the pin (1 or 0)
hsiom	HSIOM (High-Speed input output multiplexer) input selection

Code Listing 3 Single GPIO pin initialization- fast example

```
/* Quickly initialize pin P0.3 (e.g. quickly set up a test LED) */
Cy_GPIO_Pin_FastInit(P0_3_PORT, P0_3_NUM, CY_GPIO_DM_PULLUP, 1UL, P0_3_GPIO);
```

6.2.3 GPIO port initialization

Initialize a complete port of pins from a single init structure.

The configuration structure used in this function has a 1:1 mapping to the GPIO and HSIOM registers shown below. See the device technical reference manual (TRM) for the register details on how to populate them.

Code Listing 4 GPIO port configuration structure example

```
cy_stc_gpio_prt_config_t portConfig = {
/*.dr                  =*/ 0x00000008u,      /* PX.3 output = 1 */
/*.intrCfg           =*/ 0x00000080u,      /* PX.3 rising edge interrupt */
/*.pc                  =*/ 0x00000400u,      /* PX.3 resistive pull-up */
/*.pc2               =*/ 0x00000000u,      /* PX.3 input buffer on */
/*.sio                 =*/ 0x00000000u,      /* PX[7:0] ignored */
/*.selActive      =*/ 0x00000000u,      /* PX[7:0] software controlled */
};

/* Initialize GPIO port 0 */
if(CY_GPIO_SUCCESS != Cy_GPIO_Port_Init(GPIO_PRT0, &portConfig))
{
/* Insert error handling */
}
```

6.2.4 Reading from a GPIO pin

Reading from a GPIO pin is the same if a pin is configured using the Device Configurator or PDL. The port and pin are arguments when using the `CY_GPIO_Read` function as shown in below.

Code Listing 5 Reading GPIO pin

```
/* Scenario: P0.3 was initialized and input buffer enabled */
/* Read the input state of P0.3 */
if(1UL == Cy_GPIO_Read(P0_3_PORT, P0_3_NUM))
{
/* Insert logic for High pin state */
}
else
{
/* Insert logic for Low pin state */
}
```

6.2.5 Writing to a GPIO pin

Writing a value to a GPIO pin is the same if a pin is configured using the Device Configurator or PDL. The port and the pin are arguments when using the `CY_GPIO_Write` function as shown below.

Code Listing 6 Writing to GPIO pin

```
uint32_t pinState = 0UL;
/* Control P0.3 based on the pinState variable */
Cy_GPIO_Write(P0_3_PORT, P0_3_NUM, pinState);
```

6.2.6 GPIO interrupt

The GPIO interrupt for a pin is configured using one of two methods: using the Device Configurator as seen in Figure 43, or using PDL. If the device configurator is used to set up the pin, the interrupt type can be selected in the side panel as shown in Figure 43. The interrupt type can also be configured in the GPIO PDL configuration structure. See the PSoC™ 4 PDL API reference for how to configure the interrupt trigger type.

For more details of using Interrupts with PSoC™ 4 S series devices in ModusToolbox™, see the CE230654 - PSoC™ 4: GPIO interrupt code example on GitHub. This code example can also be found in the New Application Wizard in ModusToolbox™; for more information, see the ModusToolbox™ code examples section.

7 GPIO tips and tricks in PSoC™ Creator

This section provides practical examples of how to use GPIO pins when using PSoC™ Creator.

Table 6 PSoC™ Creator projects

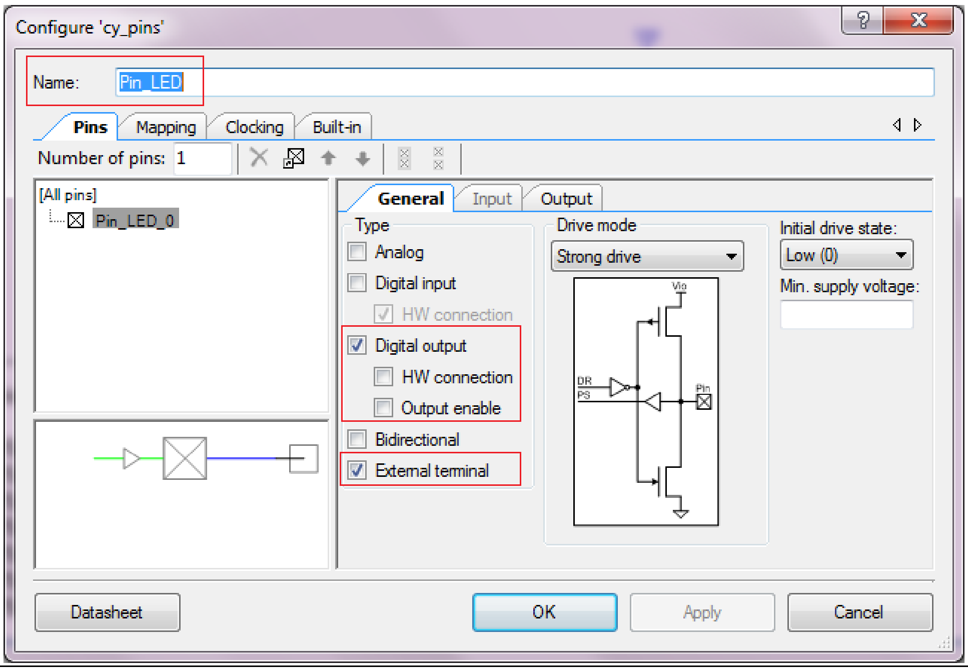
#	Section	PSoC™ 4000, 4000S, 4100S, 4100S Plus, 4100PS, 4500S, 4700S	PSoC™ 4100_BLE, 4100, 4100M	PSoC™ 4200, 4200M, 4200L, 4200-BLE
1	Toggle an LED	✓	✓	✓
2	Read an input and write to an output	✓	✓	✓
3	Drive an output from a digital logic gate			✓
4	Using a bidirectional pin		✓	✓
5	Set the GPIO input/output synchronization		✓	✓
6	Toggle GPIOs faster with Data	✓	✓	✓
7	Configure GPIO output enable logic			✓
8	Pin interrupt	✓	✓	✓
9	Configure GPIO interrupt settings with firmware	✓	✓	✓
10	Using both analog and digital on a GPIO		✓	✓
11	Gang pins for more drive/sink current			✓
12	Control register handling in Deep Sleep			✓

The Infineon development kits, listed in PSoC™ 4 development boards, can be used for testing these projects.

7.1 Toggle an LED

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

1. Place a Digital Output Pin Component in the project schematic.
2. Name the Component “Pin_LED” and disable the hardware connection.


Configure 'cy_pins'

Name: Pin_LED

Number of pins: 1

Type: [✓ Digital output] [✓ External terminal]

Drive mode: Strong drive

Initial drive state: Low (0)

Figure 46 **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.

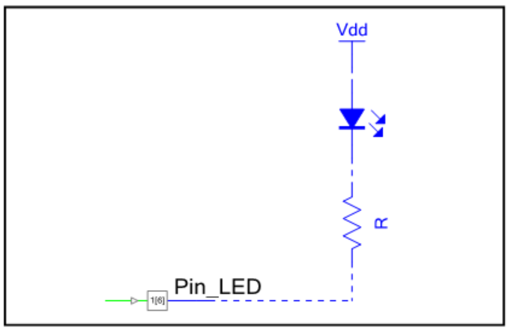


Figure 47 **Toggle an LED example schematic**

6. In *main.c*, use the Component API functions 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);
}
```

7. Build the project and program the PSoC™ 4 device.

The result is an LED blinking at a frequency of 1 Hz.

7.2 Read an input and write to an output

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

1. Place two pins in the project schematic—one digital input pin and one digital output pin with hardware connection disabled.

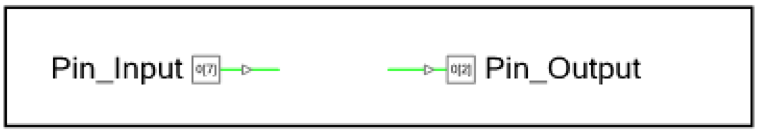


Figure 48 Input and output example schematic

2. Assign the pins for Pin_Input and Pin_Output in the .cydwr window.
3. 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());
     }
```

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

You can test this project by feeding a square wave from a signal generator to Pin_Input. The signal at Pin_Output will be an inverted form of the signal at Pin_Input.

7.3 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 a NOT gate and the output of the NOT gate is routed to another pin. Follow these steps to create the project:

1. Place two pins in the project schematic—one Digital Input Pin and one Digital Output Pin with hardware connection enabled.

Pin_Input
Pin_Output

Figure 49 Input and output pins with HW connection enabled

2. Place a NOT gate and connect to the pins as shown in Figure 50.

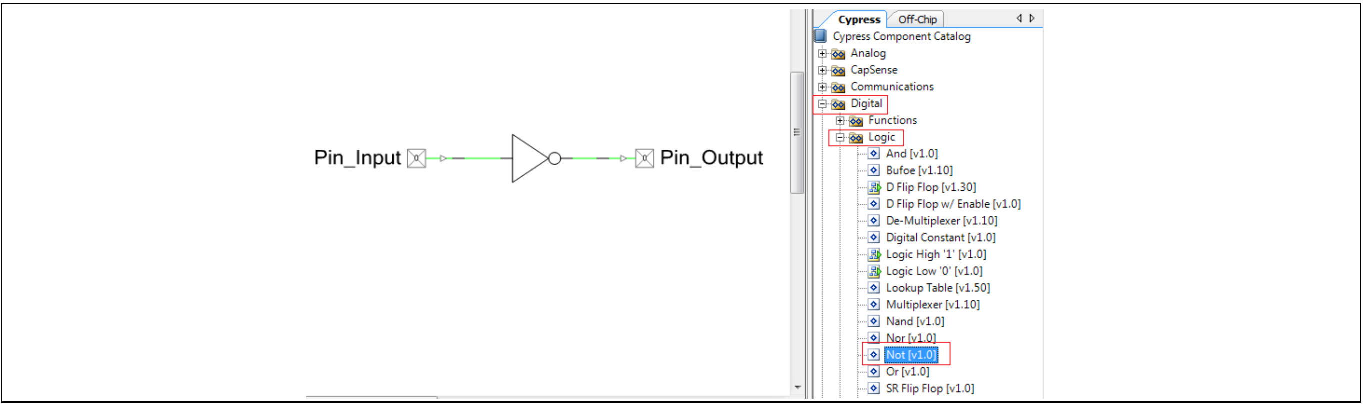


Figure 50 NOT gate connection

3. Assign the pins for Pin_Input and Pin_Output in the .cydwr window.
4. This project does not require any code. Build the project and program the PSoC™ 4 device.
5. Similar to the previous project, you can test this project by feeding a square wave from a signal generator to Pin_Input. The signal at Pin_Output will be an inverted form of the signal at Pin_Input.

7.4 Using a bidirectional pin

This example demonstrates the use of a pin in the bidirectional mode, that is, with both digital input and digital output active. PSoC™ Creator provides a Pin Component in bidirectional configuration - "Digital Bidirectional Pin".

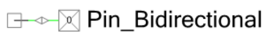
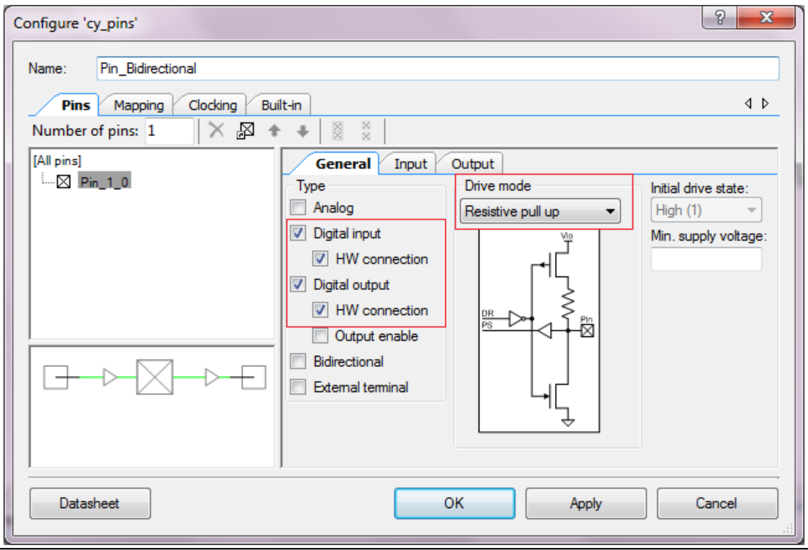


Figure 51 Pin_Bidrectional

This Pin Component, however, shows a single terminal for both input and output. Its use is limited to the I^{2}C SDA and SCL lines. In many applications, it is useful to have two terminals – one for input and one for output. This can be done by enabling both the Digital Input and Digital Output options in the Pin Component customizer. An example is shown for configuring such a pin, where a switch is connected at the input side to pull the pin to logic LOW. This pin is configured to resistive pull-up with logic '1' driven continuously. To check the bidirectional pin status, the pin signal is routed to another pin. Follow the steps below to create the project:

1. place a digital input pin on the schematic.
2. Enable digital output with Drive mode as resistive pull up:


Configure 'cy_pins'

Name: Pin_Bidirectional

Pins | Mapping | Clocking | Built-in

Number of pins: 1

[All pins]
Pin_1_0

General | Input | Output

Type
- [ ] Analog
- [✓] Digital input
  - [✓] HW connection
- [✓] Digital output
  - [✓] HW connection
  - [ ] Output enable
- [ ] Bidirectional
- [ ] External terminal

Drive mode: Resistive pull up

Initial drive state: High (1)

Min. supply voltage:

Figure 52 Pin_Bidirectional configuration

3. Ensure that the Component looks like as follows on the schematic.

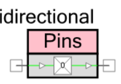


Figure 53 Pin_Bidirectional on the schematic

4. Connect a logic HIGH to drive the pin with resistive pull-up continuously.

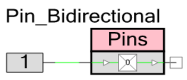


Figure 54 Logic HIGH to Pin_Bidirectional

Now, the pin status is seen using another pin (Pin_Status) connected at the input side.

5. Place a Digital Output Pin and connect at the input buffer side of the Pin_Bidirectional.

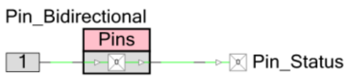


Figure 55 Pin_Status connection to Pin_Bidirectional

With the external connections enabled, schematic looks like as follows:

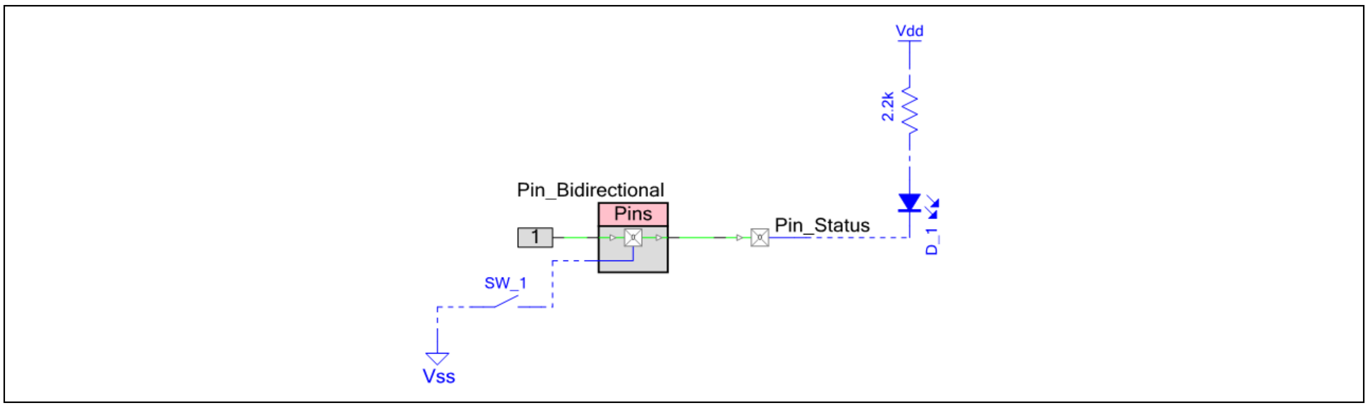


Figure 56 Complete schematic

6. Assign the pins in the .cydwr window.

The project does not require any user code. Build the project and program the PSoC™ 4 device.

The LED connected to Pin_Status shows the input buffer state of Pin_Bidirectional. When the switch is not pressed, logic '1' drives Pin_Bidirectional with a resistive pull-up. This turns OFF the LED (as the LED is connected in active LOW mode). When the switch is pressed, strong logic '0' appears at Pin_Bidirectional. This turns ON the LED. Thus, this project demonstrates two drivers on the same pin (Pin_Bidirectional) – one internal (logic 1), and other external (switch) with an input.

7.5 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 show in Figure 57 and Figure 58.

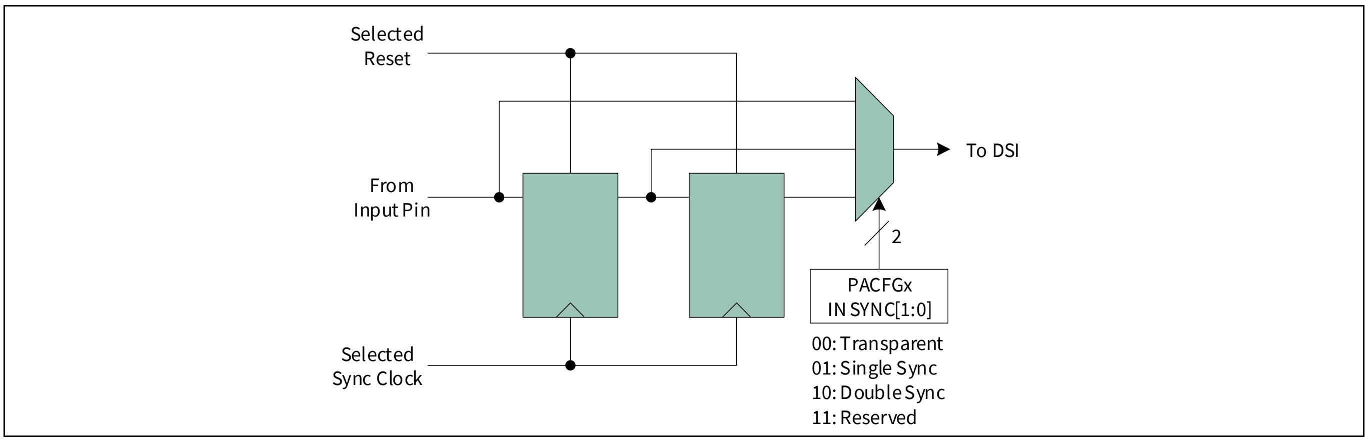


Figure 57 Input synchronization in PSoC™ 4

As shown in Figure 57, the input synchronization circuit provides the options of transparent, single sync, and double sync.

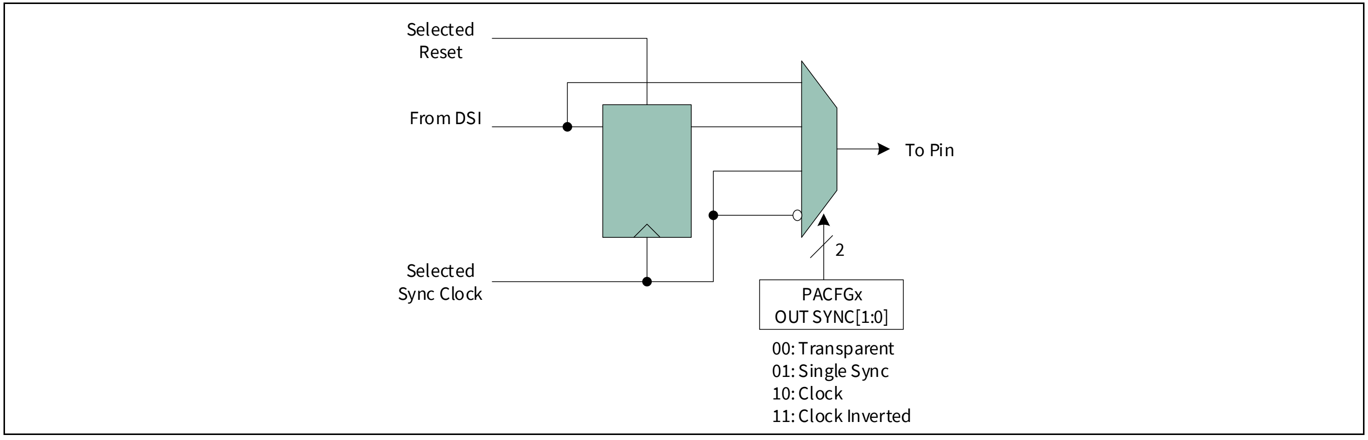
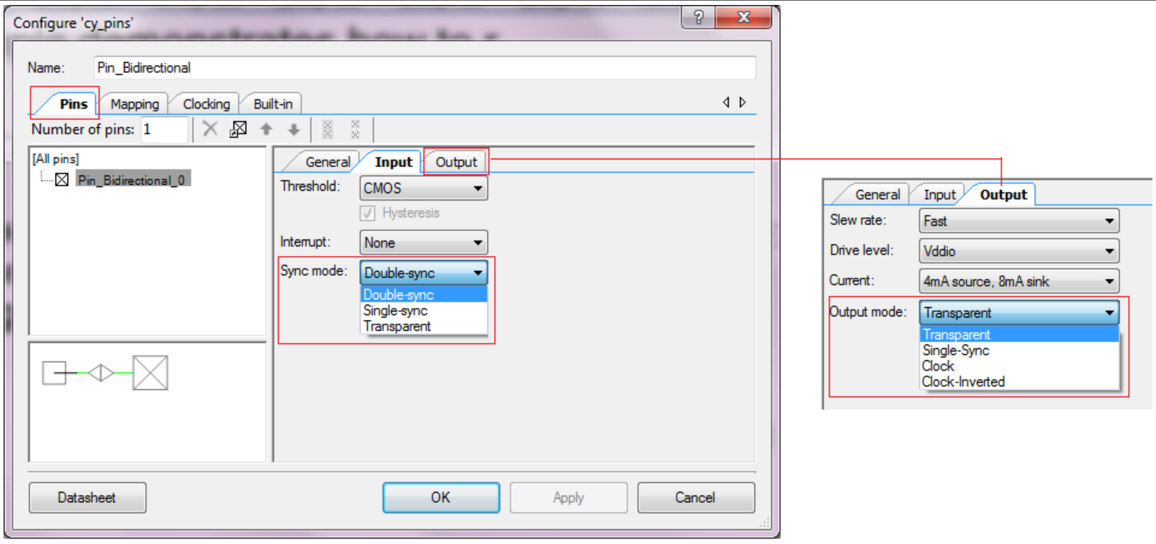


Figure 58 **Output synchronization in PSoC™ 4**

The output synchronization circuit provides the options of transparent, single sync, clock, and clock inverted, as shown in Figure 58. Clock and clock inverted route the sync clock to the output pin. These are set in the Pins Component customizer, as shown in Figure 59.

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 Clocks tab, as shown in Figure 60.

For more information on the parameters in the Clocking tab of the pins component customizer, see the Pins component datasheet.


Configure 'cy_pins'

Name: Pin_Bidirectional

Number of pins: 1

Threshold: CMOS

Hysteresis

Interrupt: None

Sync mode: Double-sync

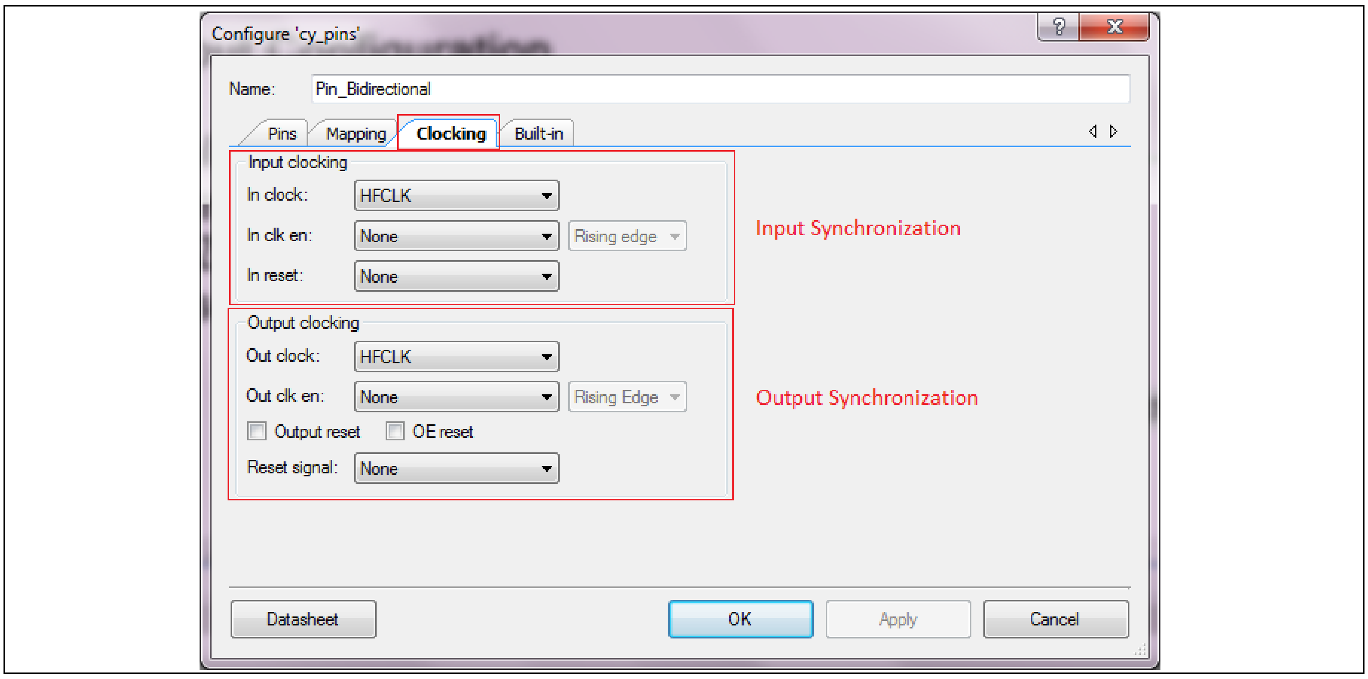
Slew rate: Fast

Drive level: Vddio

Current: 4mA source, 8mA sink

Output mode: Transparent

Figure 59 **Sync mode setting**


Configure 'cy_pins'

Name: Pin_Bidirectional

Input clocking
In clock: HFCLK
In clk en: None | Rising edge
In reset: None

Output clocking
Out clock: HFCLK
Out clk en: None | Rising Edge
Output reset: ☐ | OE reset: ☐
Reset signal: None

Figure 60 Clock setting

The pin signals are synchronized using a combination of the UDB port adapter and GPIO blocks. Also, the clock is common for all the pins of a port for internal clock synchronization. For more information, see the PSoC™ 4 architecture reference manual.

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.

7.5.1 GPIO input synchronization

1. Place one digital input pin, two digital output pins, and one clock component in the project schematic. Configure them as follows:

Table 7 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

2. Connect the pins and add the off-chip components:

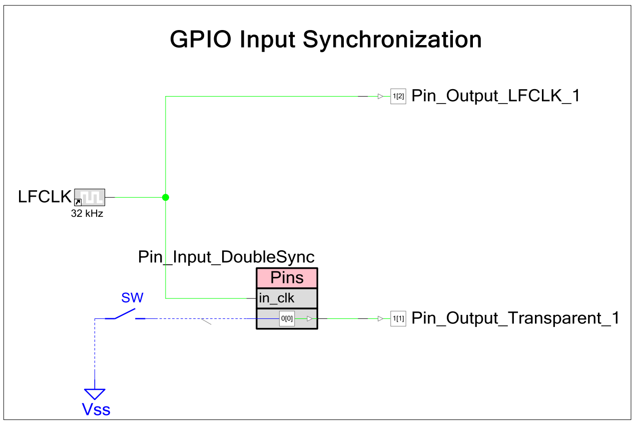


Figure 61 GPIO input synchronization example schematic

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 reference manual 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.
5. When the button connected to Pin_Input_DoubleSync is pressed, the signal waveforms occur. The Pin_Output_Transparent_1 pin becomes LOW at the second rising edge of LFCLK because the input is double synchronized with LFCLK.

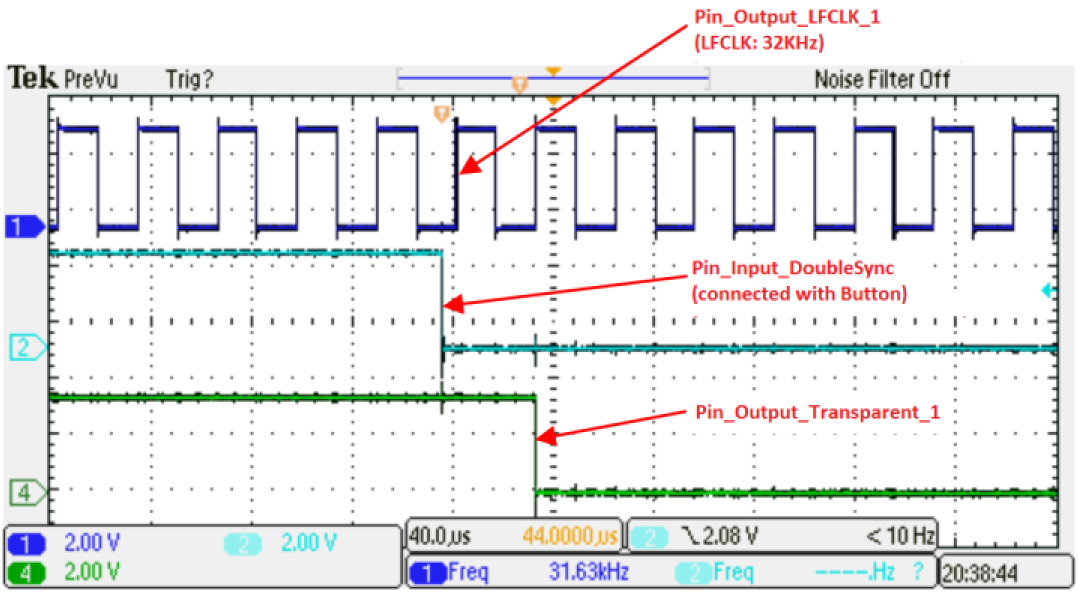


Figure 62 Input/output signal waveforms

7.5.2 GPIO output synchronization

1. Place one digital input pin, three digital output pins, and one Clock Component in the project schematic, and configure them per Table 8.

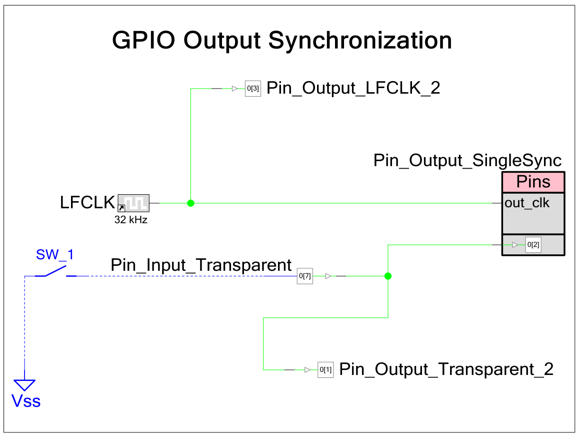


Figure 63 GPIO output synchronization example schematic

Table 8 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

2. Connect the pins as shown in Figure 63.
3. 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 do not support synchronization.
4. Build the project and program the PSoC™ 4 device.

Figure 64 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.

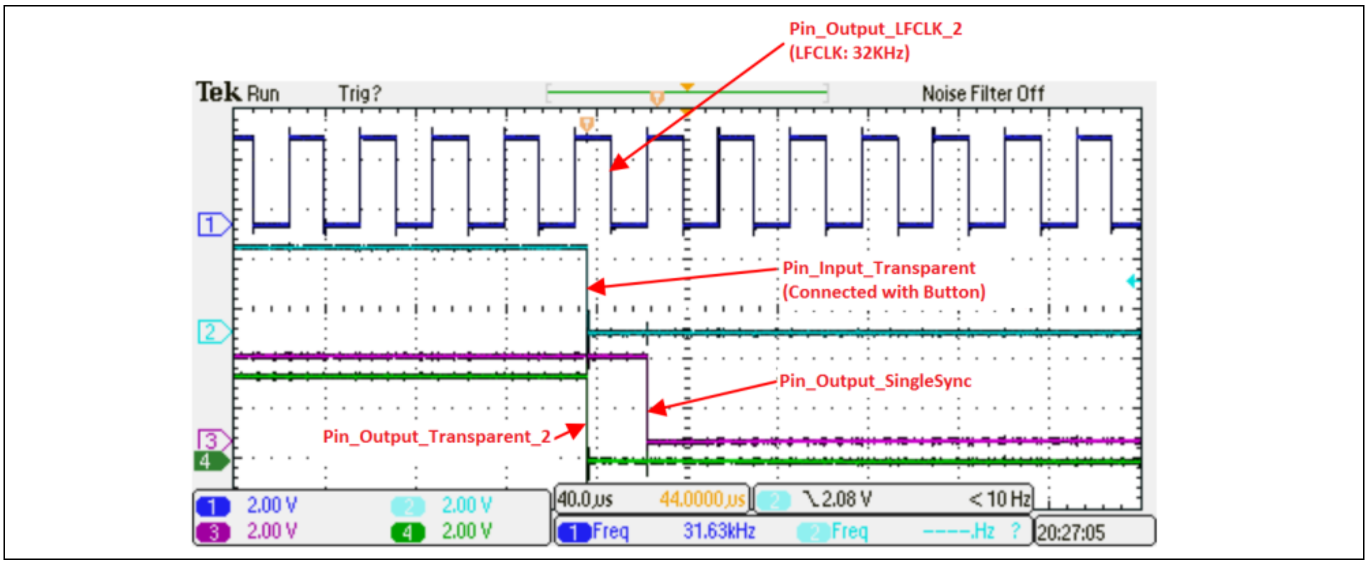


Figure 64 **Input/output signal waveform**

7.6 Toggle GPIOs faster with Data registers

Calling the component API functions 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 the 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 function.

The component API function has 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**
`CY_SYS_PINS_SET_PIN(portDR, pin)`	Sets the output value for the pin to logic HIGH `portDR` is the address of the port data register `pin` is the pin number (0 to 7)
`CY_SYS_PINS_CLEAR_PIN(portDR, pin)`	Clears the output value for the pin to logic LOW `portDR` is the address of the port data register `pin` is the pin number (0 to 7)
`CY_SYS_PINS_READ_PIN(portPS, pin)`	Reads the pin value `portPS` is the address of the port status register `pin` is the pin number (0 to 7)

See *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,".

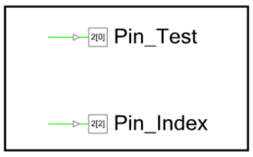


Figure 65 Toggle GPIOs with Data registers example schematic

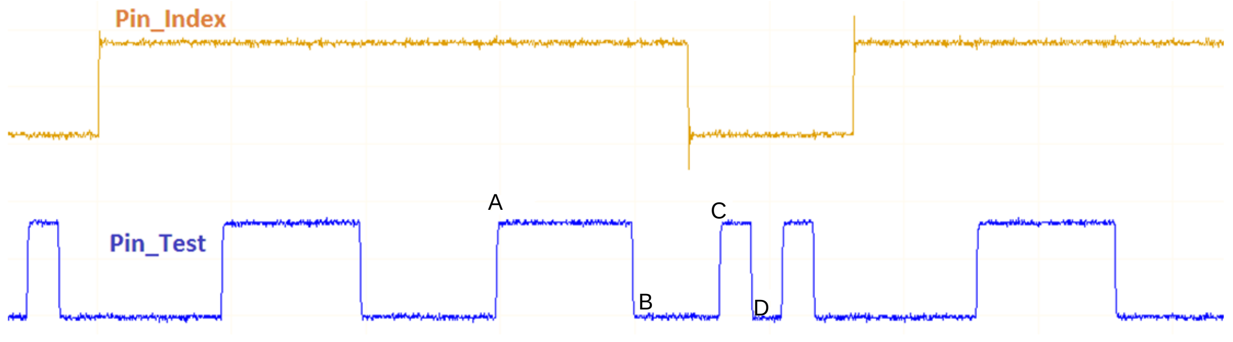
2. Assign the pins in the .cydwr window.
3. Add the following code to the *main.c* file. The code sets Pin_Index HIGH and toggles Pin_Test using Component API functions. It then sets Pin_Index LOW and toggles Pin_Test using the Data register (DR).

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


        /* 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 folder of the project workspace (in this case, Pin_Test.h) to know about the macro for the data register address.*

The code that writes to the data register is also portable similar to the API function 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.



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 66 Output signals waveform

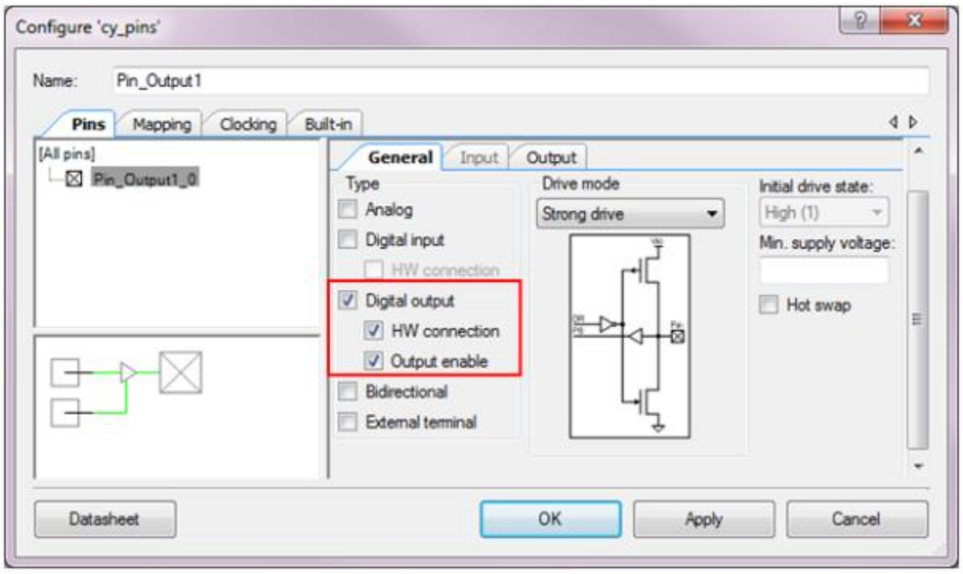
Figure 66 shows that a pin can toggle faster by directly writing to the data register, as opposed to calling API functions.

To know about coding techniques to achieve time efficiency, see the application note, AN89610 – PSoC™ 4 and PSoC™ 5LP Arm® Cortex® code optimization.

7.7 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™ 4200 Bluetooth® LE, PSoC™ 4200M, and PSoC™ 4200L 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.


Configure 'cy_pins'

Name: Pin_Output1

Pins | Mapping | Clocking | Built-in

[All pins]
Pin_Output1_0

General | Input | Output

Type
- [ ] Analog
- [ ] Digital input
  - [ ] HW connection
- [x] Digital output
  - [x] HW connection
  - [x] Output enable
- [ ] Bidirectional
- [ ] External terminal

Drive mode: Strong drive

Initial drive state: High (1)

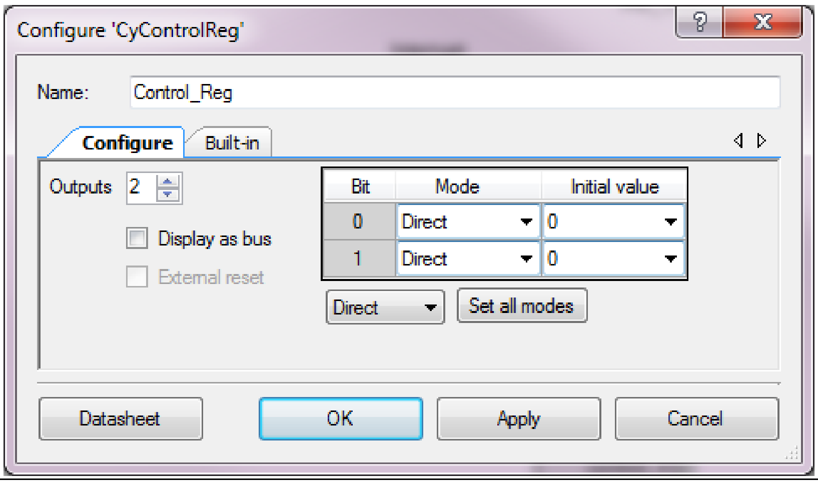
Min. supply voltage:

- [ ] Hot swap

Datasheet | OK | Apply | Cancel

Figure 67 Output enable selection

3. Place a Control register in the schematic.
4. Configure the Control register for two outputs.


Configure 'CyControlReg'

Name: Control_Reg

Configure | Built-in

Outputs: 2

- [ ] Display as bus
- [ ] External reset

Bit	Mode	Initial value
0	Direct	0
1	Direct	0

Direct | Set all modes

Datasheet | OK | Apply | Cancel

Figure 68 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.

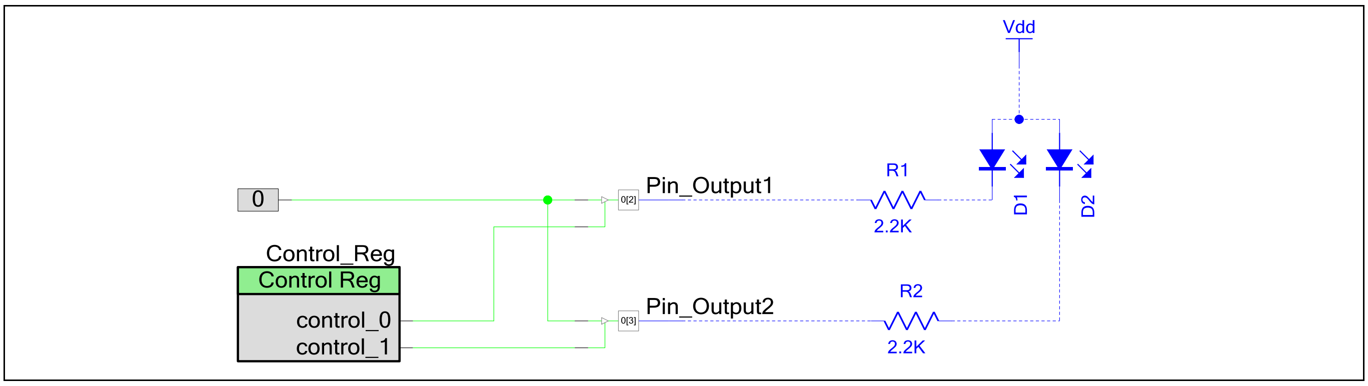


Figure 69 Control register driving pins' output enable

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.

7.8 Pin interrupt

This example demonstrates how to use an interrupt generated from two pins in the same port, using Component API functions. 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.

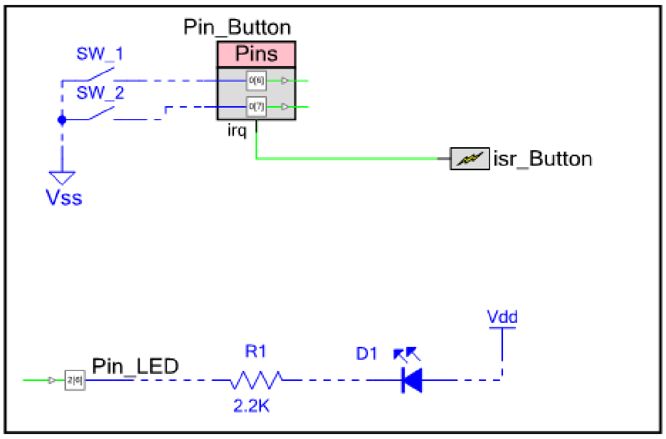


Figure 70 **Pin interrupt example schematic**

4. Assign the pins in the .cywdr window.
5. Use the Component API functions 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 more 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.

Note: *Not all ports have dedicated interrupts. For higher ports, a common interrupt signal is generated.*

See the "Interrupts" chapter in the respective device Architecture reference manual

For more information on interrupts and writing interrupt handlers, see AN90799 – PSoC™ 4 interrupts.

7.9 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 per the following table. You can change the configuration at any time to enable or disable pin interrupts.

Table 9 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:

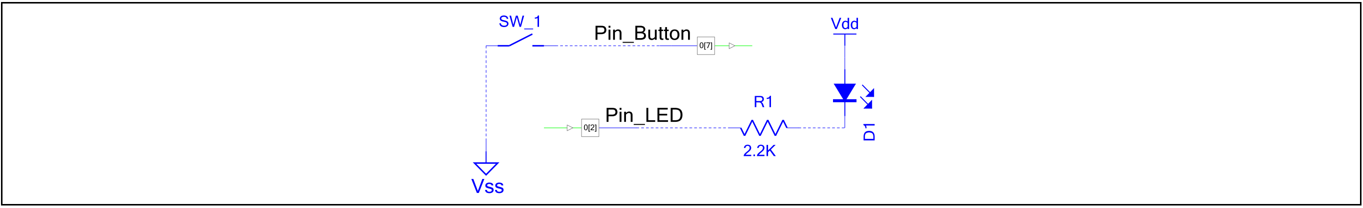


Figure 71 Example schematic

2. Assign the pins to Pin_Button and Pin_LED in the cydwr window.
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 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 reference manual contains more information about the GPIO interrupt, including block diagrams and functional descriptions. Another good resource is the application note AN90799 – PSoC™ 4 interrupts.

7.10 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 firmware, the LED blinks. When controlled by the IDAC, the LED gradually brightens.

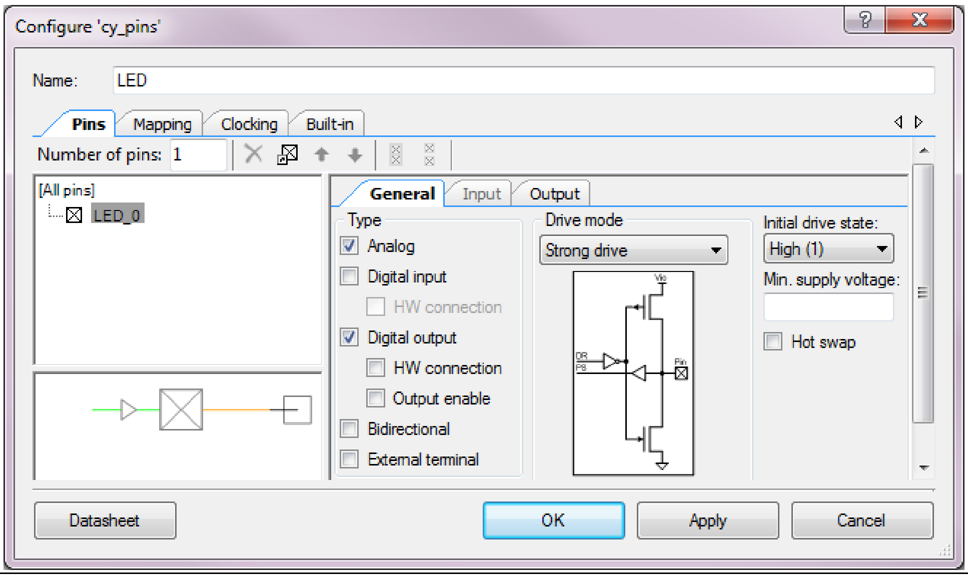
This type of multiplexing is useful when you need analog and digital functionality from a single pin. It can also reduce the number of GPIOs used in a design.

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.


Configure 'cy_pins'

Name: LED

Pins | Mapping | Clocking | Built-in

Number of pins: 1

[All pins]
LED_0

General | Input | Output

Type
- [✓] Analog
- [ ] Digital input
- [ ] HW connection
- [✓] Digital output
- [ ] HW connection
- [ ] Output enable
- [ ] Bidirectional
- [ ] External terminal

Drive mode: Strong drive

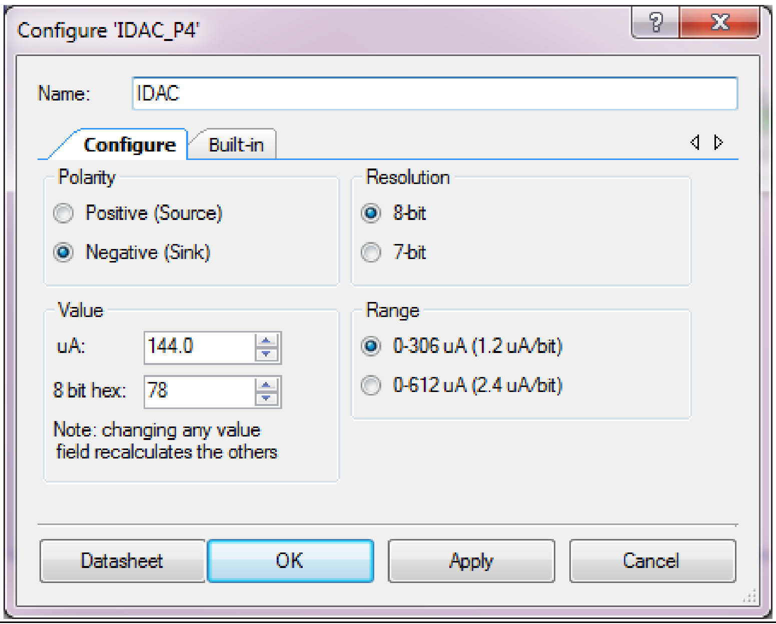
Initial drive state: High (1)

Min. supply voltage:

- [ ] Hot swap

Figure 72 LED pin configured as both analog and digital

4. Set the IDAC's **Polarity** as **Negative (Sink)**, as shown in Figure 73. Connect the IDAC to the analog terminal, as shown in Figure 74.


Configure 'IDAC_P4'

Name: IDAC

Configure | Built-in

Polarity: Positive (Source) [✓ Negative (Sink)]

Resolution: [✓ 8-bit] 7-bit

Value
uA: 144.0
8 bit hex: 78
Note: changing any value field recalculates the others

Range: [✓ 0-306 uA (1.2 uA/bit)] 0-612 uA (2.4 uA/bit)

Figure 73 **IDAC setting**

5. Build the project to create the necessary APIs.

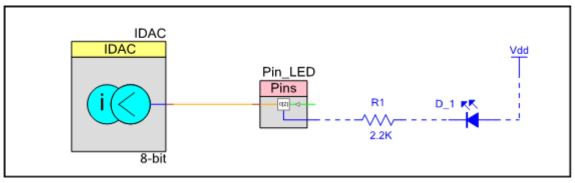


Figure 74 **PSoC™ Creator schematic of analog and digital switching scheme**

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 reference manual for more details.

7.11 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, PSoC™ 4200M, and PSoC™ 4200L 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.

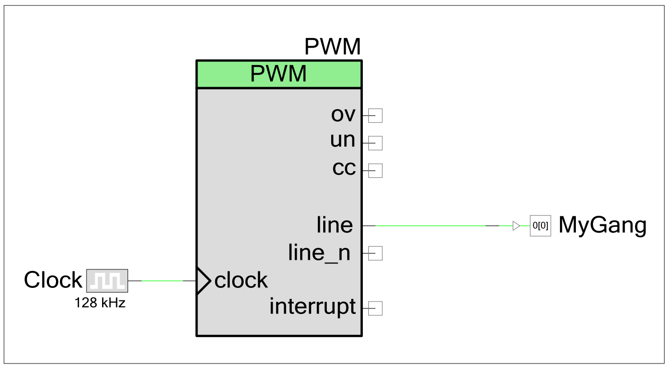
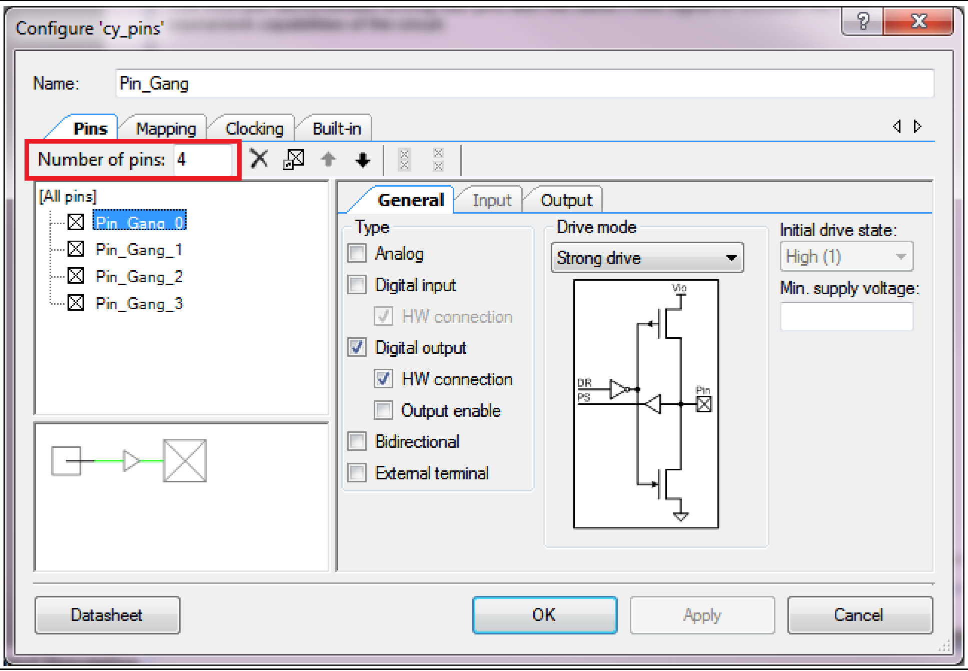


Figure 75 PWM driven to single pin

4. Open the pins configuration dialog and set the number of pins accordingly. This example uses four GPIO pins. Set the **Output Mode** to **Single-Sync** and **Out Clock** to **External**.

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



Configure 'cy_pins'

Name: Pin_Gang

Number of pins: 4

[All pins]
Pin_Gang_0
Pin_Gang_1
Pin_Gang_2
Pin_Gang_3

General

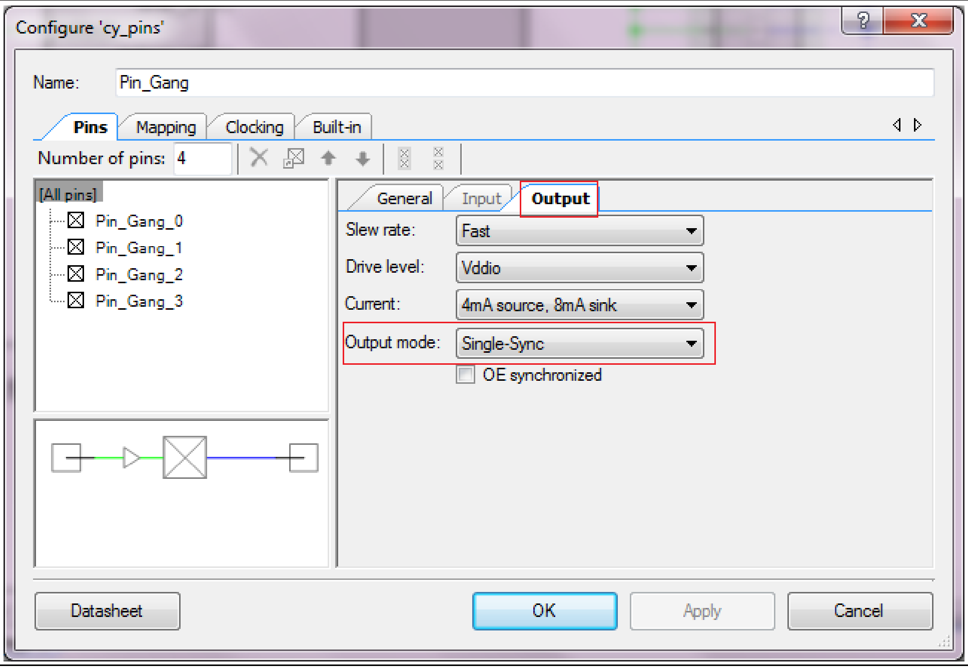
Type
- [ ] Analog
- [ ] Digital input
- [✓] HW connection
- [✓] Digital output
- [✓] HW connection
- [ ] Output enable
- [ ] Bidirectional
- [ ] External terminal

Drive mode: Strong drive

Initial drive state: High (1)

Min. supply voltage:

Figure 76 **Configure multiple pins in the component**



Configure 'cy_pins'

Name: Pin_Gang

Number of pins: 4

[All pins]
Pin_Gang_0
Pin_Gang_1
Pin_Gang_2
Pin_Gang_3

Output

Slew rate: Fast

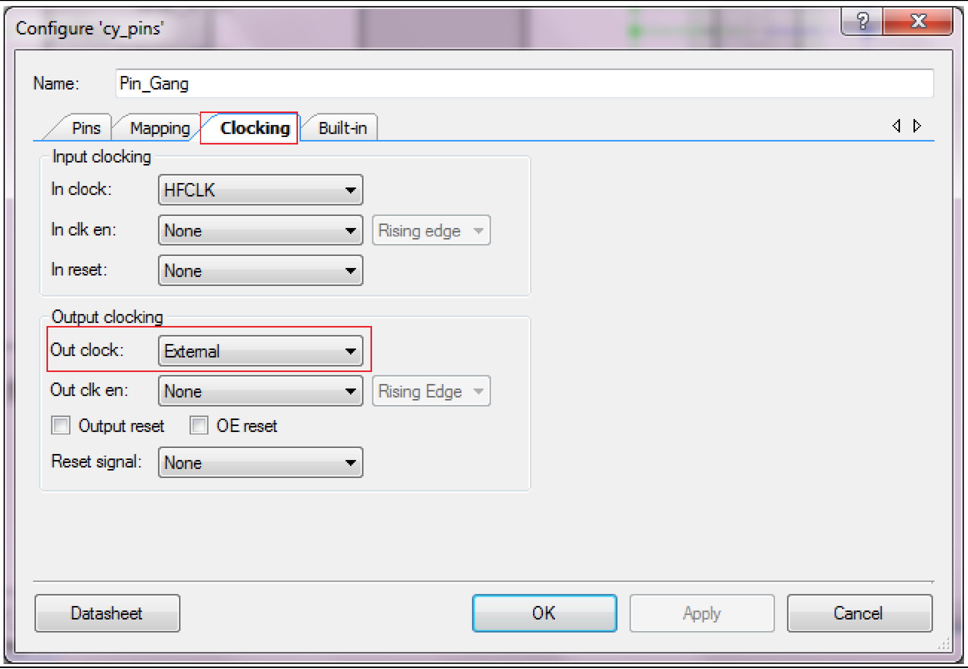
Drive level: Vddio

Current: 4mA source, 8mA sink

Output mode: Single-Sync

- [ ] OE synchronized

Figure 77 **Output mode setting**


Configure 'cy_pins'

Name: Pin_Gang

Tabs: Pins | Mapping | **Clocking** | Built-in

Input clocking

In clock: HFCLK

In clk en: None   Rising edge

In reset: None

Output clocking

Out clock: External

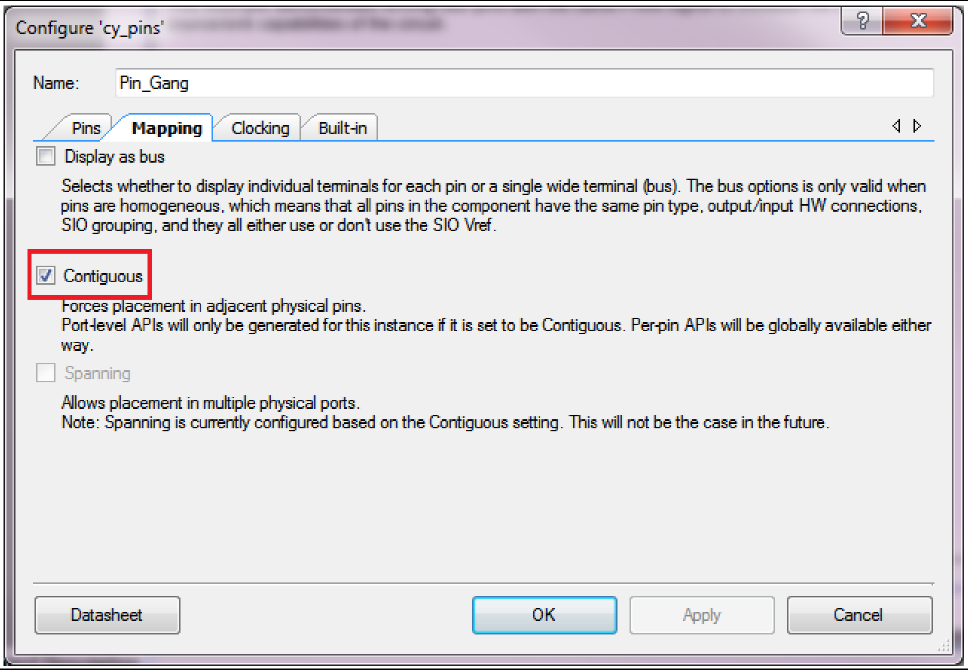
Out clk en: None   Rising Edge

☐ Output reset   ☐ OE reset

Reset signal: None

Figure 78 Out clock setting

5. (Optional) Set the pin mapping to **Contiguous** for easier PCB routing.


Configure 'cy_pins'

Name: Pin_Gang

Tabs: Pins | **Mapping** | Clocking | Built-in

☐ Display as bus

Selects whether to display individual terminals for each pin or a single wide terminal (bus). The bus options is only valid when pins are homogeneous, which means that all pins in the component have the same pin type, output/input HW connections, SIO grouping, and they all either use or don't use the SIO Vref.

☑ Contiguous

Forces placement in adjacent physical pins.
Port-level APIs will only be generated for this instance if it is set to be Contiguous. Per-pin APIs will be globally available either way.

☐ Spanning

Allows placement in multiple physical ports.
Note: Spanning is currently configured based on the Contiguous setting. This will not be the case in the future.

Figure 79 Enable contiguous mapping

6. Assign the Pins component to physical pins.
7. 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.

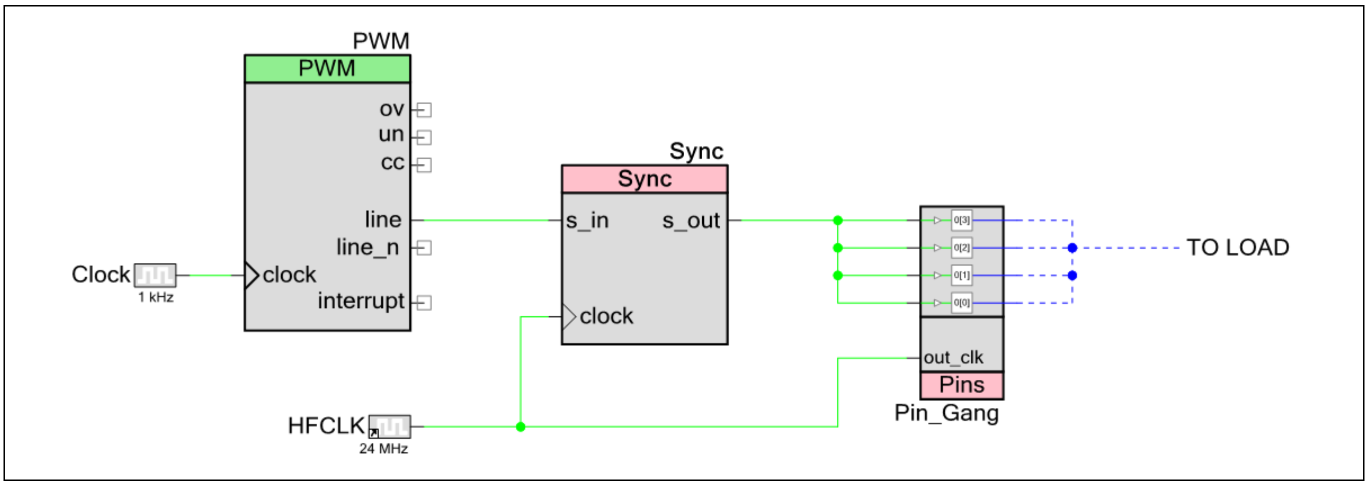


Figure 80 PWM driving four pins

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

7.12 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 to freeze and unfreeze the GPIOs using the `CySysPmFreezeIo()` and `CySysPmUnfreezeIo()` API functions. Follow these steps to create the PSoC™ Creator project. Note that this project is applicable only for PSoC™ 4200, PSoC™ 42xx_BL, PSoC™ 4200M, and PSoC™ 4200L.

1. Place one digital input pin, two digital output pins, a Clock, a Control register, and an interrupt component in the schematic.
2. Configure the components as shown in Table 10. Connect the components as shown in Figure 81.

Table 10 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_Buttton	Default configuration
Control register	Ctrl_Reg	Output: 1

Component	Name	Configuration
		Initial value: 1

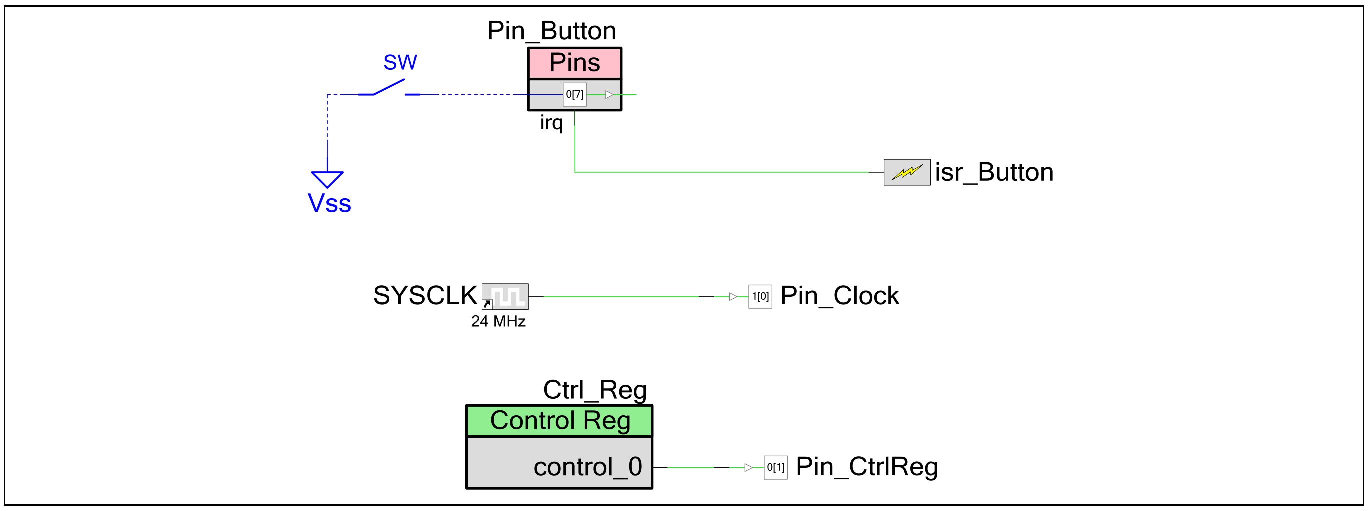


Figure 81 Avoiding glitch while exiting Deep Sleep

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
                }

                /* 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 shown in Figure 82. 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 shown in Figure 83.

Note: Not all ports have dedicated interrupts. For higher ports, a common interrupt signal is generated.

See the "Interrupts" chapter in the respective device PSoC™ 4 architecture reference manual.

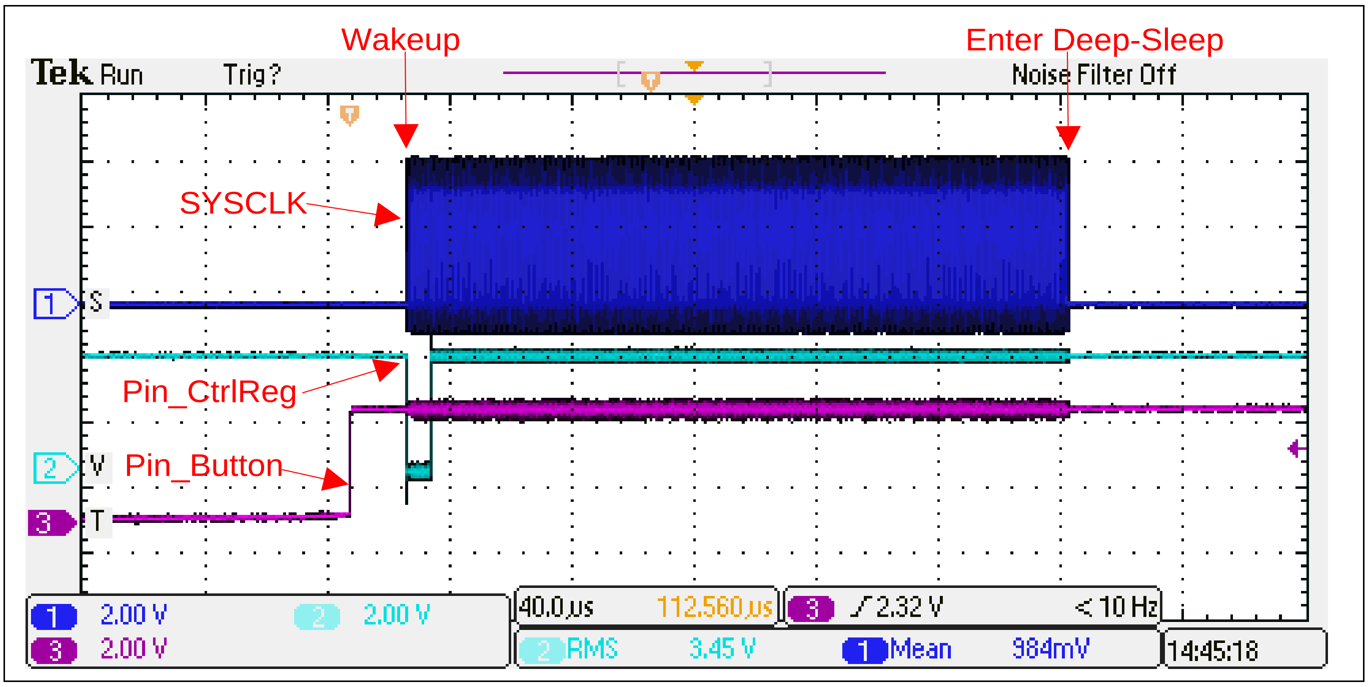


Figure 82 **Output signal waveform (no freeze I/O)**

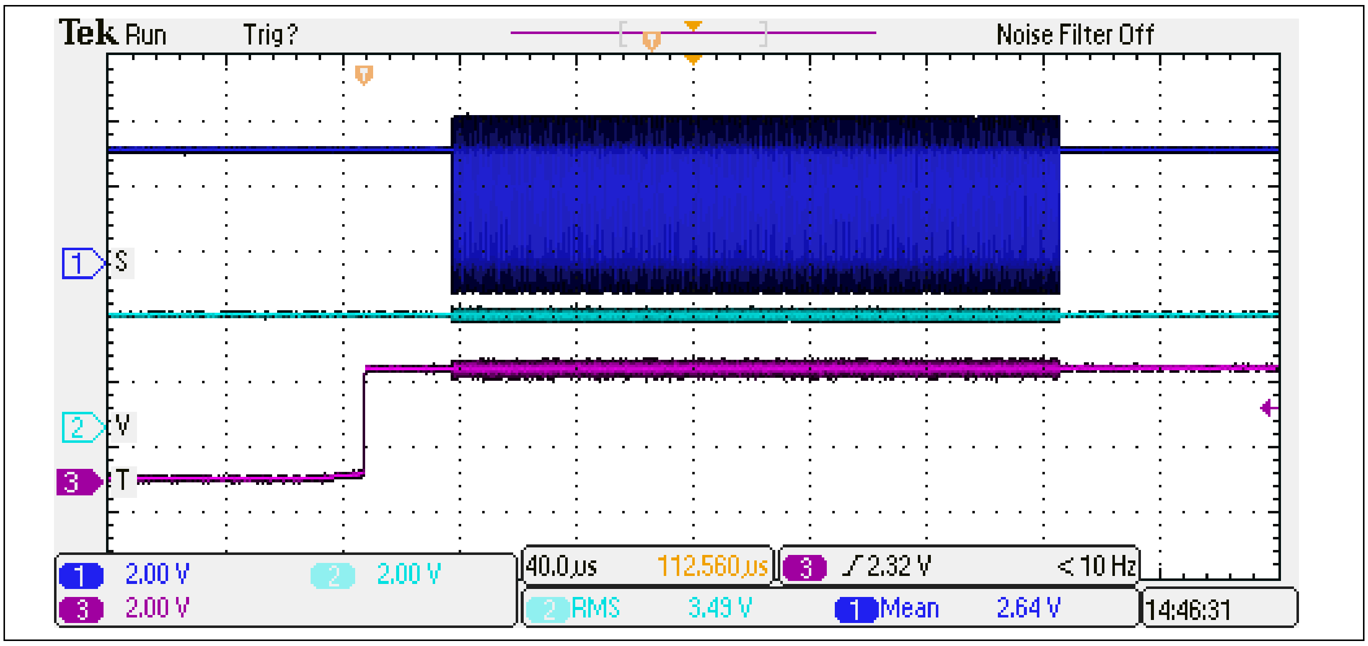


Figure 83 **Output signal waveform (freeze I/O)**

8 GPIO tips and tricks in ModusToolbox™ software

This section provides practical examples of how to use GPIO pins when using ModusToolbox™.

8.1 Read an input and write to an output

Reading from an input and writing to an output are done using GPIO PDL functions.

PSoC™ 4 code examples can be accessed in the ModusToolbox™ new application wizard. Code example CE231741 demonstrates multiple methods of configuring, reading, writing, and generating interrupts with PSoC™ 4 General Purpose Input/Output (GPIO) pins.

8.2 Pin interrupt

For a code example of a GPIO interrupt project in ModusToolbox™, see the CE230654 - PSoC™ 4: GPIO interrupt code example. This code example can also be accessed in the ModusToolbox™ new application wizard.

8.3 More code examples

More code examples for PSoC™ 4 developed with ModusToolbox™ that cover a wide array of topics can be found on the Infineon GitHub or in ModusToolbox™. See the ModusToolbox™ code examples section for more information on accessing code examples.

9 Related application notes

- AN79953 – Getting started with PSoC™ 4 MCU
- AN86233 – PSoC™ 4 low-power modes and power reduction techniques
- AN72382 – Using PSoC™ 3 and PSoC™ 5LP GPIO pins
- AN90799 – PSoC™ 4 interrupts
- AN2094 – PSoC™ 1 getting started with GPIO
- AN89610 – PSoC™ Arm® Cortex® code optimization

10 PSoC™ 4 GPIO compared to PSoC™ 1, PSoC™ 3, and PSoC™ 5LP GPIO

The PSoC™ 4 GPIO are different from that of PSoC™ 1, PSoC™ 3, and PSoC™ 5LP; see Table 11 for details.

Table 11 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.

11 PSoC™ 4 development boards

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

Device Family	Development board
PSoC™ 4000	CY8CKIT-040 PSoC™ 4000 Pioneer Development Kit
PSoC™ 4000S / 4100S	CY8CKIT-041 PSOC™ 4 S-Seriesm Pioneer Kit
PSoC™ 4100S Plus	CY8CKIT-149 PSoC™ 4100S Plus prototyping Kit
PSoC™ 4100PS	CY8CKIT-147 PSoC™ 4100PS prototyping Kit
PSoC™ 4200 / PSoC™ 4100	CY8CKIT-042 PSoC™ 4 Pioneer Kit
PSoC™ 42x7_BL	CY8CKIT-042-BLE-A PSoC™ 4 Bluetooth® Low Energy Pioneer Kit
PSoC™ 4200M	CY8CKIT-044 PSoC™ 4 M-Series Pioneer Kit
PSoC™ 4200L	CY8CKIT-046 PSoC™ 4 L-Series Pioneer Kit
PSoC™ 4200DS	PSoC™ 4 CY8CKIT-146 4200DS Prototyping Kits
PSoC™ 4700S	CY8CKIT-148 PSoC™ 4700S Inductive Sensing Evaluation Kit

References

The wealth of information available on the Infineon webpage can help you select the right PSoC™ device and, additionally, integrate the device into your designs efficiently and effectively. The following is an abbreviated list for PSoC™ 4:

- Overview: PSoC™ portfolio
- Product selectors: PSoC™ 4. In addition, PSoC™ Creator includes a device selection tool.
- Datasheets describe and provide electrical specifications for each family.
- Application notes cover a broad range of topics, from basic to advanced level, and include the following:
  - AN88619: PSoC™ 4 MCU hardware design considerations
  - AN73854: PSoC™ Creator - Introduction to bootloaders
  - AN89610: PSoC™ Arm® Cortex® code optimization
  - AN86233: PSoC™ 4 MCU low-power modes and power reduction techniques
  - AN57821: Mixed-signal circuit board layout considerations
  - AN89056: PSoC™ 4 - IEC 60730 class B and IEC 61508 SIL Safety Software Library
  - AN64846: Getting started with CAPSENSE™
  - AN85951: PSoC™ 4 and PSoC™ 6 MCU CAPSENSE™ design guide
- Code examples demonstrate product features and usage
- Reference manuals: Provide detailed descriptions of the architecture and registers in each PSoC™ 4 device family
- PSoC™ 4 programming specification provides the information necessary to program PSoC™ 4 nonvolatile memory.
- Development tools:
  - CY8CKIT-040, CY8CKIT-042, CY8CKIT-044, CY8CKIT-046, CY8CKIT-042-BLE, CY8CKIT-045S, and CY8CKIT-041S-MAX PSoC™ 4 Pioneer kits are easy-to-use and inexpensive development platforms. These include connectors for Arduino-compatible shields and Digilent Pmod daughter cards.
  - CY8CKIT-043, CY8CKIT-145-40XX, CY8CKIT-147, and CY8CKIT-149 are very low-cost prototyping platforms for sampling PSoC™ 4 devices.
  - CY8CKIT-040T is a low-cost evaluation kit showing the low power CAPSENSE, low power wake on touch and liquid tolerant features of the PSoC™ 4000T device.
  - The MiniProg3 (CY8CKIT-002) or MiniProg4 (CY8CKIT-005)kit provides an interface for flash programming and debug.
  - Integrated Development Environment (IDE): There are two development platforms that can be used for application development with PSoC™ 4 – ModusToolbox™ and PSoC™ Creator.
  - PSoC™ 4 CAD libraries provide footprint and schematic support for common tools. IBIS models are also available.
- Training videos are available in Infineon website on a wide range of topics including the PSoC™ MCUs
- Infineon community enables connection with fellow PSoC™ developers around the world, 24 hours a day, 7 days a week, and hosts a dedicated PSoC™ 4 MCU community.

Revision history

Document revision	Date	Description of changes
**	2014-03-27	New application note
*A	2014-05-16	Updated for PSoC™ 4000
*B	2015-07-02	Updated for PSoC™ 4 Bluetooth® LE 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
*C	2016-01-05	Updated for PSoC™ 4 L-Series Added Figure 17. PSoC™ 4200L analog routing diagram Added section 7.4 to introduce Bidirectional Pin Added an example project “Project04_BidirectionanlPin” Updated Table 6 Updated the projects to PSoC™ Creator 3.3
*D	2016-03-22	Updated for PSoC™ 4000S, PSoC™ 4100S and PSoC™ analog coprocessor Added Figure 12 Added Figure 18 Updated Table 6 Updated Appendix B: PSoC™ 4 development boards
*E	2017-04-19	Updated logo and copyright
*F	2017-12-13	Added support to PSoC™ 4100S Plus Modified Figure 9, Figure 10, and Figure 14. Added reference to TRM in section 3.4.1, 7.8 and 7.12. Modified Section 4.
*G	2018-08-27	Updated for PSoC™ 4100PS
*H	2018-09-17	Updated Section 3.1 with respect to 1.8V CMOS Updated note in section 3.3 on reset provision via pin P1[6] in PSoC™ 4000 Updated Table 4 to include information on wake up from low power modes Updated components in the associated projects
*I	2018-11-28	Corrected section 4 on the higher voltage limit that the OVT pin can withstand
*J	2020-10-16	Added in support for ModusToolbox™ and PDL for PSoC™ 4 Added Chapter 6 GPIO pins in ModusToolbox™ Added Chapter 8 GPIO tips and tricks in ModusToolbox™
*K	2020-12-17	Added in references to CE231741 in section 8.1.
*L	2023-02-04	Updated note in Section 3.2.1
*M	2024-03-01	Updated broken links. Moved section PSoC™ resources to References.

Edition 2024-03-01

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

001-86439 Rev.*M