PSOC™ Automotive 4 MCU - Using GPIO pins

About this document

Scope and purpose

This application note explains how to effectively use PSOC™ Automotive 4 MCU GPIO pins with various use case examples to demonstrate their features. 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™ Automotive 4 MCU GPIO pins.

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Introduction . . . 4
2 ModusToolbox™ . . . 5
2.1 ModusToolbox™ code examples . . . 5
2.2 Technical support . . . 7
3 GPIO pin basics . . . 8
3.1 Physical structure of GPIO pins . . . 8
3.2 Pin routing . . . 12
3.2.1 Digital routing . . . 12
3.2.2 Analog routing . . . 14
3.3 Start-up and low-power behavior . . . 20
3.4 GPIO interrupt . . . 22
3.4.1 Limitations in GPIO interrupt . . . 23
3.5 Smart I/O . . . 24
3.5.1 Block components . . . 24
3.5.2 Routing . . . 25
3.5.3 Operation . . . 25
4 Overvoltage-tolerant (OVT) pins . . . 27
5 GPIO pins in ModusToolbox™ software . . . 28
5.1 Configuring GPIO pins using ModusToolbox™ Device Configurator . . . 28
5.1.1 Using the Device Configurator . . . 28
5.1.2 Device Configurator code preview . . . 31
5.2 GPIO using the Peripheral Driver Library (PDL) . . . 32
5.2.1 GPIO pin initialization - full . . . 32
5.2.2 GPIO pin initialization - fast . . . 33
5.2.3 GPIO port initialization . . . 33
5.2.4 Reading from a GPIO pin . . . 34
5.2.5 Writing to a GPIO pin . . . 34
5.2.6 GPIO interrupt . . . 35
6 GPIO recommendations in ModusToolbox™ software . . . 36
6.1 Toggle an LED . . . 36
6.2 Read an input and write to an output (GPIO pins) . . . 38
6.3 Using bidirectional pin . . . 42
6.4 Toggle GPIOs faster with data registers . . . 44
6.5 Pin interrupt (GPIO interrupt) . . . 46
6.6 Configure GPIO interrupt settings with firmware . . . 49

Table of contents

6.7 Clock buffer in Smart I/O 52
6.8 More code examples 56
7 PSOC™ 4 development boards 57
References 58
Revision history 59
Trademarks 60
Disclaimer 61

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 ModusToolbox™ and the PSOC™ 4 architecture. If you are new to PSOC™ 4, refer AN79953 – Getting started with PSOC™ 4 MCU. If you are new to ModusToolbox™, visit the ModusToolbox™ software. For information on device packages or GPIO specifications, refer to 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 ModusToolbox™ software GPIO tips and tricks Section.

This application note also 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 series, PSOC™ 4 S-series, PSOC™ 4 HV-series devices.

PSOC™ 4 M series, PSOC™ 4 L series devices are currently not supported in ModusToolbox™ are supported only in PSOC™ Creator.

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

2 ModusToolbox™

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 Automotive PSOC™ 4 MCU, PSOC™ 4 kits, and a comprehensive set of middle-ware libraries enabling industry-leading features:

- CAPSENSE™
- LIN peripheral device libraries
- Impressive set of thoroughly tested and helpful code example applications

Visit the ModusToolbox™ software to download the latest version of ModusToolbox™. Refer to the following resources 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
- ModusToolbox™ is compatible with KitProg3 and MiniProg4 (CY8CKIT-005-A) programming devices

2.1 ModusToolbox™ code examples

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

Example codes 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 1 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 2.

2 ModusToolbox™

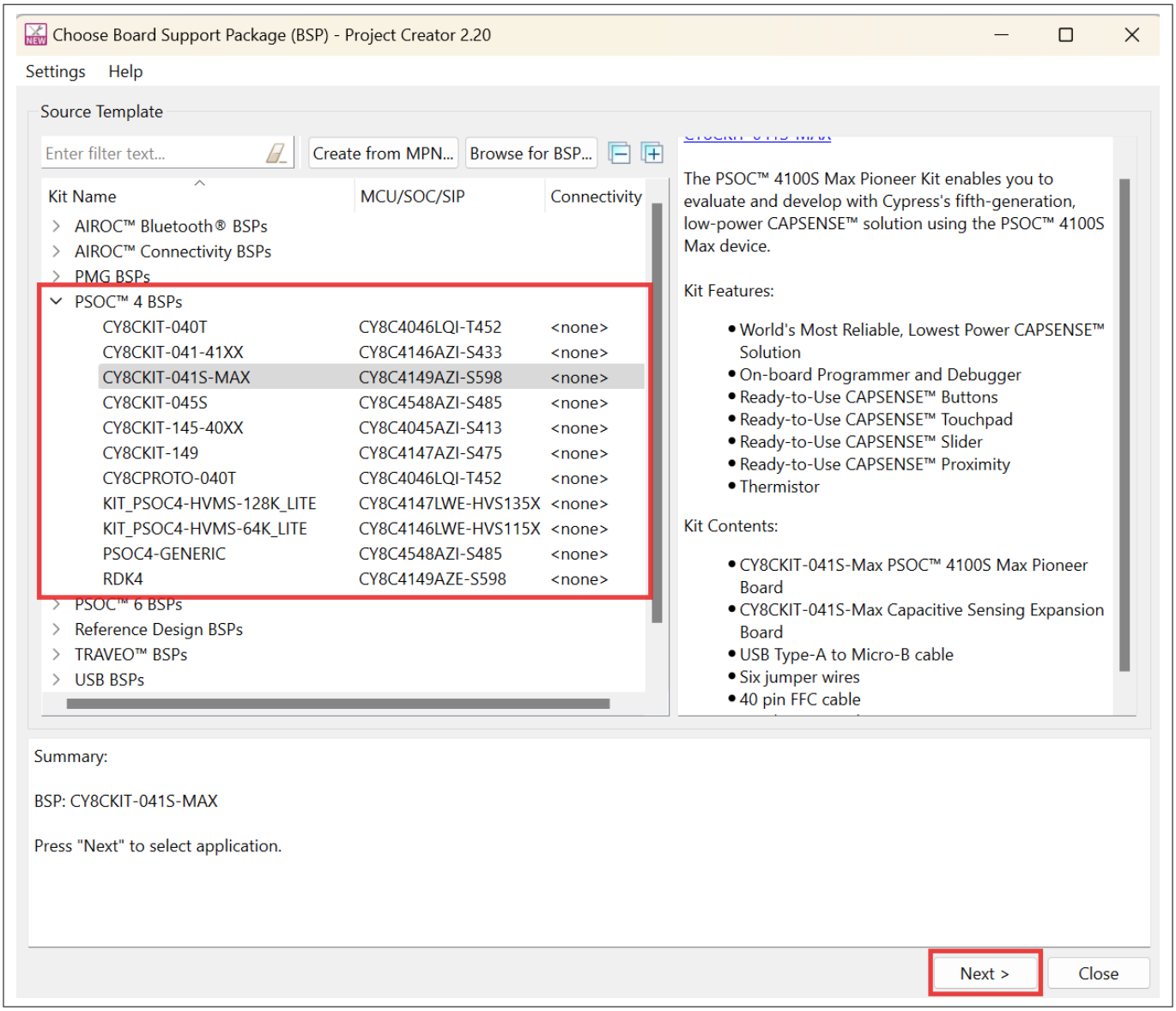


Figure 1 New application wizard

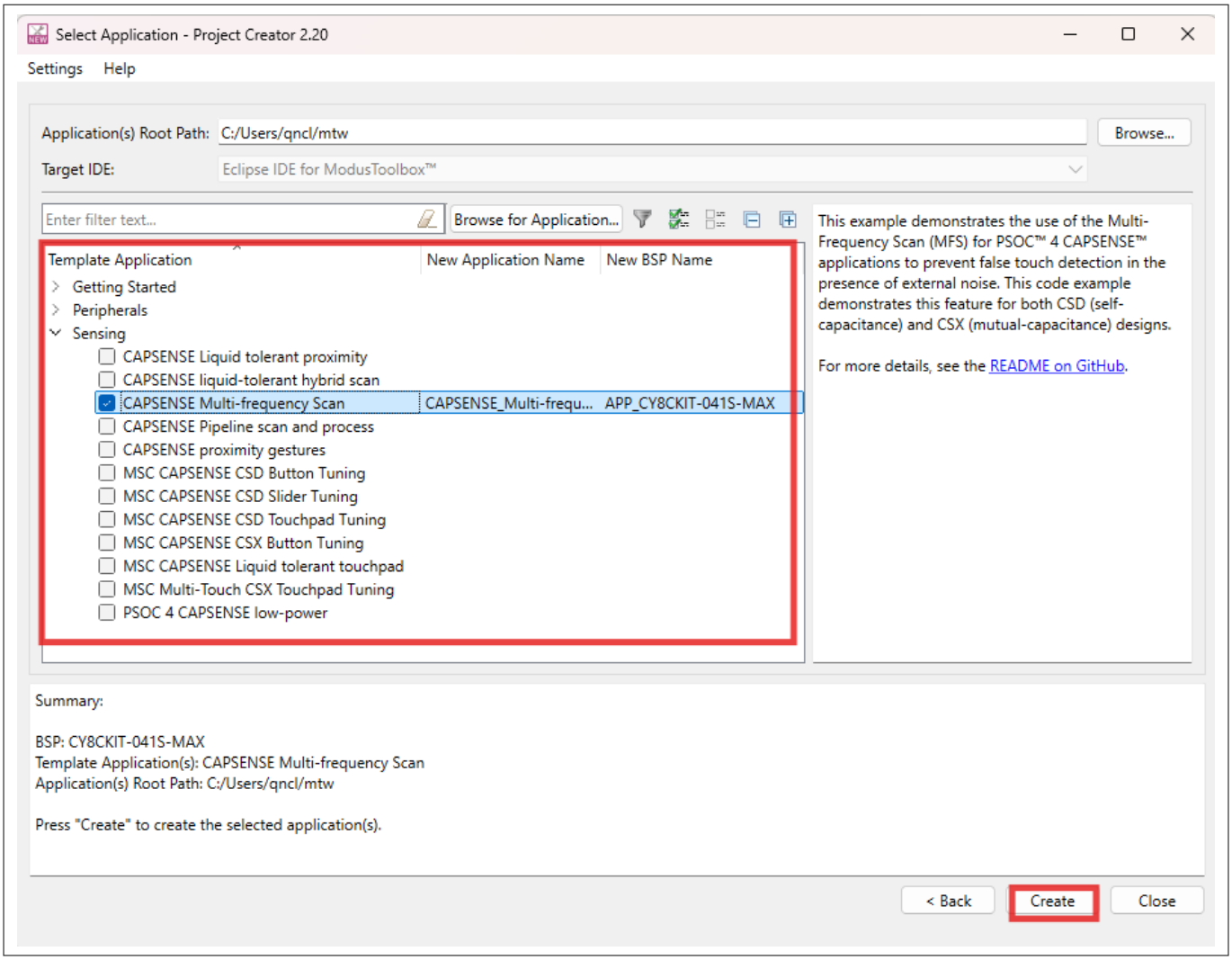


Figure 2 Code examples

2.2 Technical support

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

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™ 4200L)
- 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 M-Series, and PSOC™ 4 L- Series)

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 3 shows the pin connections with the resources in the PSOC™ device.

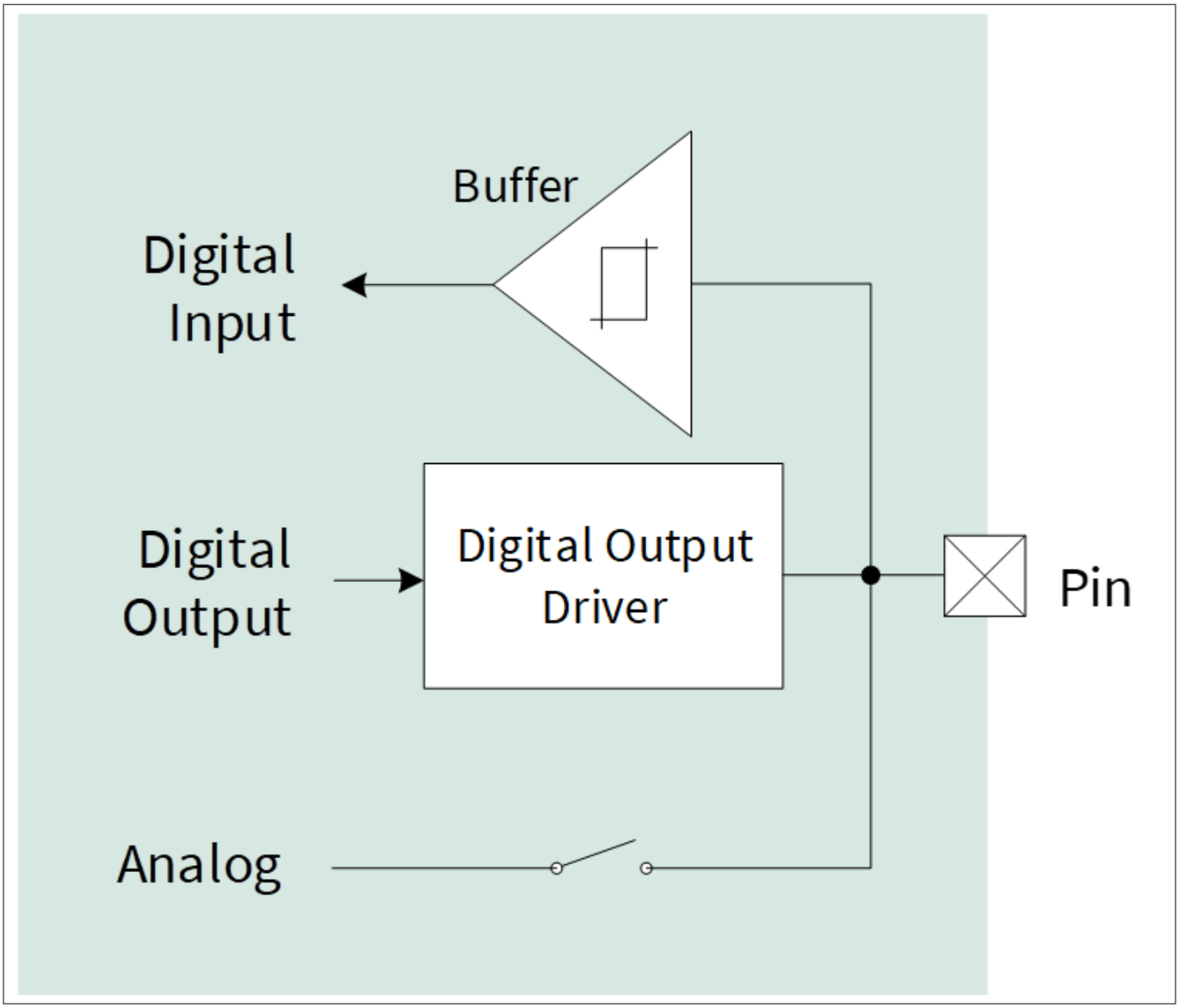


Figure 3 Simplified GPIO block diagram

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^2C. It can also act as an analog pin for use with op-amps 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 and LVTTL.

For the input threshold values, see the device datasheet.

The digital output driver supports different drive modes and slew-rate control (see Figure 4).

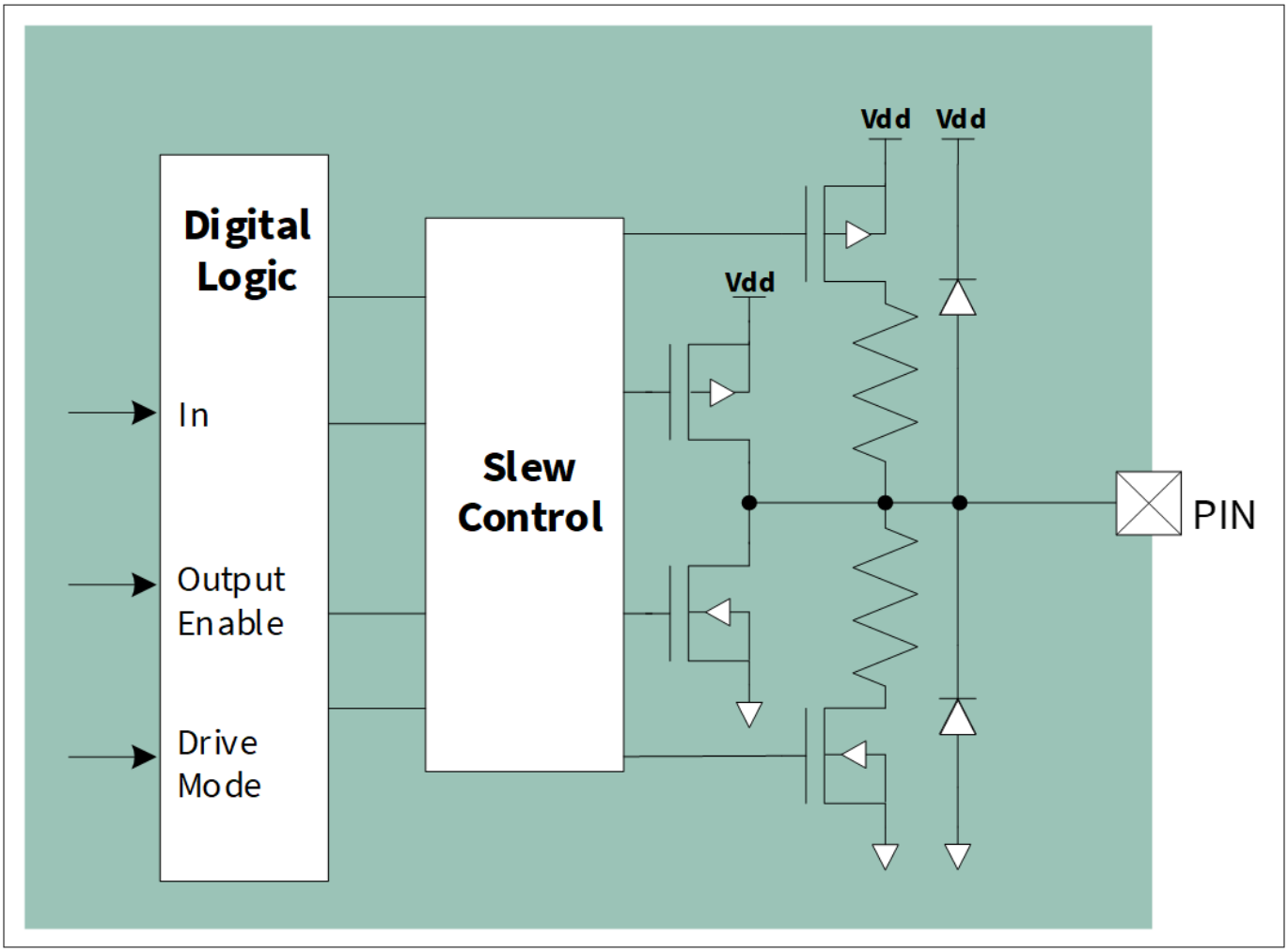


Figure 4 Digital output driver

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

Table 1 Drive modes and applications

#	Drive mode	Application example
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

(table continues...)

Table 1 (continued) Drive modes and applications

#	Drive mode	Application example
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

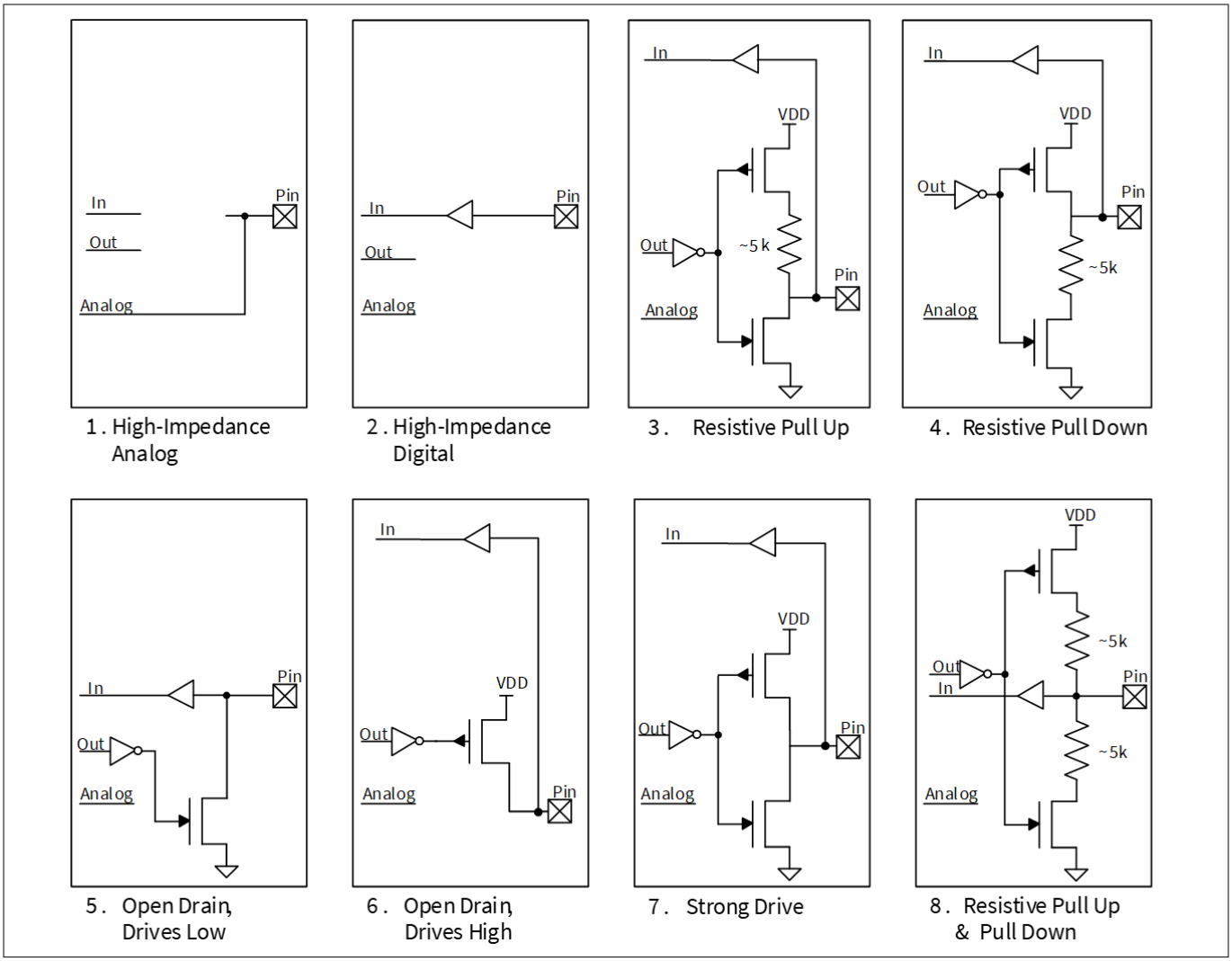


Figure 5 Drive modes

***Note**: The resistor values for pull-up and pull-down drive modes, shown in Figure 5, 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 6 shows the routing for an input pin and Figure 7 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.

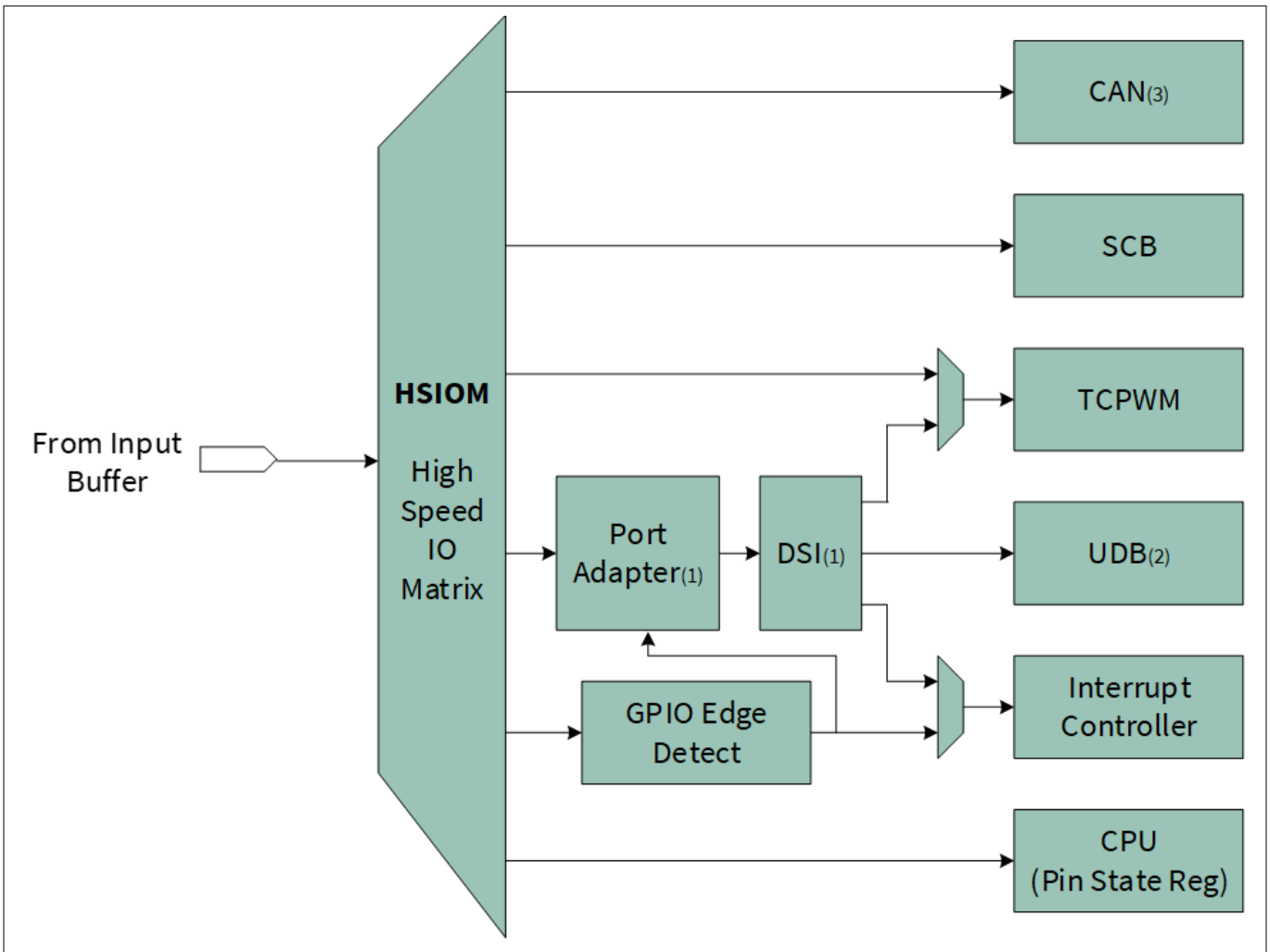


Figure 6 Digital pin input path

SCB (I^2C, 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), 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.

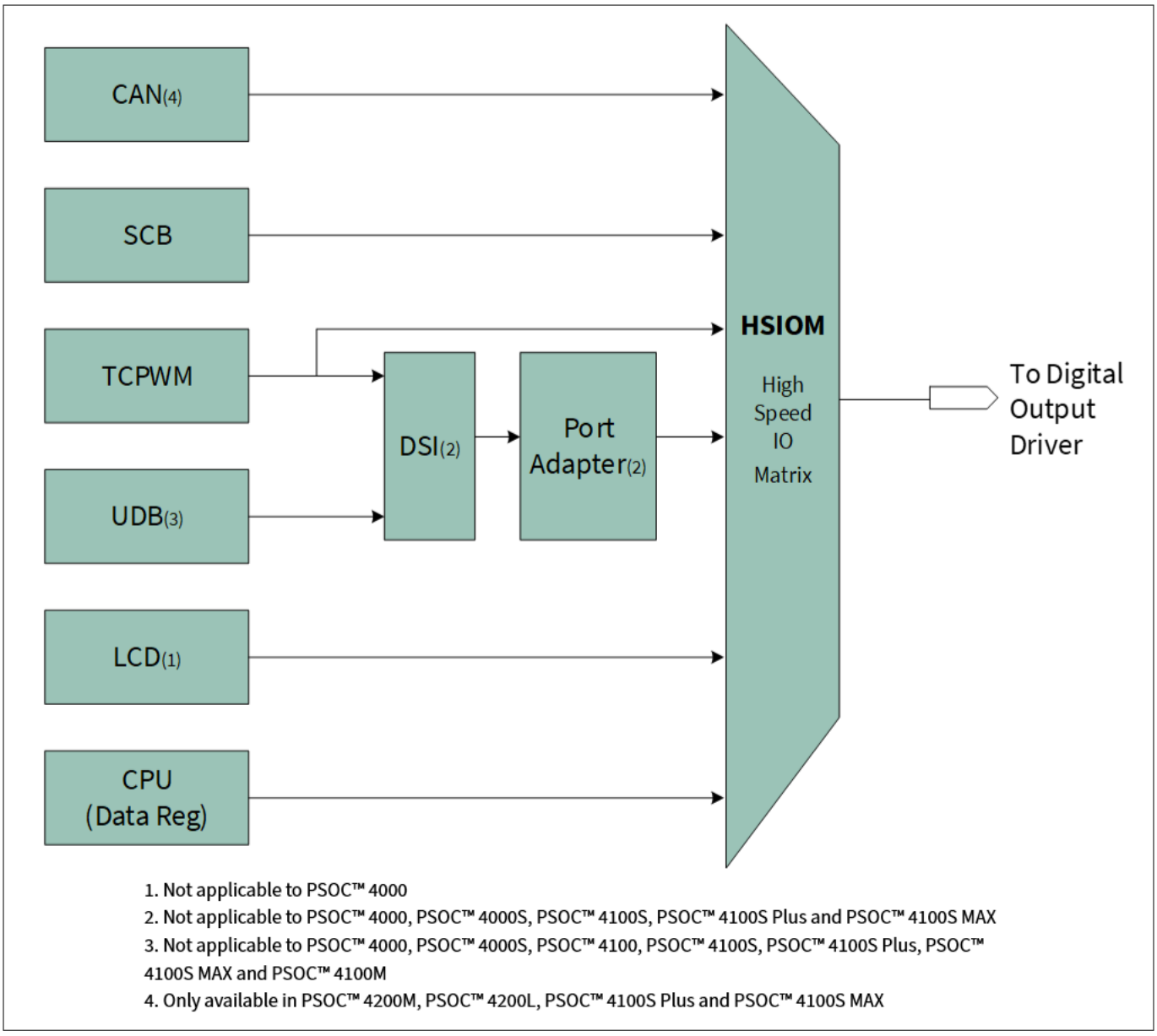


Figure 7 Digital pin output path

***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; so UDB based digital signals cannot be routed to the pins of these ports
- Cannot be used for analog blocks such as SAR ADC, opamp - continuous time block mini (CTBm) and low-power comparator (LPCOMP)
- 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™ pin
- 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 "Pin-outs" 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 8 through Figure 11.

The following are the key highlights of the analog routing in the PSOC™ parts shown in Figure 8:

- 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 10 through Figure 13.

- 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 op-amp inputs and outputs, which provide better performance due to lower trace resistance and parasitic capacitance. Direct connections are also available for low-power comparator (LPCOMP) inputs without switches
- There are dedicated pins for CAPSENSE™ CMOD and the shield tank capacitor. See Figure 10 through Figure 13 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 11 through Figure 13. This is useful if the AMUX buses are required for non-CAPSENSE™ applications such as op-amp/ 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
  - CTBm
  - Temperature sensor output

Multiplexing is done by controlling the switches shown in red in Figure 8 through Figure 11. Note that the SAR ADC can also take the input from any pin using AMUXBUS without the sequencer.

***Note**: The op-amp 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 op-amp 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.*

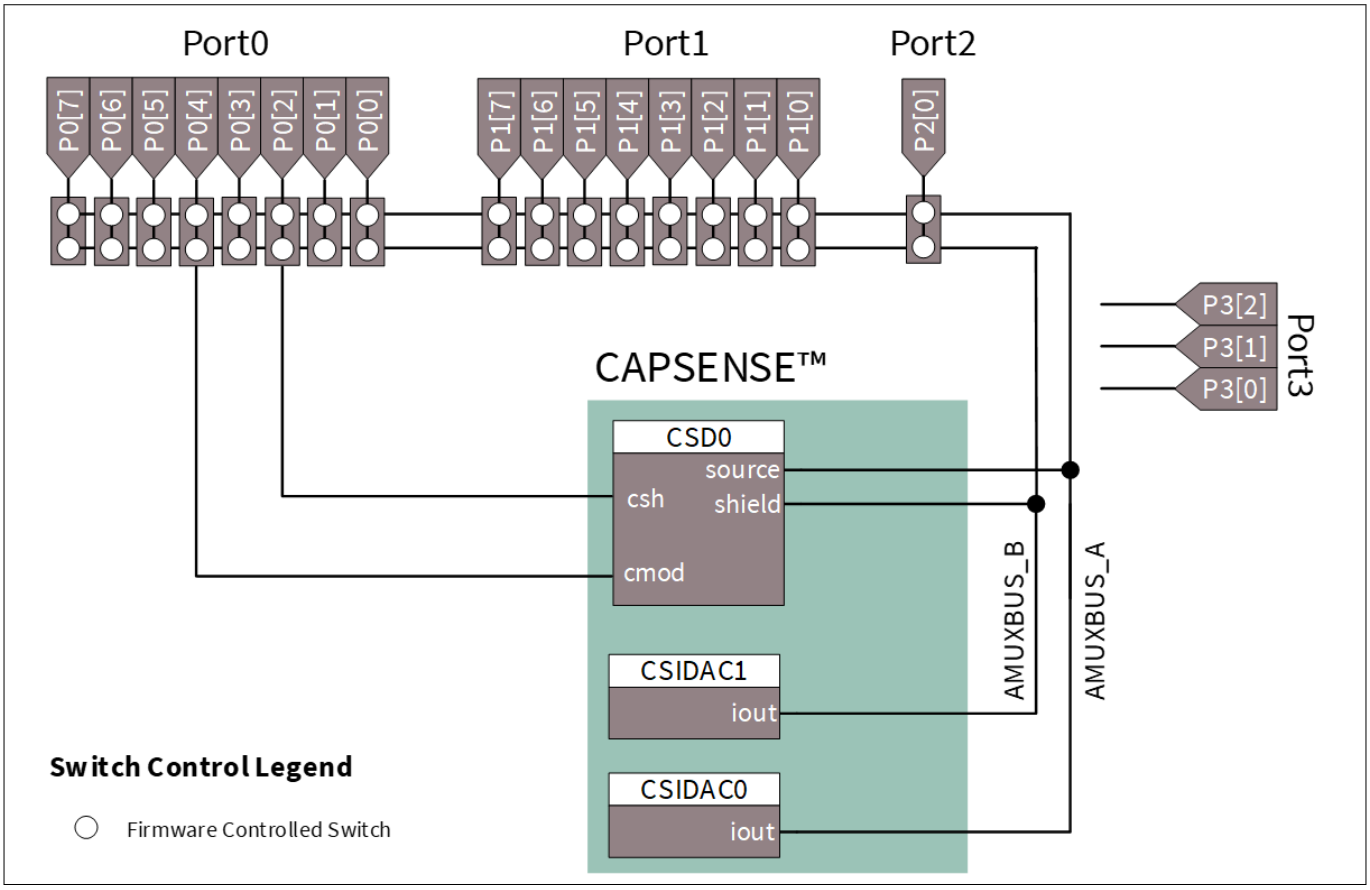


Figure 8 PSOC™ 4000 analog routing diagram

3 GPIO pin basics

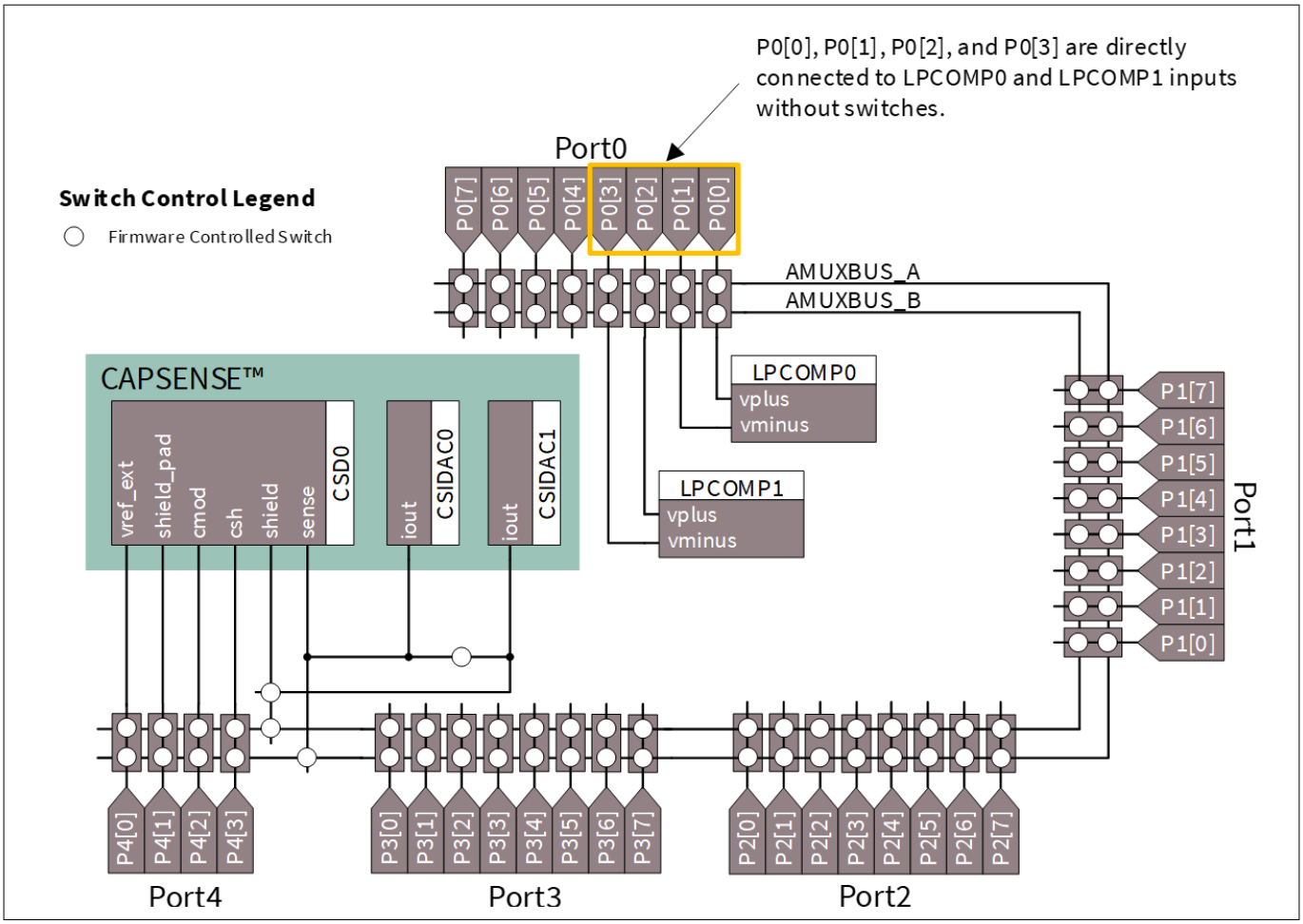


Figure 9 PSOC™ 4000S analog routing diagram

3 GPIO pin basics

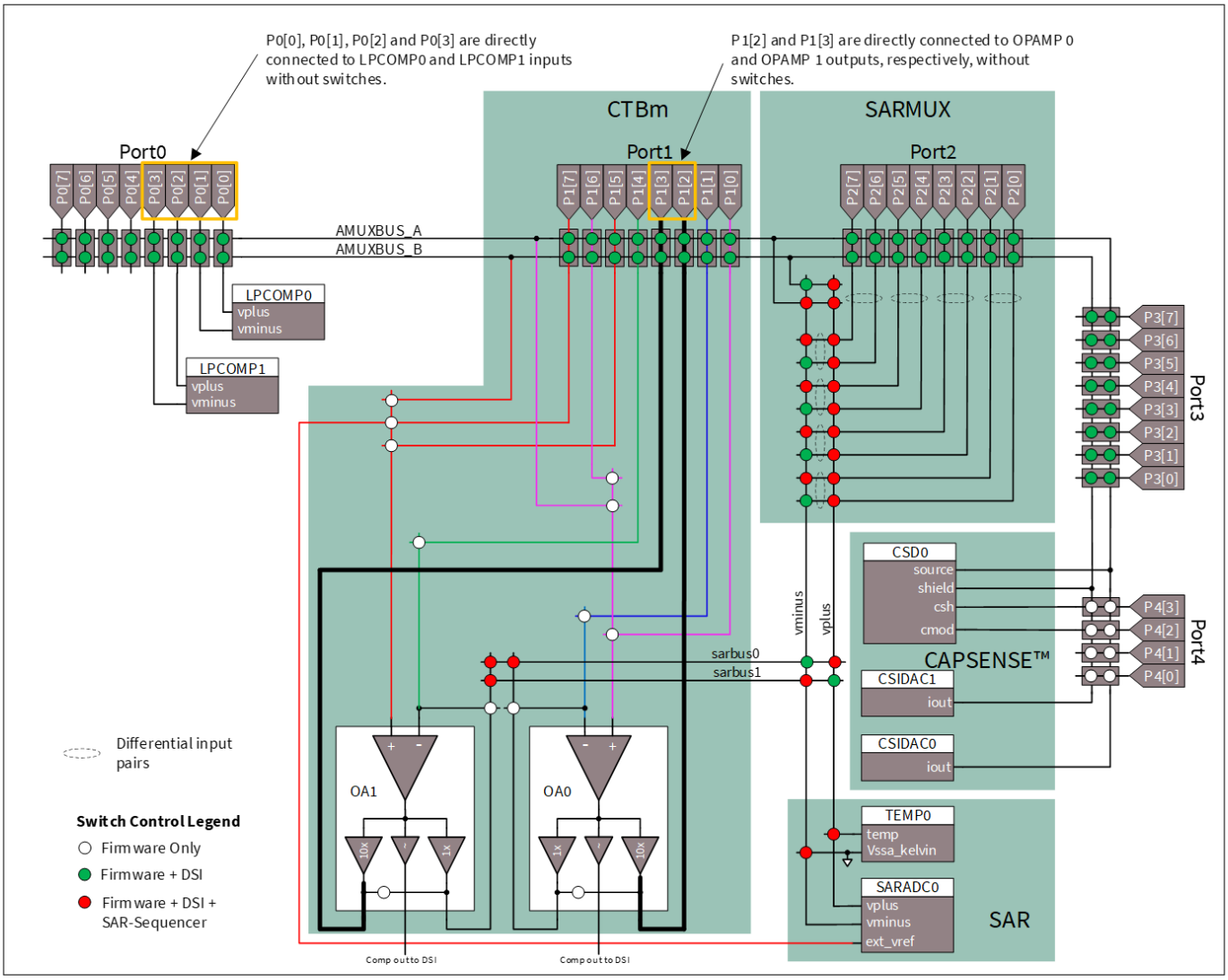


Figure 10 PSOC™ 4200/PSOC™ 4100 analog routing diagram

3 GPIO pin basics

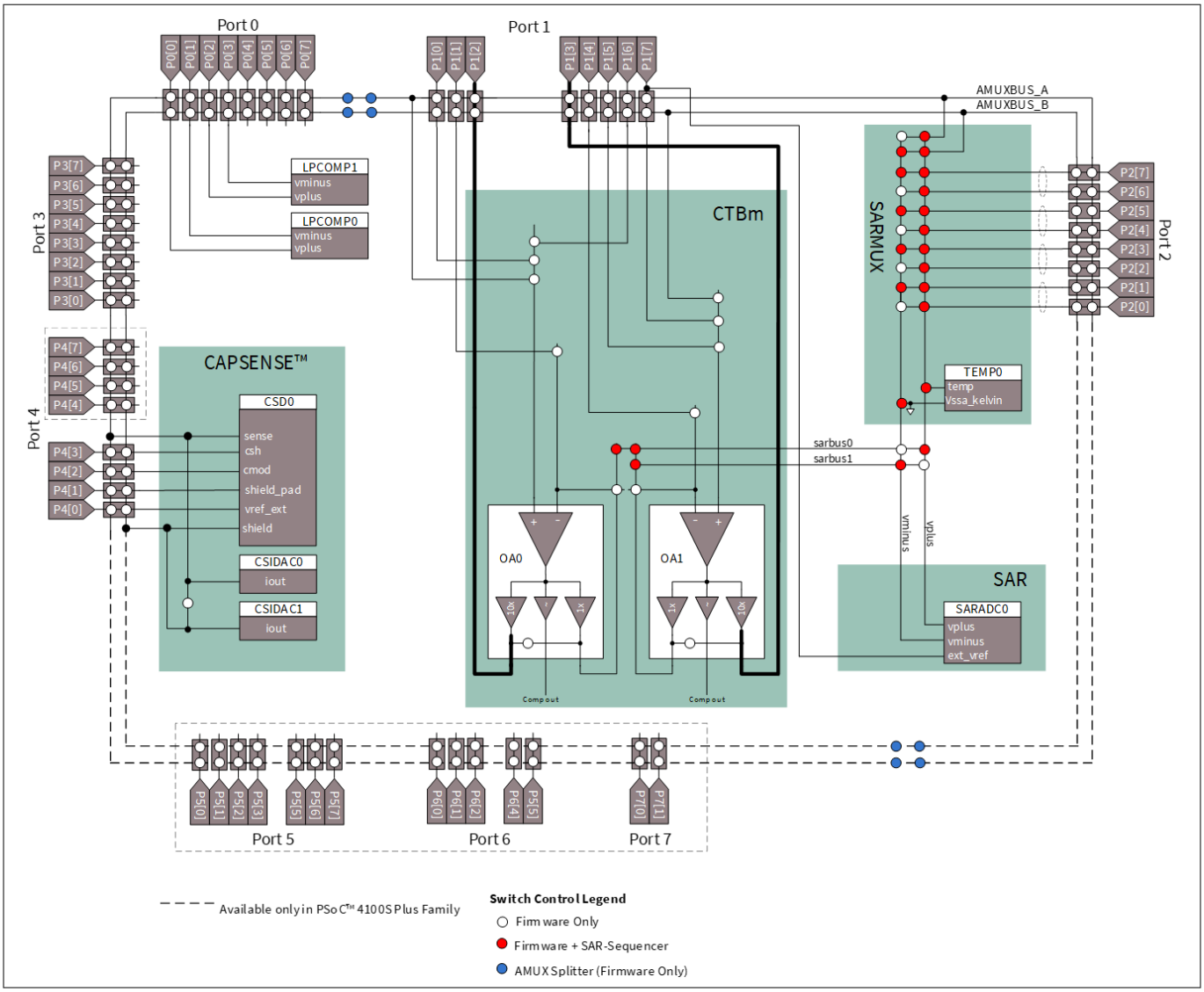


Figure 11 PSOC™ 4100S/PSOC™ 4100S Plus/PSOC™ 4100S MAX analog routing diagram

3 GPIO pin basics

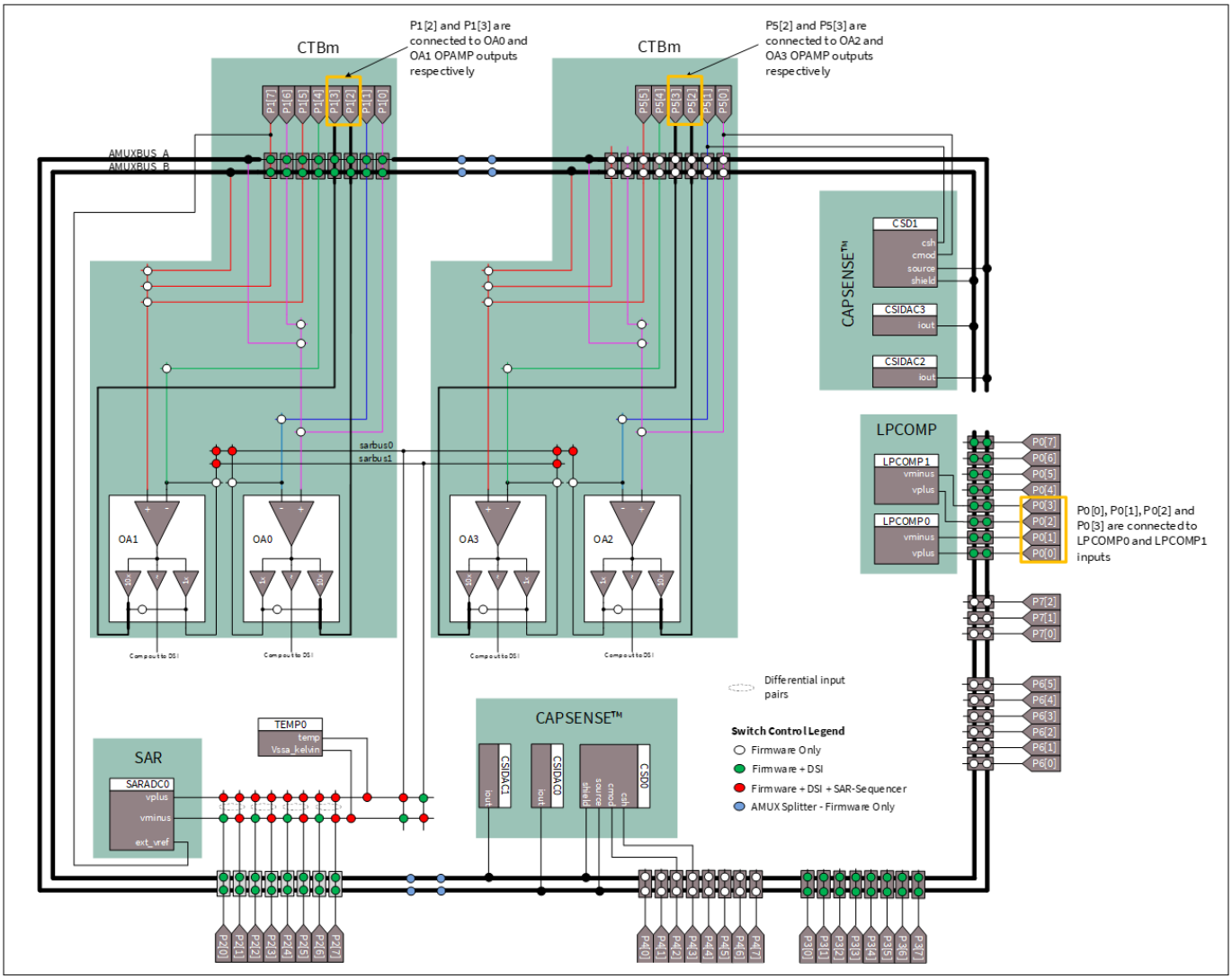


Figure 12 PSOC™ 4100M/PSOC™ 4200M analog routing diagram

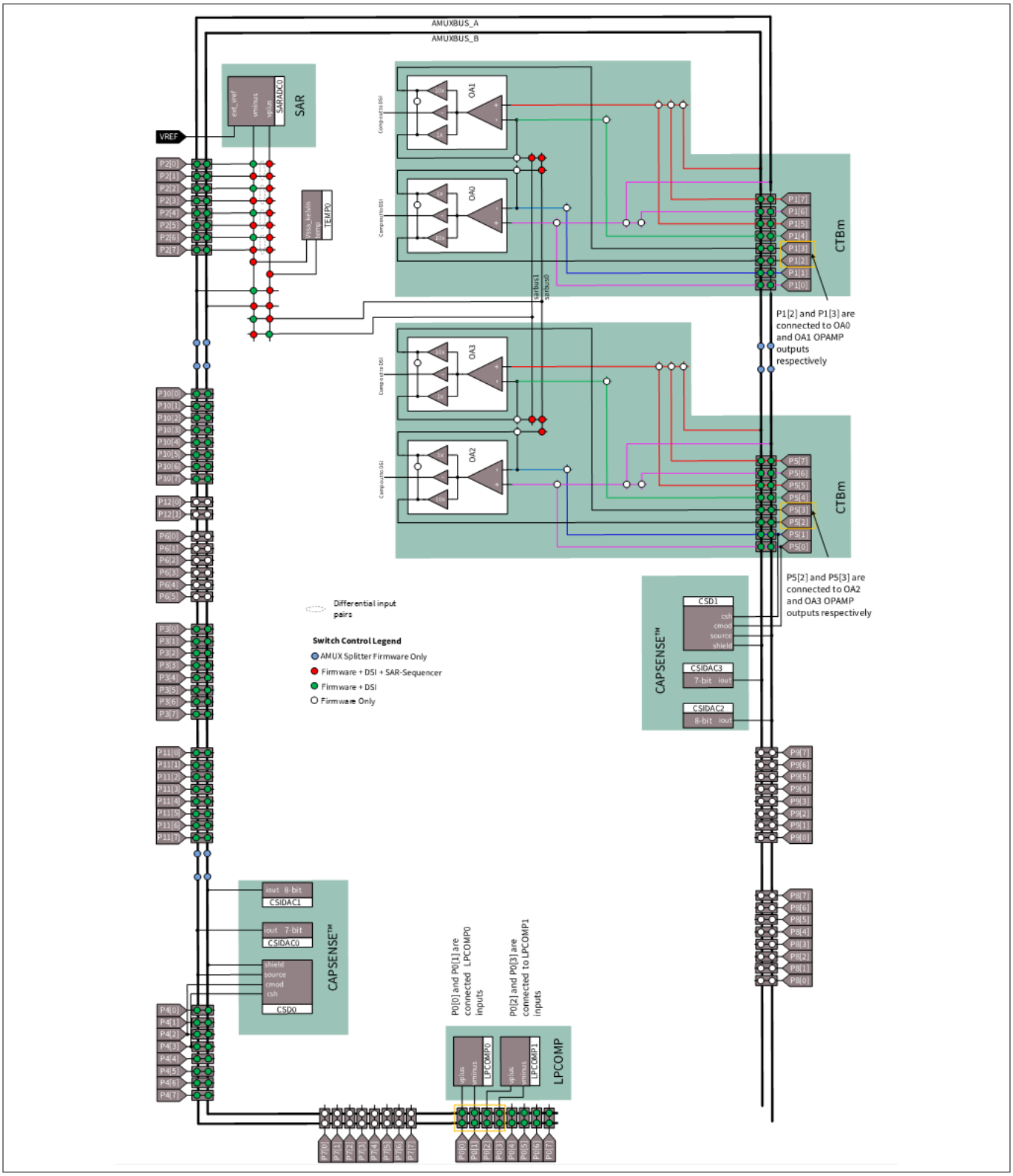


Figure 13 PSOC™ 4200L analog routing diagram

3.3 Start-up 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 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	Yes	Yes	No	No
PSOC™ 4000S	Yes	Yes	No	No
PSOC™ 4100/PSOC™ 4200	Yes	Yes	Yes	Yes
PSOC™ 4100S	Yes	Yes	No	No
PSOC™ 4100S Plus	Yes	Yes	No	No
PSOC™ 4100S Max	Yes	Yes	No	No
PSOC™ 4100S Plus 256 KB	Yes	Yes	No	No
PSOC™ 4700S Plus	Yes	Yes	No	No
PSOC™ 4 M series	Yes	Yes	No	No
PSOC™ 4 L series	Yes	Yes	No	No

The ModusToolbox™ software environment currently supports PSOC™ 4 MCU devices that have Sleep and Deep Sleep power modes.

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^2C, LCD driver, opamp, and comparator are functional. The I^2C 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.

UDB based Components such as control registers are not active and lose the data in Deep Sleep. 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 pin's state have to be controlled manually using software before entering Deep Sleep mode.

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

3.4 GPIO interrupt

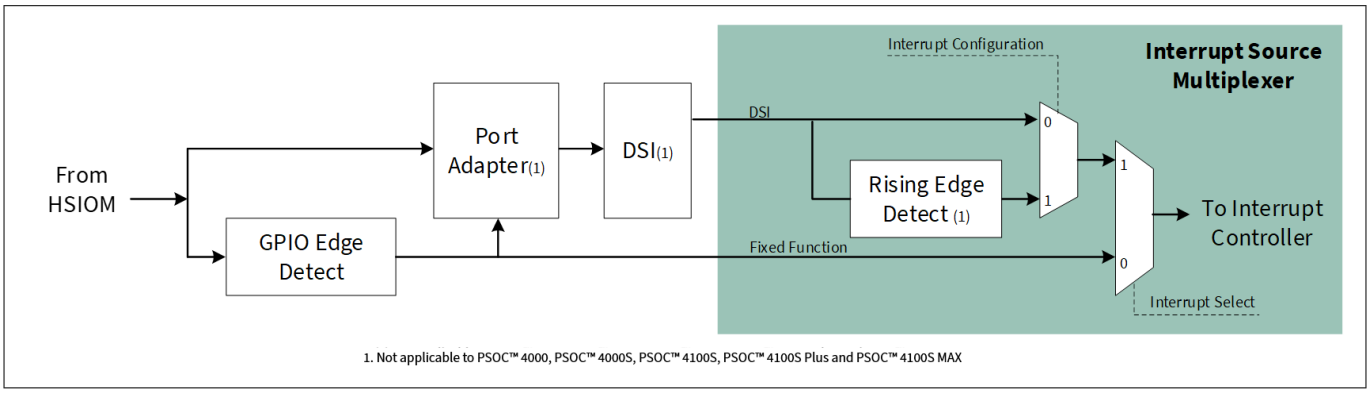


Figure 14 GPIO interrupt signal routing to the interrupt controller

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 show in Figure 14

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 Configuring GPIO pins using ModusToolbox™ Device Configurator 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 OR-ed together to generate a single interrupt request. Thus, there is one interrupt vector for each port.

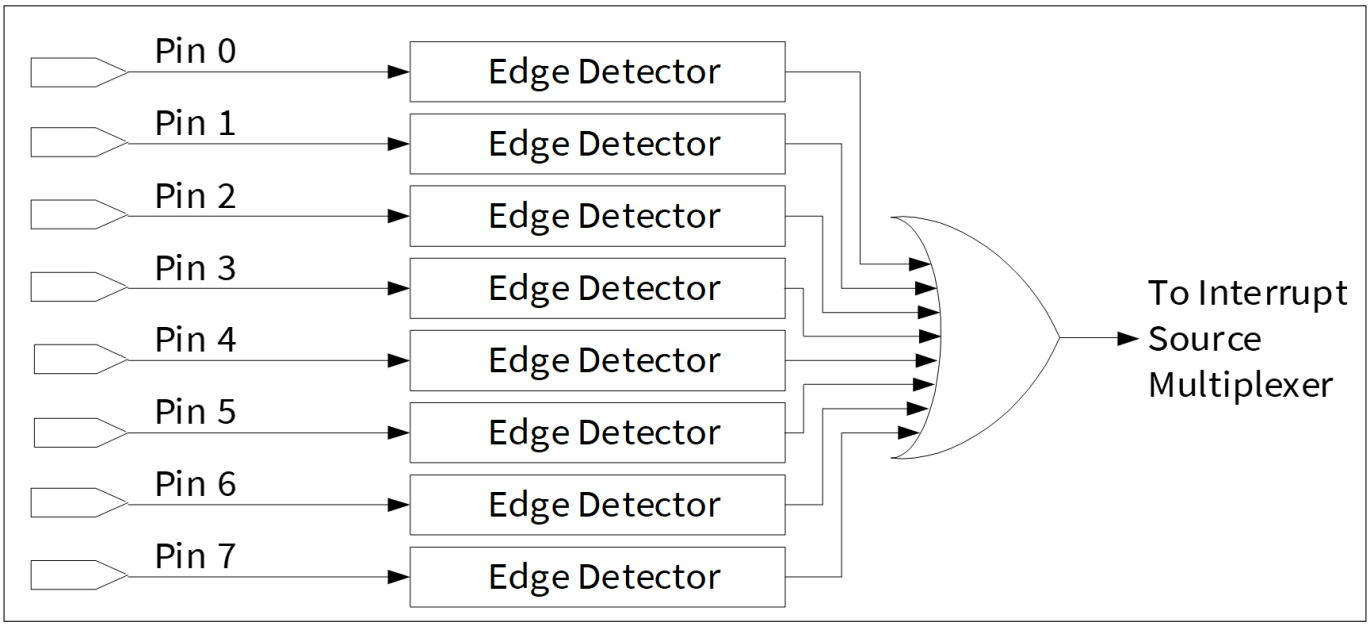


Figure 15 GPIO edge detect

As it is clear from Figure 15, 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 non-responsive 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 wake-up source. A dedicated wake-up pin, P0[7], is available to wake the device from Stop mode in PSOC™ 4200/PSOC™ 4100 and PSOC™ 4 M series.*

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 M series 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.

3.5 Smart I/O

The Smart I/O block adds programmable logic to an I/O port. This programmable logic integrates board-level Boolean logic functionality such as AND, OR, and XOR into the port. The Smart I/O block has these features:

- Integrate board-level Boolean logic functionality into a port
- Ability to preprocess HSIOM input signals from the GPIO port pins
- Ability to post-process HSIOM output signals to the GPIO port pins
- Support in all device power modes
- Integrate close to the I/O pads, providing low signal delay

The Smart I/O block is positioned in the signal path between the HSIOM and the I/O port. The HSIOM multiplexes the output signals from fixed-function peripherals and CPU to a specific port pin and vice versa. The Smart I/O block is placed on this signal path, acting as a bridge that can process signals from port pins and HSIOM, as shown in Figure 16.

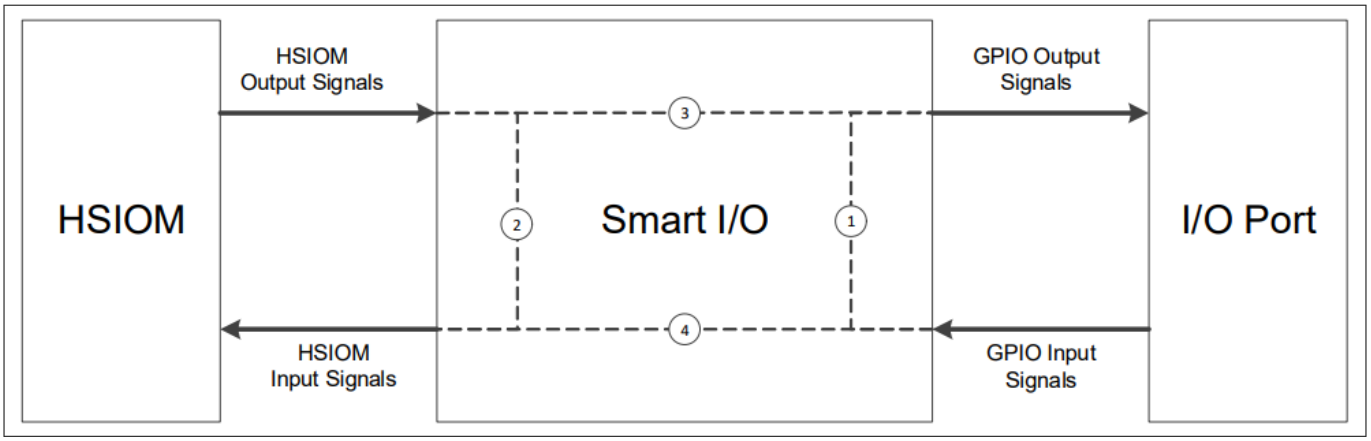


Figure 16 Smart I/O interface

The signal paths supported through the Smart I/O block as shown in Figure 16 are as follows:

1. Implement self-contained logic functions that directly operate on port I/O signals
2. Implement self-contained logic functions that operate on HSIOM
3. Operate on and modify HSIOM output signals and route the modified signals to port I/O signals
4. Operate on and modify port I/O signals and route the modified signals to HSIOM input signals

The following sections discuss the Smart I/O block components, routing, and configuration in detail. In these sections, GPIO signals (io_data_in) refer to the input/output signals from the I/O port; device or chip (chip_data) signals refer to the input/out\u0002put signals from HSIOM.

3.5.1 Block components

The internal logic of the Smart I/O includes these components:

- Clock/reset component
- Synchronizers
- Lookup Table (LUT3) components
- Data unit component

Details can be found in device architecture reference manual.

3.5.2 Routing

The Smart I/O block includes many switches that are used to route the signals in and out of the block and also between various components present inside the block. The routing switches are handled through the PRTGIO_PRTx_LUT_SELy and PRGIO_PRTx_DU_SEL registers. The Smart I/O internal routing is shown in Figure 17. In the figure, note that LUT7 to LUT4 operate on io_data_in/chip_data[7] to io_data_in/chip_data[4] whereas LUT3 to LUT0 operate on io_data_in/chip_data[3] to io_data_in/chip_data[0].

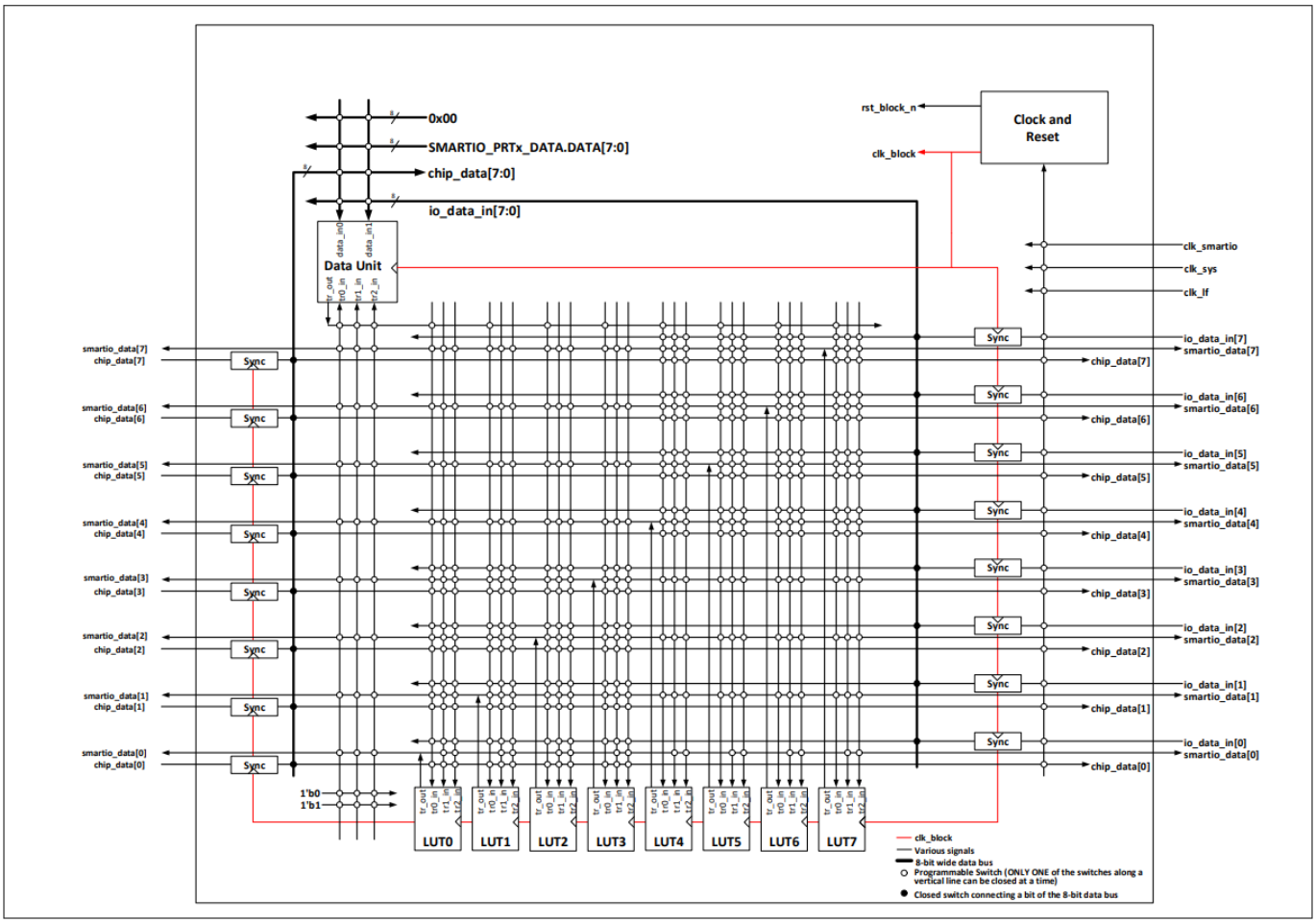


Figure 17 Smart I/O routing

3.5.3 Operation

The Smart I/O block should be configured and operated as follows. See Table 3 for register control details.

1. Before enabling the block, all the components should be configured and the routing should be selected, refer device architecture reference manual
2. In addition to configuring the components and routing, some block level settings need to be configured correctly for desired operation
   - **a.** Bypass control: The Smart I/O path can be bypassed for a particular GPIO signal by setting the BYPASS[i] bit field in the PRGIO_PRTx_CTL register. When bit 'i' is set in the BYPASS[7:0] bit field, the 'i'th GPIO signal is bypassed to the HSIOM signal path directly – Smart I/O logic will not be present in that signal path. This is useful when the Smart I/O functionality is required only on select I/Os
   - **b.** Pipe-lined trigger mode: The LUT3 input multiplexers and the LUT3 component itself do not include any combinatorial loops. Similarly, the data unit also does not include any combinatorial loops. However, when one LUT3 interacts with the other or to the data unit, inadvertent

combinatorial loops are possible. To overcome this limitation, the PIPE\u0002LINE_EN bit field of the PRGIO_PRTx_CTL register is used. When set, all the outputs (LUT3 and data unit) are regis\u0002tered (flopped) before branching out to other components. The output will be unflopped when the PIPELINE_EN bit is cleared

3. After the Smart I/O block is configured for the desired functionality, the block can be enabled by setting the ENABLED bit field of the PRGIO_PRTx_CTL register. If disabled, the Smart I/O block is put in bypass mode, where the GPIO signals are directly controlled by the HSIOM signals and vice-versa. The Smart I/O block must be configured; that is, all register settings must be updated before enabling the block to prevent glitches during register updates

Table 3 Smart I/O block controls

Register [BIT_POS]	Bit Name	Description
PRGIO_PRTx_CTL[25]	PIPELINE_EN	Enable for pipeline register: '0': Disabled (register is bypassed). '1': Enabled
PRGIO_PRTx_CTL[31]	ENABLED	Enable Smart I/O. Should only be set to '1' when the Smart I/O is completely configured: '0': Disabled (signals are bypassed; behavior as if BYPASS[7:0] is 0xFF). When disabled, the block (data unit and LUTs) reset is activated. If the block is disabled: • The PIPELINE_EN register field should be set to '1', to ensure low power consumption • The CLOCK_SRC register field should be set to 20 to 30 (clock is constant '0'), to ensure low power consumption '1': Enabled. When enabled, it takes three “clk_block” clock cycles until the block reset is de\u0002activated and the block becomes fully functional. This action ensures that the I/O pins' input synchronizer states are flushed when the block is fully functional
PRGIO_PRTx_CTL[7:0]	BYPASS[7:0]	Bypass of the Smart I/O, one bit for each I/O pin: BYPASS[i] is for I/O pin i. When ENABLED is '1', this field is used. When ENABLED is '0', this field is not used and Smart I/O is always bypassed. '0': No bypass (Smart I/O is present in the signal path) '1': Bypass (Smart I/O is absent in the signal path)

4 Overvoltage-tolerant (OVT) pins

Port 6 in PSOC™ 4 M series are the OVT pins. For PSOC™ 4 L series, 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^2C and its lines routed to OVT pins, an SCB meets the following I^2C specifications:
4. Fast Mode Plus LOW-level output current (IOL) specification
5. 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 ModusToolbox™ software

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

5.1 Configuring GPIO pins using ModusToolbox™ Device Configurator

5.1.1 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.

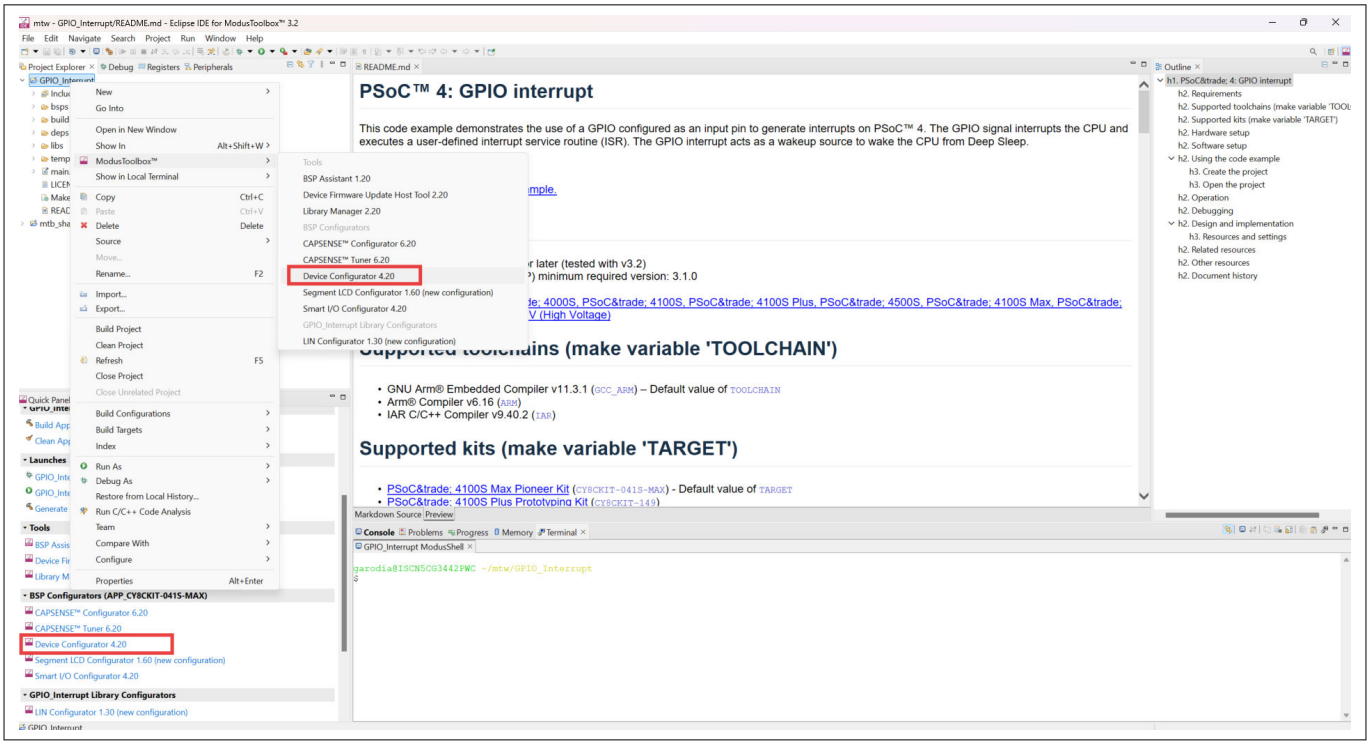


Figure 18 Accessing the Device Configurator

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 Configurator window.

5 GPIO pins in ModusToolbox™ software

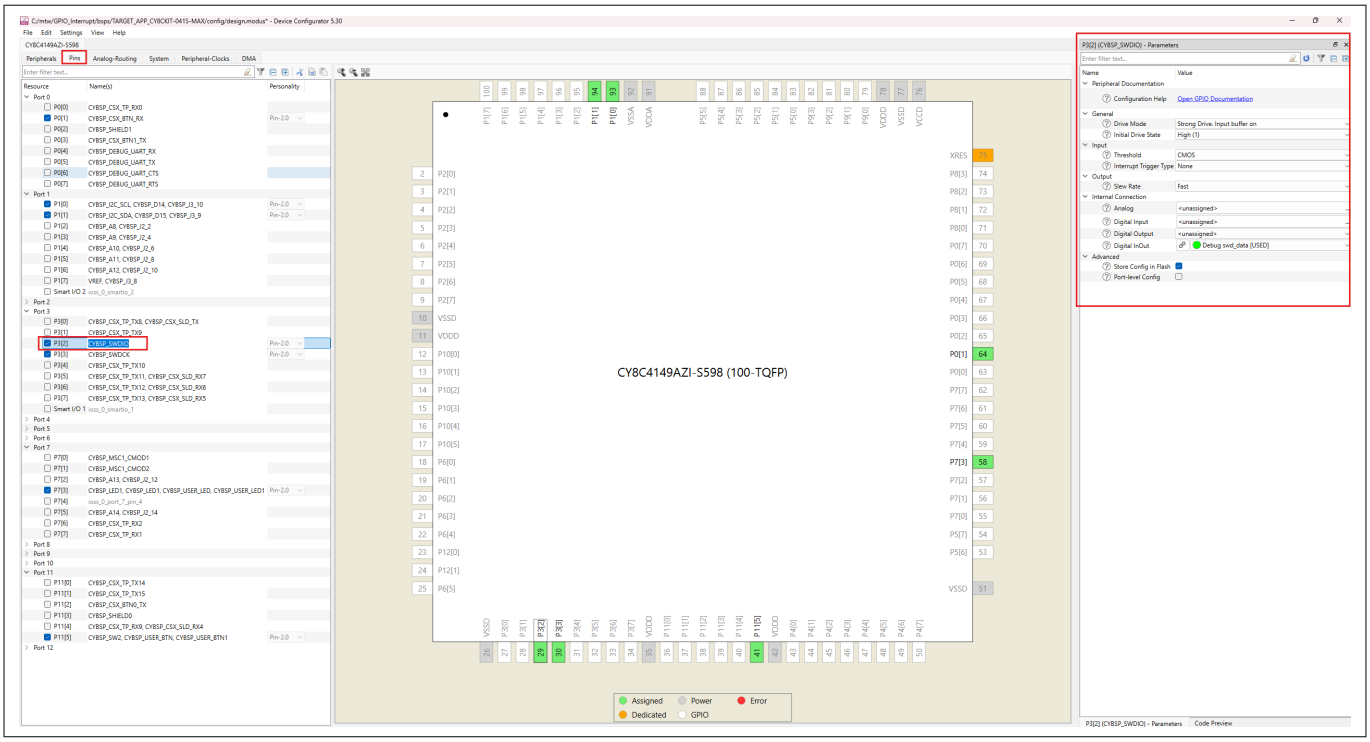


Figure 19 Device Configurator

Table 4 Pin component settings

Setting	Description
General > **Drive Mode**	This setting configures the pin with one of the eight drive modes as shown in Configuring GPIO pins using ModusToolbox™ Device Configurator. 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 Configuring GPIO pins using ModusToolbox™ Device Configurator. 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

(table continues...)

Table 4 (continued) Pin component settings

Setting	Description
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^2C interfaces
Advanced > **Store Config in Flash**	This setting controls whether the configuration structure is stored in flash (const, true) or SRAM (not const, false)

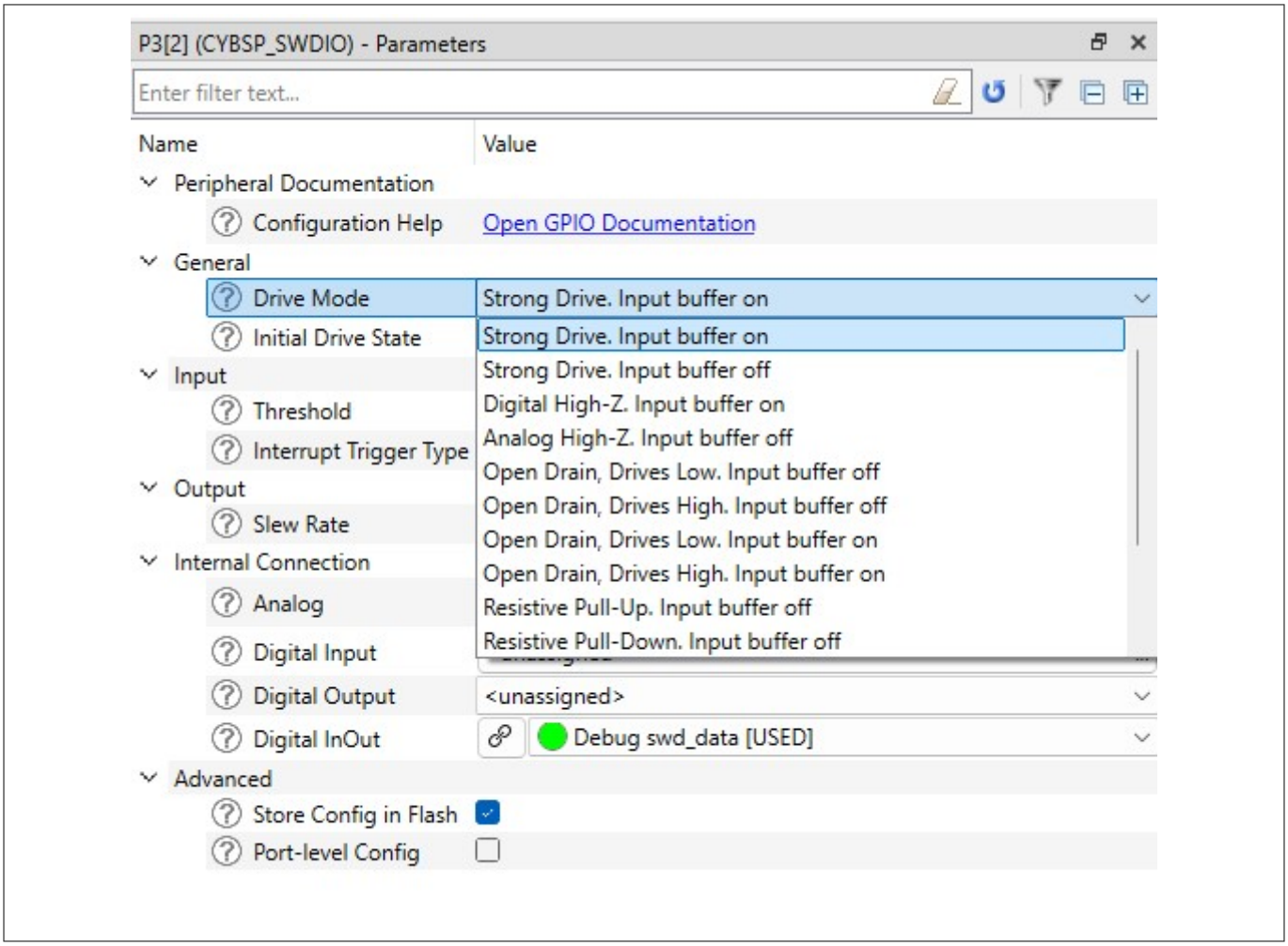


Figure 20 Drive modes

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 the 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 Configuring GPIO pins using ModusToolbox™ Device Configurator.*

5.1.2 Device Configurator code preview

Code Preview

Enter search text...

```
/*
 * NOTE: This is a preview only. It combines elements of the
 * cycfg_pins.c and cycfg_pins.h files located in the folder:
 * C:/Users/garodia/mtw/GPIO_Interrupt/bsps/TARGET_APP_CY8CKIT-041S-MAX/config/GeneratedSource
 */

#include "cy_gpio.h"
#include "cycfg_routing.h"

#if defined (CY_USING_HAL)
#include "cyhal_hwmgr.h"
#endif /* defined (CY_USING_HAL) */

#define CYBSP_SWDIO_PORT GPIO_PRT3
#define CYBSP_SWDIO_PORT_NUM 3U
#define CYBSP_SWDIO_PIN 2U
#define CYBSP_SWDIO_NUM 2U
#define CYBSP_SWDIO_DRIVEMODE CY_GPIO_DM_STRONG
#define CYBSP_SWDIO_INIT_DRIVESTATE 1
#ifndef ioss_0_port_3_pin_2_HSIOM
    #define ioss_0_port_3_pin_2_HSIOM HSIOM_SEL_GPIO
#endif
#define CYBSP_SWDIO_HSIOM ioss_0_port_3_pin_2_HSIOM
#define CYBSP_SWDIO_IRQ ioss_interrupts_gpio_3_IRQn

#if defined (CY_USING_HAL)
#define CYBSP_SWDIO_HAL_PORT_PIN P3_2
#define CYBSP_SWDIO P3_2
#define CYBSP_SWDIO_HAL_IRQ CYHAL_GPIO_IRQ_NONE
#define CYBSP_SWDIO_HAL_DIR CYHAL_GPIO_DIR_BIDIRECTIONAL
#define CYBSP_SWDIO_HAL_DRIVEMODE CYHAL_GPIO_DRIVE_STRONG
#endif /* defined (CY_USING_HAL) */

const cy_stc_gpio_pin_config_t CYBSP_SWDIO_config =
{
    .outVal = 1,
    .driveMode = CY_GPIO_DM_STRONG,
    .hsiom = CYBSP_SWDIO_HSIOM,
    .intEdge = CY_GPIO_INTR_DISABLE,
    .vtrip = CY_GPIO_VTRIP_CMOS,
    .slewRate = CY_GPIO_SLEW_FAST,
};

#if defined (CY_USING_HAL)
const cyhal_resource_inst_t CYBSP_SWDIO_obj =
{
    .type = CYHAL_RSC_GPIO,
    .block_num = CYBSP_SWDIO_PORT_NUM,
    .channel_num = CYBSP_SWDIO_PIN,
};
#endif /* defined (CY_USING_HAL) */

void init_cycfg_pins(void)
{
    Cy_GPIO_Pin_Init(CYBSP_SWDIO_PORT, CYBSP_SWDIO_PIN, &CYBSP_SWDIO_config);
}
void reserve_cycfg_pins(void)
{
#if defined (CY_USING_HAL)
    cyhal_hwmgr_reserve(&CYBSP_SWDIO_obj);
#endif /* defined (CY_USING_HAL) */
}
```

P3[2] (CYBSP_SWDIO) - Parameters | Code Preview

Figure 21 Device Configurator 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 Configuring GPIO pins using ModusToolbox™ Device Configurator. 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.

5.2 GPIO using the Peripheral Driver Library (PDL)

The PDL integrates device header files, start-up 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, so 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 Configuring GPIO pins using ModusToolbox™ 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 as explained in section Configuring GPIO pins using ModusToolbox™ Device Configurator.

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 */
}
```

5.2.1 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 as shown in Configuring GPIO pins using ModusToolbox™ Device Configurator.

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.

5.2.2 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 as shown in Configuring GPIO pins using 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.

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);
```

5.2.3 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.

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 */
}
```

5.2.4 Reading from a GPIO pin

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

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 */
}
```

5.2.5 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 as mentioned in section Configuring GPIO pins using ModusToolbox™ Device Configurator. The port and the pin are arguments when using the `CY_GPIO_Write` function as shown below.

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);
```

5.2.6 GPIO interrupt

The GPIO interrupt for a pin is configured using one of two methods: using the Device Configurator as seen in Configuring GPIO pins using ModusToolbox™ Device Configurator, 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 . 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.

6 GPIO recommendations in ModusToolbox™ software

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

6.1 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. Open the Project Creator GUI tool and on the **Choose Board Support Package (BSP)** page, select the kit followed by code example section as shown in section ModusToolbox™ code examples

   ***Note**: If the code example is not present in the list of Project Creator, select "Empty PSOC™ 4 App" inside "Getting started" and follow these steps.*

2. Open Device Configurator and select Pin section as shown in section Configuring GPIO pins using ModusToolbox™ Device Configurator
3. Select the Pin and name the component "CYBSP_PIN_LED" and configure the drive mode as shown in Figure 22

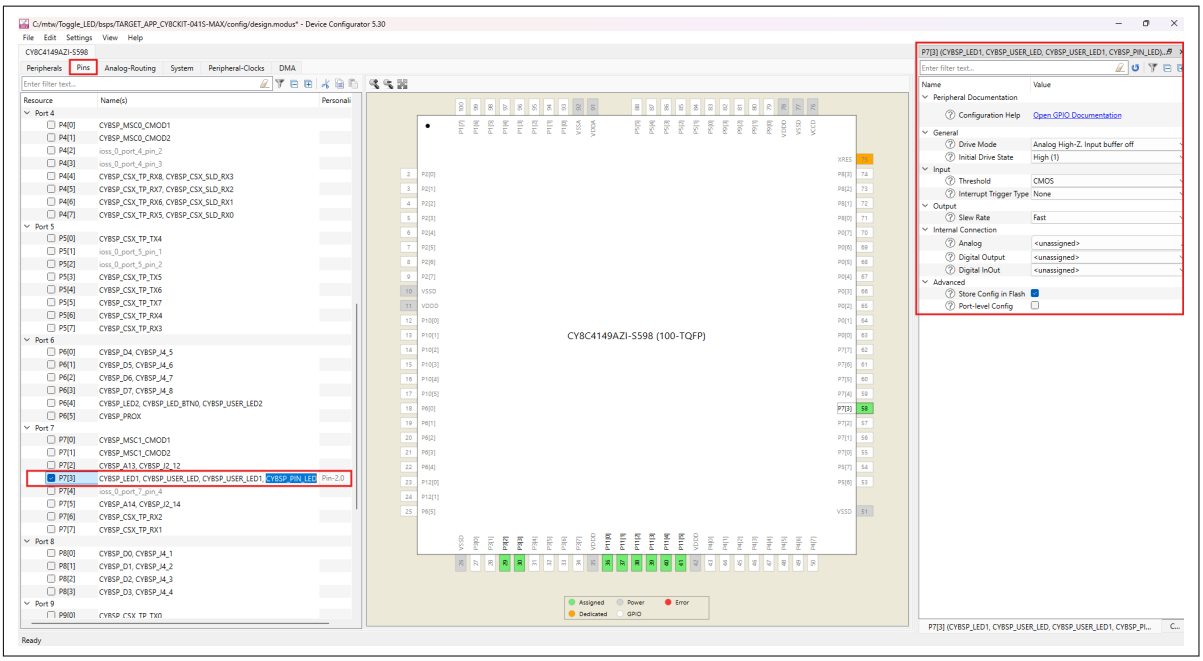


Figure 22 Device configuration for CYBSP_PIN_LED configuration

4. Save and close the Device Configurator

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

```
/*******************************************************************************
* Macros
*******************************************************************************/
#define LED_DELAY_MS            (500u)
#define CY_ASSERT_FAILED        (0u)


/*******************************************************************************
* Function Name: main
********************************************************************************
* Summary:
*  System entrance point. This function performs
*  - initial setup of device
*  - Toggles an LED under firmware control at 1 Hz
*******************************************************************************/
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    for (;;)
    {
        /* Toggle the user LED state */
        Cy_GPIO_Inv(CYBSP_PIN_LED_PORT, CYBSP_PIN_LED_NUM);

        /* Wait for 0.5 seconds */
        Cy_SysLib_Delay(delayMs);
    }
}
```

6. Build the project and program the PSOC™ 4 device as shown in Figure 23

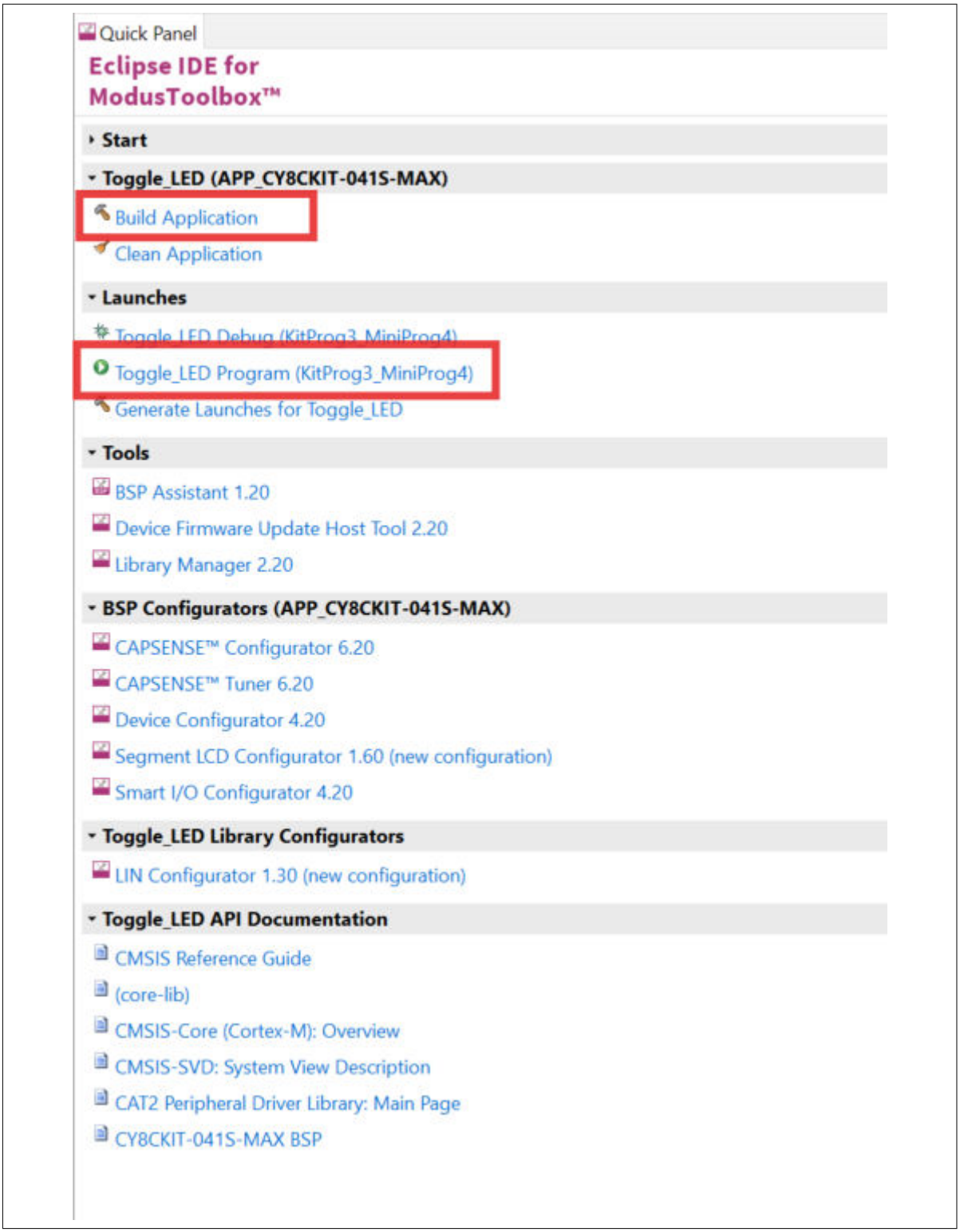


Figure 23 Build and Program the Toggle LED project using Quick panel

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

6.2 Read an input and write to an output (GPIO pins)

This example demonstrates multiple methods of configuring, reading, writing, and generating interrupts with PSOC™ 4 general-purpose input/output (GPIO) pins.

6 GPIO recommendations in ModusToolbox™ software

1. Open the Project Creator GUI tool and on the **Choose Board Support Package (BSP)** page, select the kit followed by code example section as shown in section ModusToolbox™ code examples

   ***Note**: If the code example is not present in the list of Project Creator, select "Empty PSOC™ 4 App" inside "Getting started" and follow these steps.*

2. Open Device Configurator and select Pin section as shown in section Configuring GPIO pins using ModusToolbox™ Device Configurator
3. Select the Pins, name the components and configure the drive mode as shown in Figure 24

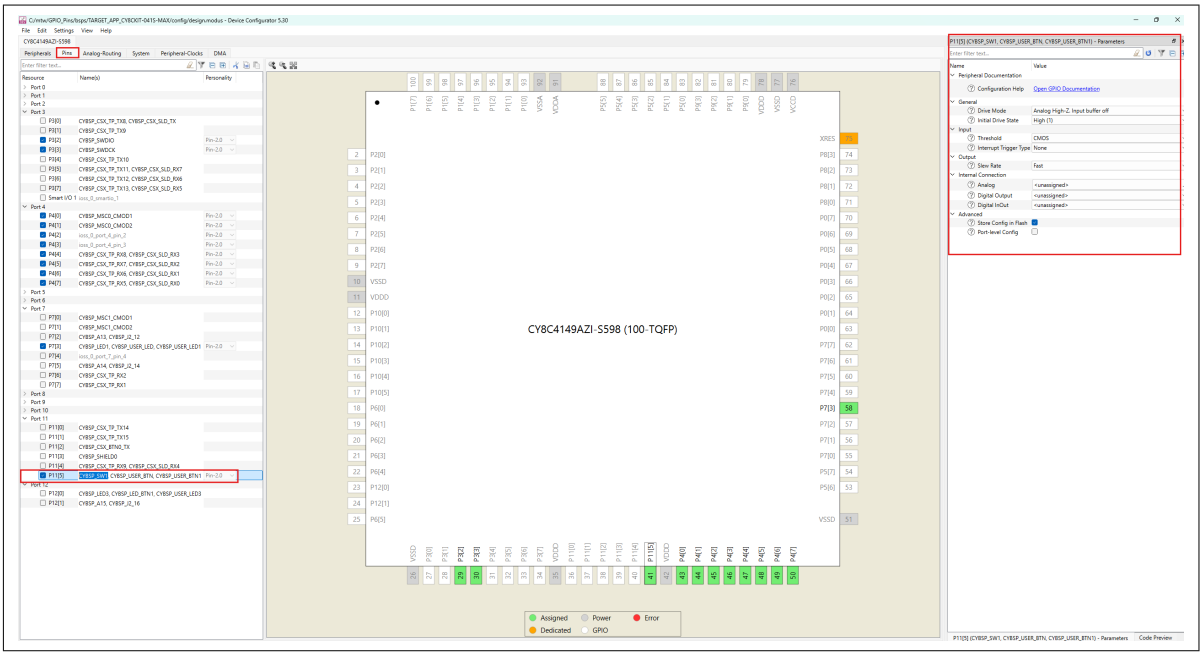


Figure 24 Device configuration for GPIO pins code example configuration

4. Save and close the Device Configurator

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

```
for(;;)
    {
        /* Simultaneous Port Pin access */

        /* Direct register access is used to interface with multiple pins in
         * one port at the same time. May not be thread or multi-core safe due
         * to possible read-modify-write operations. All pins in a Port under
         * direct register control should only be accessed by a single CPU core.
         */
        portReadValue = GPIO_PRT4->DR;
        portReadValue++;
        GPIO_PRT4->DR = portReadValue;

        if(interruptFlag == INTERRUPT_FLAG_SET)
        {
            /* If interrupt occurs i.e., if button is pressed, read the input
             * value from the Reference pin and write the value to the User LED.
             */
            pinReadValue = Cy_GPIO_Read(CYBSP_REFERENCE_GPIO_PORT,
CYBSP_REFERENCE_GPIO_NUM);
            Cy_GPIO_Write(CYBSP_USER_LED_PORT, CYBSP_USER_LED_NUM, pinReadValue);
            Cy_SysLib_Delay(LED_DELAY_MS);
            /* Clear interrupt flag */
            interruptFlag = INTERRUPT_FLAG_CLEAR;
        }
        else
        {
            Cy_SysLib_Delay(LED_DELAY_MS);
        }
    }
```

6. Build the project and program the PSOC™ 4 device as shown in Figure 25

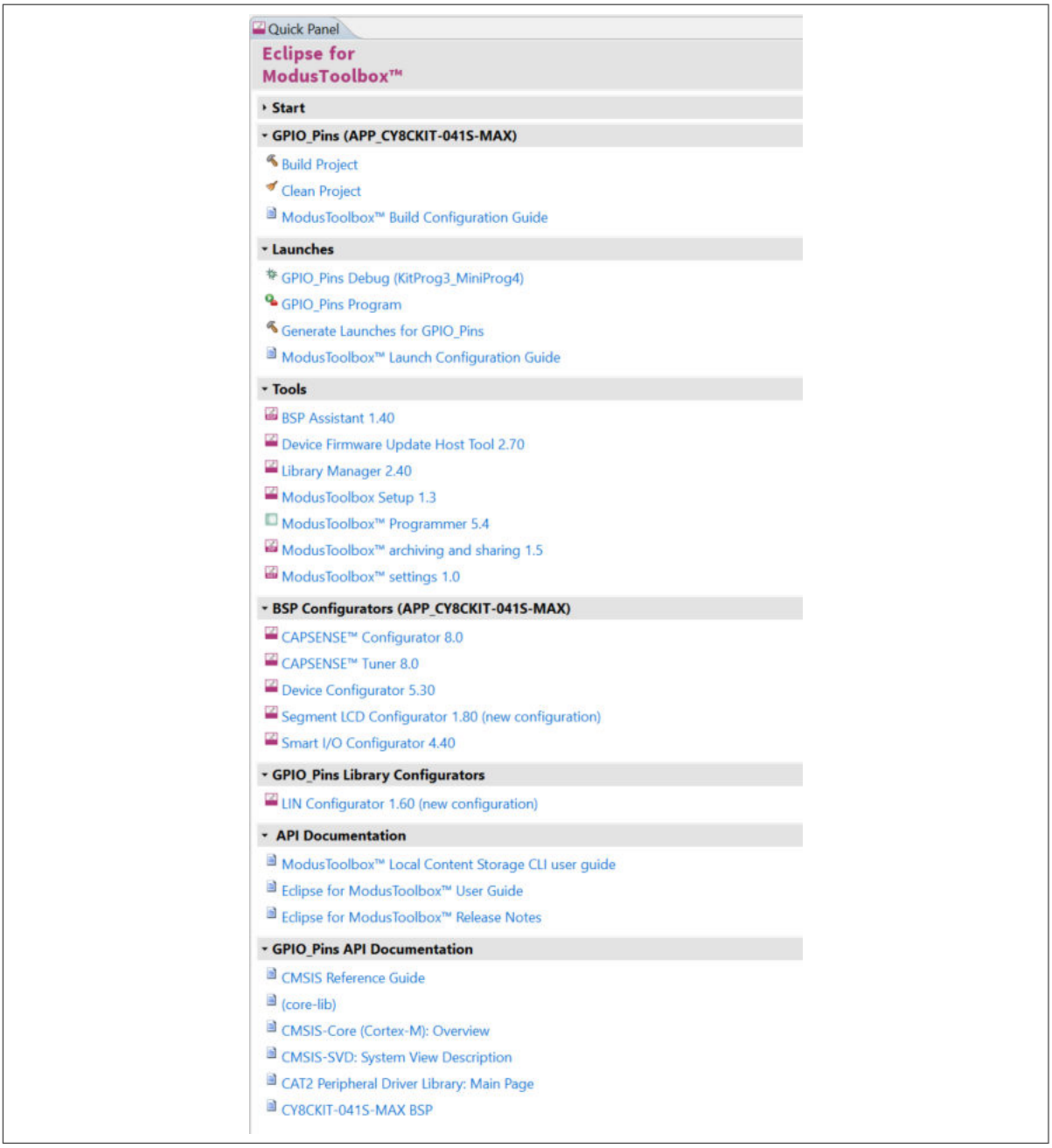


Figure 25 Build and Program the GPIO Pins project using Quick panel

7. After programming, the application starts automatically

- **a.** Confirm that the user LED toggles twice
- **b.** Connect the reference pin to logic HIGH externally using a jumper wire. Press the user button to observe the change in the state of the user LED. When the user button is pressed, the state of the reference pin is read and written to the user LED using the `Cy_GPIO_Write()` PDL API
- **c.** Now connect the reference pin to logic LOW. Press the user button to observe the change in the state of the user LED

The value on port 4 pins are incremented continuously and the toggling port pins can be monitored on an oscilloscope.

Code example CE231741 - PSOC™ 4: GPIO pins demonstrates multiple methods of configuring, reading, writing, and generating interrupts with PSOC™ 4 General Purpose Input/Output (GPIO) pins.

6.3 Using bidirectional pin

This example demonstrates the use of a pin in the bidirectional mode, that is, with both digital input and digital output active. ModusToolbox™ Device Configurator for Pins provides drive mode "Strong drive, input buffer on" with which bi-directional can be achieved.

1. Open the Project Creator GUI tool and on the **Choose Board Support Package (BSP)** page, select the kit followed by code example section as shown in section ModusToolbox™ code examples

   ***Note**: If the code example is not present in the list of Project Creator, select "Empty PSOC™ 4 App" inside "Getting started" and follow these steps.*

2. Open Device Configurator and select Pin section as shown in section Configuring GPIO pins using ModusToolbox™ Device Configurator
3. Select the Pins, name the components and configure the drive mode as shown in Figure 26

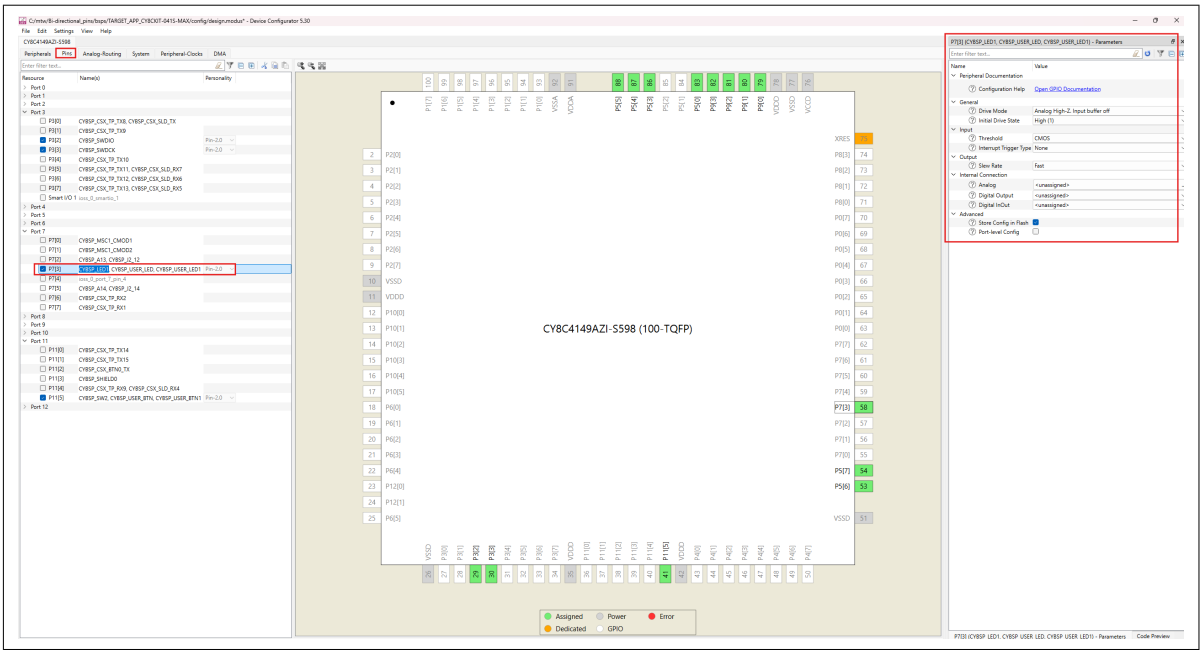


Figure 26 Device configuration for using bidirectional pin

4. Save and close the Device Configurator

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

```
for (;;)
{
    /* 1. Read the User Button state */
    /* Pin is pulled HIGH by internal resistor. Pressing switch connects it to GND (LOW)
*/
    uint32_t buttonState = Cy_GPIO_Read(CYBSP_USER_BTN_PORT, CYBSP_USER_BTN_PIN);
    bool switchIsPressed = (buttonState == 0);
    /* Adjust LED blink delay based on switch state */
    if (switchIsPressed)
    {
        if (blinkdelayMs != DELAY_SHORT)
        {
        /* Faster blink if switch is pressed */
            blinkdelayMs = DELAY_SHORT;
        }
    }
    else
    {
        if (blinkdelayMs != DELAY_LONG)
        {
            /* Slower blink if switch is not pressed */
            blinkdelayMs = DELAY_LONG;
        }
    }

    /* 2. "Output" phase for User LED Pin */
    /* Set pin to output mode (Strong Drive) to control the LED */
    /* The User LED on the kit is Active LOW */
    Cy_GPIO_SetDrivemode(CYBSP_USER_LED_PORT, CYBSP_USER_LED_PIN,
CY_GPIO_DM_STRONG_IN_OFF);
    /* Toggle the LED's blink state */
    ledON = !ledON;
    if (ledON)
    {
        /* Drive LOW to turn LED ON */
        Cy_GPIO_Clr(CYBSP_USER_LED_PORT, CYBSP_USER_LED_PIN);
    }
    else
    {
        /* Drive HIGH to turn LED OFF */
        Cy_GPIO_Set(CYBSP_USER_LED_PORT, CYBSP_USER_LED_PIN);
    }

    /* 3. "Input" demonstration phase for User LED Pin */
    /* Switch the LED pin to input mode (High-Z) to demonstrate it can be read */
    Cy_GPIO_SetDrivemode(CYBSP_USER_LED_PORT, CYBSP_USER_LED_PIN, CY_GPIO_DM_HIGHZ);
    /* Short delay for mode transition */
    Cy_SysLib_DelayUs(10);
        uint32_t ledReadValue = Cy_GPIO_Read(CYBSP_USER_LED_PORT, CYBSP_USER_LED_PIN);
        // This read_value is not an independent external input here, but demonstrates
        // that the pin's mode was changed and it was read.
```

```
        /* 4. Delay for the current LED blink state */
        Cy_SysLib_Delay(blinkdelayMs);
    }
```

6. Build the project and program the PSOC™ 4 device
7. After programming, the application starts automatically
   a. The User LED will start blinking. The initial blink rate will be relatively slow (e.g., at 700 milliseconds). This is the "output phase" for the pin connected to the User LED
   b. Press and hold down the User Button. Observe the User LED again. You should see its blinking rate increase significantly. It will now blink much faster (e.g., at 150 milliseconds).This input from the User Button is used to change the behavior that controls the User LED. This shows an external input influencing the pin we are treating "bidirectionally."
   c. Release the User Button. Observe the User LED. Its blinking rate will return to the original slower speed. Internal "Bidirectional" Software Logic (Not Directly Visible):

The blink rate of the User LED changing (faster/slower) in direct response to you pressing and releasing the User Button. This sequence confirms that the code is running, the GPIO for the LED is functioning as an output, the GPIO for the button is functioning as an input, and the logic to alter behavior based on input is working. The underlying software switching of the LED pin's mode for a conceptual "read" phase is the part that fulfills the "bidirectional pin management" demonstration.

6.4 Toggle GPIOs faster with data registers

This example demonstrates how to achieve rapid GPIO pin toggling on a PSOC™ 4 device by directly writing to the Port Output Invert (DR_INV) data register, offering a highly efficient, atomic method for flipping a pin's state.

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. Open the Project Creator GUI tool and on the **Choose Board Support Package (BSP)** page, select the kit followed by code example section as shown in section ModusToolbox™ code examples

   ***Note**: If the code example is not present in the list of Project Creator, select "Empty PSOC™ 4 App" inside "Getting started" and follow these steps.*

2. Open Device Configurator and select Pin section as shown in section Configuring GPIO pins using ModusToolbox™ Device Configurator
3. Select the Pins, name the components and configure the drive mode as shown in Figure 27

6 GPIO recommendations in ModusToolbox™ software

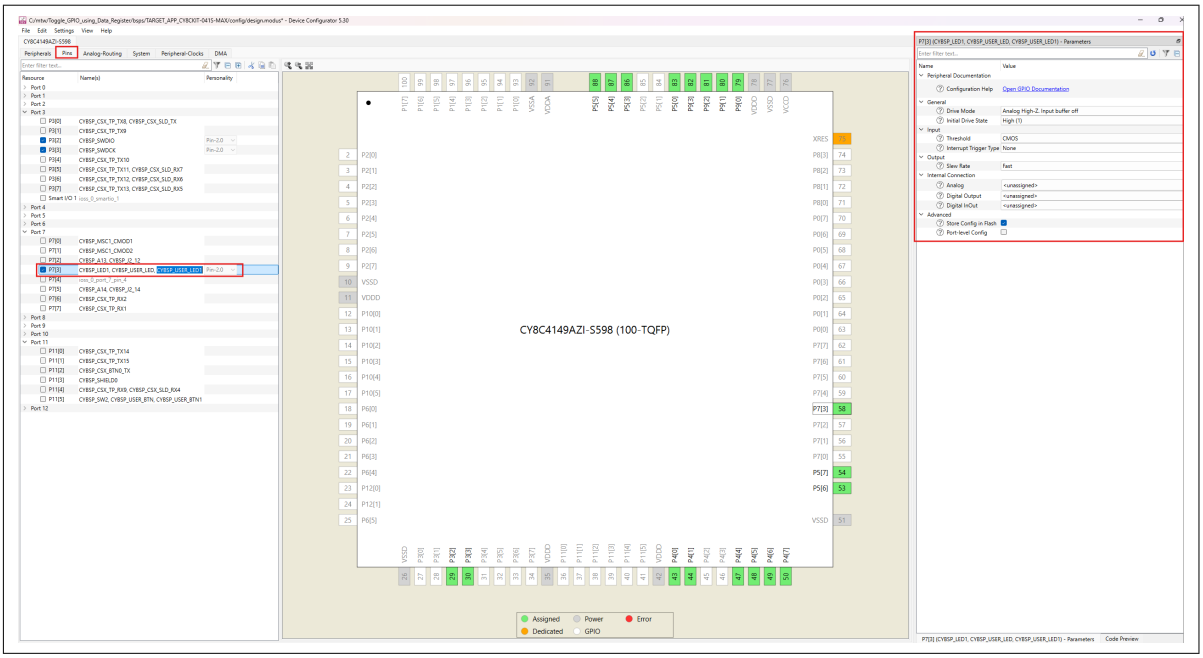


Figure 27 Device configuration for toggle GPIO using data registers CE

4. Save and close the Device Configurator
5. In *main.c*, use the Component API functions to set the output as follows:

```
for (;;)
{
    /* Faster GPIO Toggling using Port Output Invert Register (DR_INV)
     * Writing a '1' to a bit in the DR_INV register inverts the
     * current state of the corresponding output pin. This is an atomic
     * operation and typically the most efficient way to toggle a single pin.
     * It avoids read-modify-write sequences on the main output (DR) register.
     */
    CYBSP_USER_LED_PORT->DR_INV = (1UL << CYBSP_USER_LED_PIN);
    /* Optional delay for visual observation.
     * For measuring maximum toggle frequency with an oscilloscope,
     * this delay should be removed or made extremely small (e.g., a few NOPs).
     * With the delay, the LED will blink. Without it, the LED might appear
     * dimly lit or at half-brightness due to the very high toggle speed.
     */
    #if (TOGGLE_DELAY_MS > 0)
        Cy_SysLib_Delay(TOGGLE_DELAY_MS);
    #else
        // For maximum speed, you might insert a few NOP instructions
        // if the loop overhead itself is still too high, though on modern
        // MCUs, the register write itself is very fast.
        // __NOP(); __NOP(); __NOP(); __NOP();
    #endif
}
```

6. Build the project and program the PSOC™ 4 device
7. After programming, the application starts automatically. The User LED will start blinking very rapidly

If the TOGGLE_DELAY_MS in the code is set to a small value (like 1ms as in the example), the blinking will be fast but likely still perceptible; if the delay is removed entirely for speed testing, the LED might appear constantly on but

dimmer, or at half-brightness, due to toggling faster than the eye can see, which would be best observed as a high-frequency square wave on an oscilloscope.

6.5 Pin interrupt (GPIO interrupt)

This code example demonstrates the use of a GPIO configured as an input pin to generate interrupts on PSOC™ 4. The GPIO signal interrupts the CPU and executes a user-defined interrupt service routine (ISR). The GPIO interrupt acts as a wake up source to wake the CPU from Deep Sleep.

1. Open the Project Creator GUI tool and on the **Choose Board Support Package (BSP)** page, select the kit followed by code example section as shown in section ModusToolbox™ code examples

   ***Note**: If the code example is not present in the list of Project Creator, select "Empty PSOC™ 4 App" inside "Getting started" and follow these steps.*

2. Open Device Configurator and select Pin section as shown in section Configuring GPIO pins using ModusToolbox™ Device Configurator
3. Select the Pins, name the components and configure the drive mode as shown in Figure 28

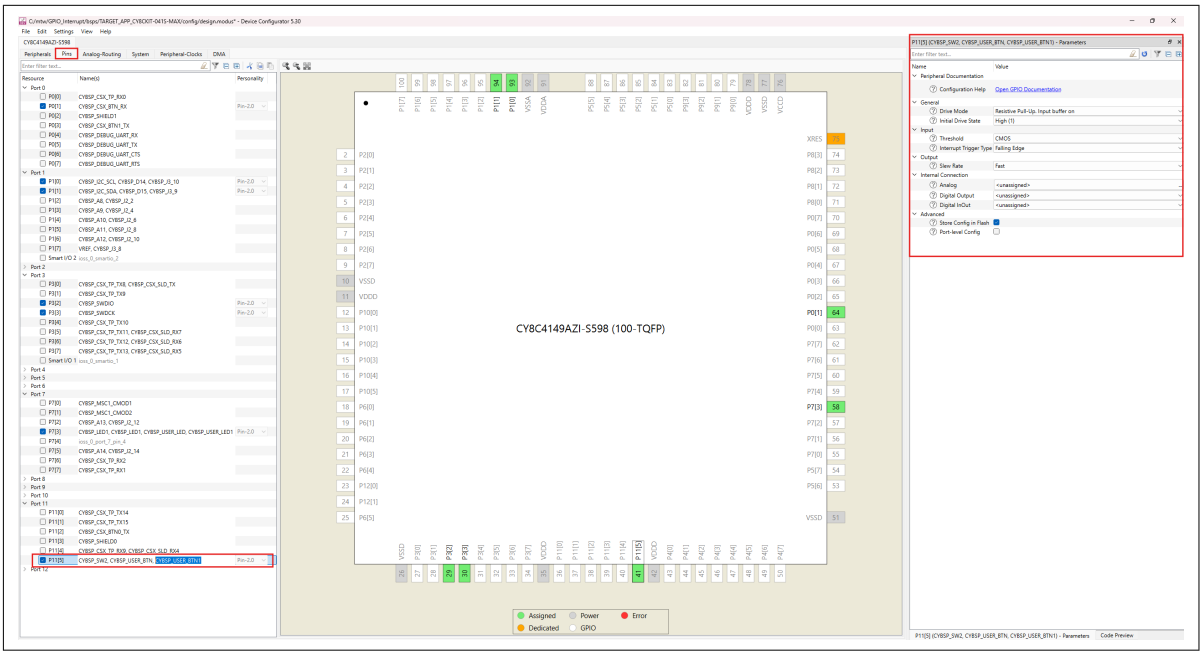


Figure 28 Device configuration for GPIO interrupt CE

4. Save and close the Device Configurator

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

```
int main(void)
{
    cy_rslt_t result;
    uint32_t count = 0;
    uint32_t delayMs = DELAY_LONG;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize and enable GPIO interrupt */
    result = Cy_SysInt_Init(&switch_intr_config, Switch_ISR);
    if(result != CY_SYSINT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Clearing and enabling the GPIO interrupt in NVIC */
    NVIC_ClearPendingIRQ(switch_intr_config.intrSrc);
    NVIC_EnableIRQ(switch_intr_config.intrSrc);

    cy_stc_sysclk_context_t sysClkContext;

    cy_stc_syspm_callback_params_t sysClkCallbackParams =
    {
        .base       = NULL,
        .context    = (void*)&sysClkContext
    };

    /* Callback declaration for Deep Sleep mode */
    cy_stc_syspm_callback_t sysClkCallback =
    {
        .callback       = &Cy_SysClk_DeepSleepCallback,
        .type           = CY_SYSPM_DEEPSLEEP,
        .skipMode       = 0UL,
        .callbackParams = &sysClkCallbackParams,
        .prevItm        = NULL,
        .nextItm        = NULL,
        .order          = 0
    };

    /* Register Deep Sleep callback */
    Cy_SysPm_RegisterCallback(&sysClkCallback);

    for (;;)
```

```
    {
        /* Check the interrupt status */
        if(true == interrupt_flag)
        {
            interrupt_flag = false;

            if(DELAY_LONG == delayMs)
            {
                delayMs = DELAY_SHORT;
            }
            else
            {
                delayMs = DELAY_LONG;
            }
        }

        /* Blink LED four times */
        for (count = 0; count < LED_BLINK_COUNT; count++)
        {
            Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_NUM);
            Cy_SysLib_Delay(delayMs);
            Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_NUM);
            Cy_SysLib_Delay(delayMs);
        }

        /* Enter deep sleep mode */
        Cy_SysPm_CpuEnterDeepSleep();

    }
}
```

6. Build the project and program the PSOC™ 4 device
7. After programming, the application starts automatically
   - **a.** Observe that the LED blinks four times and then turns OFF, indicating that the CPU has entered Deep Sleep
   - **b.** Press the user button to trigger an interrupt. This should wake up the device, causing the LED to resume blinking at a faster rate (default = 2 Hz). The faster blink rate indicates that the ISR has executed to change the delay variable used to determine the LED blink rate. The LED blinks four times and the device enters Deep Sleep again
   - **c.** Press the button again to repeat the wake up cycle. The LED resumes blinking at a slower rate (default = 1 Hz). With every interrupt and execution of the ISR, the interval of blinking is alternated between slower and faster rates

This code example uses a GPIO interrupt to wake the CPU from Deep Sleep. An LED is connected to an output pin; it is used for indicating the current state of the CPU. A blinking LED indicates that the CPU is active. After four successive blinks, the CPU is instructed to enter Deep Sleep. The GPIO state is retained during Deep Sleep, so the LED stops blinking and stays "OFF" to indicate that the CPU is in Deep Sleep.

An input pin, externally connected to a switch, is configured to generate an interrupt when the switch is pressed. The interrupt triggers the following two actions:

1. Generates a signal that wakes the CPU from Deep Sleep
2. Executes an ISR

When the ISR is executed, a flag is updated, which is used to change the rate of blinking the LED. With every press of the switch, the LED alternates the blinking interval.

See the *CE230654 - PSOC™ 4: GPIO interrupt* code example.

6.6 Configure GPIO interrupt settings with firmware

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

In this example, User 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. Open the Project Creator GUI tool and on the **Choose Board Support Package (BSP)** page, select the kit followed by code example section as shown in section ModusToolbox™ code examples

   ***Note**: If the code example is not present in the list of Project Creator, select "Empty PSOC™ 4 App" inside "Getting started" and follow these steps.*

2. Open Device Configurator and select Pin section as shown in section Configuring GPIO pins using ModusToolbox™ Device Configurator
3. Select the Pins, name the components and configure the drive mode as shown in Figure 29

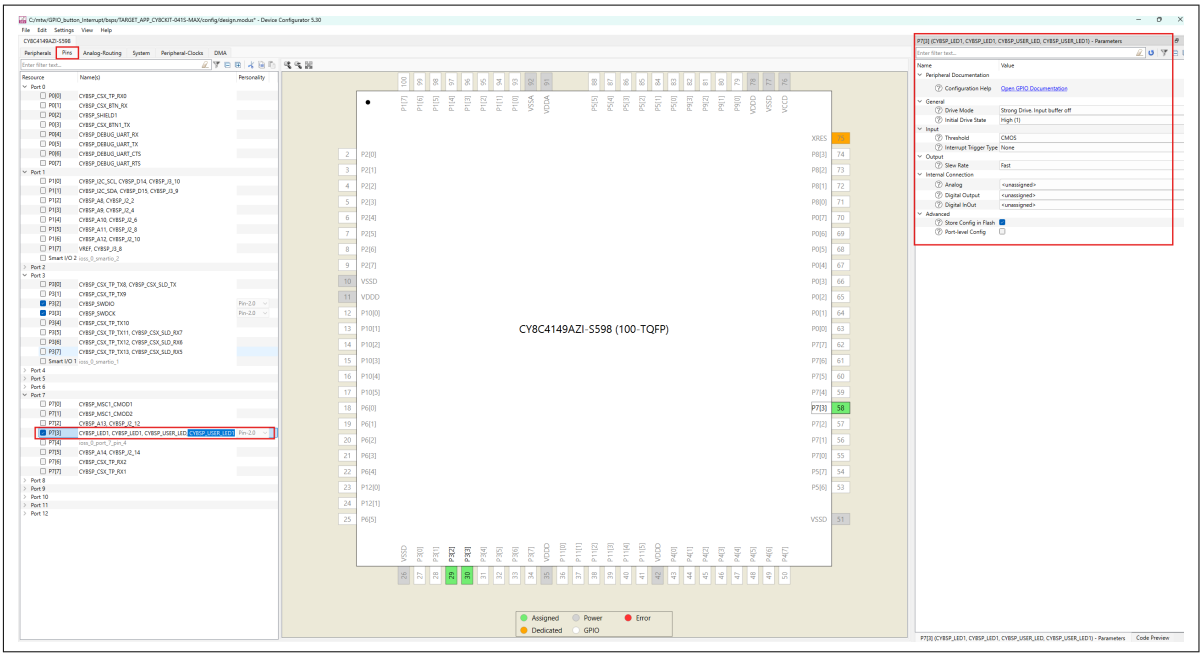


Figure 29 Device configuration for GPIO button interrupt CE

4. Save and close the Device Configurator

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

```
/*******************************************************************************
* Macros
*******************************************************************************/
#define BUTTON_INTR_PRIORITY    (3U)

// GPIO_PRTx_INTR_CFG register edge selection values (for EDGEx_SEL bits)
#define EDGE_SEL_DISABLED       (0UL) // 00b
#define EDGE_SEL_RISING         (1UL) // 01b
#define EDGE_SEL_FALLING        (2UL) // 10b
#define EDGE_SEL_BOTH           (3UL) // 11b


/*******************************************************************************
* Global Variables
*******************************************************************************/
static bool current_edge_is_rising = true; // Start with rising edge

/*******************************************************************************
* Function Prototypes
*******************************************************************************/
static void Button_Interrupt_Handler(void);
static void Configure_Button_Interrupt_Edge(uint32_t edge_sel_value);

/*******************************************************************************
* Interrupt Configuration Structure
*******************************************************************************/
const cy_stc_sysint_t button_intr_config =
{
    .intrSrc = CYBSP_USER_BTN_IRQ,          // Interrupt source (defined by BSP)
    .intrPriority = BUTTON_INTR_PRIORITY
};

/*******************************************************************************
* Function Name: Button_Interrupt_Handler
*******************************************************************************/
static void Button_Interrupt_Handler(void)
{
    // Clear the specific pin interrupt flag in the GPIO port's INTR register
    Cy_GPIO_ClearInterrupt(CYBSP_USER_BTN_PORT, CYBSP_USER_BTN_NUM);

    // Clear the pending interrupt in the NVIC
    NVIC_ClearPendingIRQ(button_intr_config.intrSrc);

    // Toggle the User LED
    Cy_GPIO_Inv(CYBSP_USER_LED_PORT, CYBSP_USER_LED_NUM);

    // Dynamically reconfigure the interrupt edge
    if (current_edge_is_rising)
    {
        // Was rising edge, change to falling edge
        Configure_Button_Interrupt_Edge(EDGE_SEL_FALLING);
```

```
            current_edge_is_rising = false;
        }
        else
        {
            // Was falling edge, change to rising edge
            Configure_Button_Interrupt_Edge(EDGE_SEL_RISING);
            current_edge_is_rising = true;
        }
    }

    /*******************************************************************************
    * Function Name: Configure_Button_Interrupt_Edge
    *******************************************************************************/
    static void Configure_Button_Interrupt_Edge(uint32_t edge_sel_value)
    {
        uint32_t pin_num = CYBSP_USER_BTN_NUM;
        GPIO_PRT_Type* port = CYBSP_USER_BTN_PORT;

        // Calculate the bit position for the EDGEx_SEL field for this pin.
        // Each pin's edge select is 2 bits. Pin 0 is bits 1:0, Pin 1 is bits 3:2, etc.
        uint32_t shift = pin_num * 2;
        uint32_t mask = (0x3UL << shift); // Mask for the 2 bits

        // Read the current INTR_CFG register value
        uint32_t reg_val = port->INTR_CFG;

        // Clear the existing edge selection bits for this pin
        reg_val &= ~mask;

        // Set the new edge selection bits
        reg_val |= (edge_sel_value << shift);

        // Write the modified value back to the INTR_CFG register
        port->INTR_CFG = reg_val;
    }

    int main(void)
    {
        cy_rslt_t result;

        /* Initialize the device and board peripherals */
        result = cybsp_init() ;
        if (result != CY_RSLT_SUCCESS)
        {
            CY_ASSERT(0);
        }

        /* Enable global interrupts */
        __enable_irq();

        Cy_GPIO_Write(CYBSP_USER_LED_PORT, CYBSP_USER_LED_NUM, 1);
        result = Cy_SysInt_Init(&button_intr_config, Button_Interrupt_Handler);
        if (result != CY_SYSINT_SUCCESS)
```

```
        {
            CY_ASSERT(0); // Interrupt init failed
        }

        /* Initially configure for rising edge interrupt */
        current_edge_is_rising = true; // Redundant, but for clarity
        Configure_Button_Interrupt_Edge(EDGE_SEL_RISING);

        /* Enable the interrupt in the NVIC */
        NVIC_ClearPendingIRQ(button_intr_config.intrSrc);
        NVIC_EnableIRQ(button_intr_config.intrSrc);

        for (;;)
        {
        }
    }

    /* [] END OF FILE */
```

6. Build the project and program the PSOC™ 4 device
7. After programming,
   a. **Initial State:** The User LED will be OFF
   b. **First Button Press (Press and Hold):** When you press the User Button, the pin goes LOW. Nothing happens yet because it is waiting for a rising edge
   c. **First Button Release (Rising Edge):** When you release the User Button, the pin goes HIGH (due to the pull-up)
      - The **User LED will toggle** (e.g., turn ON)
      - Internally, the interrupt is now reconfigured for a **falling edge**
   d. **Second Button Press (Falling Edge):** When you press the User Button again, the pin goes LOW
      - The **User LED will toggle** (e.g., turn OFF)
      - Internally, the interrupt is now reconfigured for a **rising edge**
   e. **Second Button Release:** When you release the User Button, the pin goes HIGH
      - The **User LED will toggle** (e.g., turn ON)
      - Internally, the interrupt is reconfigured for a **falling edge**
   f. This cycle will continue: the LED will toggle on each press (falling edge) and then again on each release (rising edge) as the interrupt configuration dynamically changes

This example directly addresses the requirement of dynamically configuring the interrupt edge by writing to the relevant bits in the GPIO port's interrupt configuration register.

6.7 Clock buffer in Smart I/O

This example shows how to use a smart I/O peripheral in PSOC™ 4 device to implement a clock buffer that can operate in chip low-power modes. It can also be used to drive a heavier load than one GPIO is rated for by replicating the signal and driving two pins.

1. Open the Project Creator GUI tool and on the **Choose Board Support Package (BSP)** page, select the kit followed by code example section as shown in section ModusToolbox™ code examples

   ***Note**: If the code example is not present in the list of Project Creator, select "Empty PSOC™ 4 App" inside "Getting started" and follow these steps.*

2. Open Device Configurator and select Pin section as shown in section Configuring GPIO pins using ModusToolbox™ Device Configurator
3. Select the Pins, name the components and configure the drive mode as shown in Figure 30

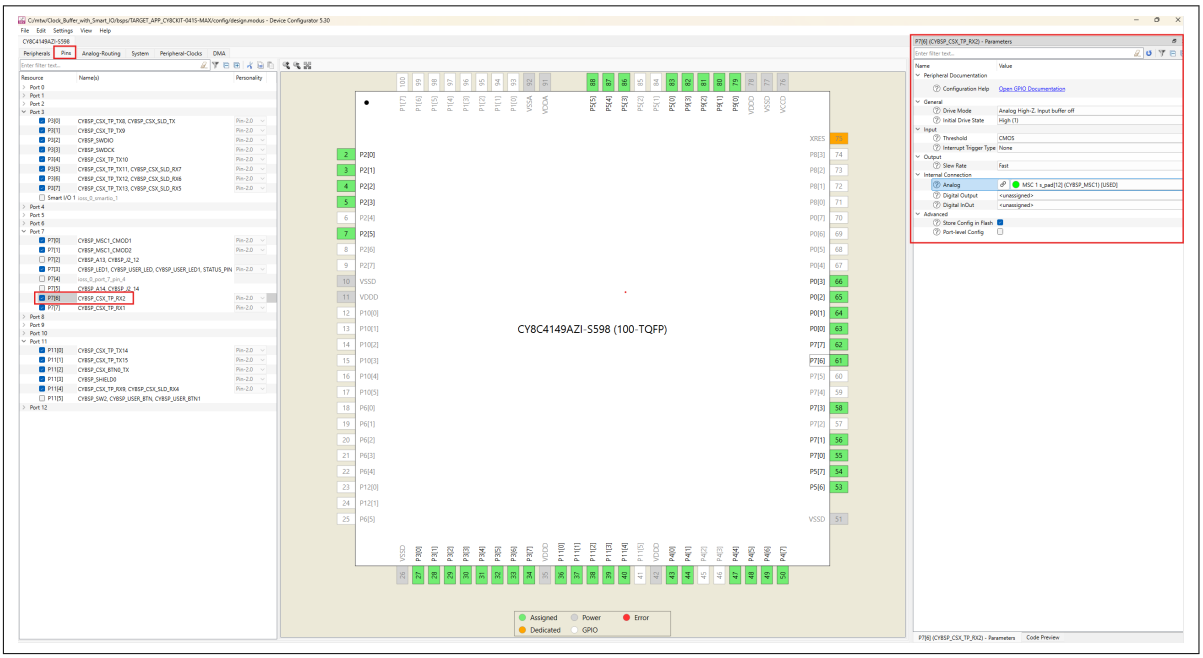


Figure 30 Device configuration for clock buffer with Smart I/O CE

4. Save and close the Device Configurator
5. Open Smart I/O Configurator and select Pin section as shown in Figure 31

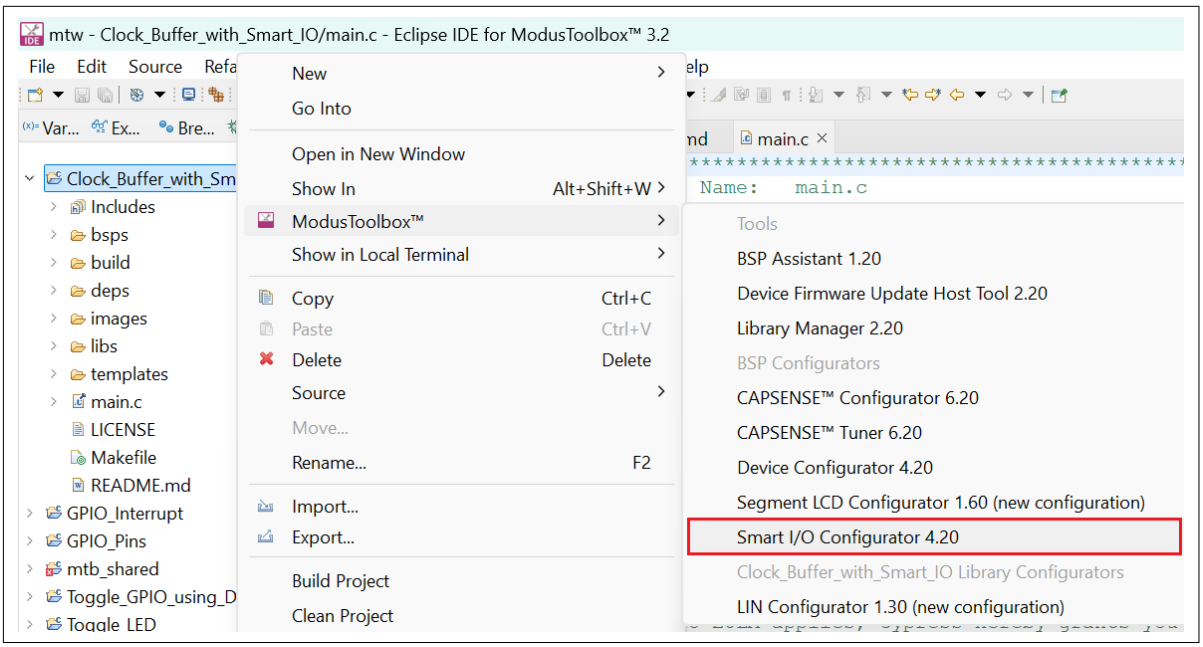


Figure 31 Smart I/O configuration

6. Configure Smart I/O as shown in Figure 32

6 GPIO recommendations in ModusToolbox™ software

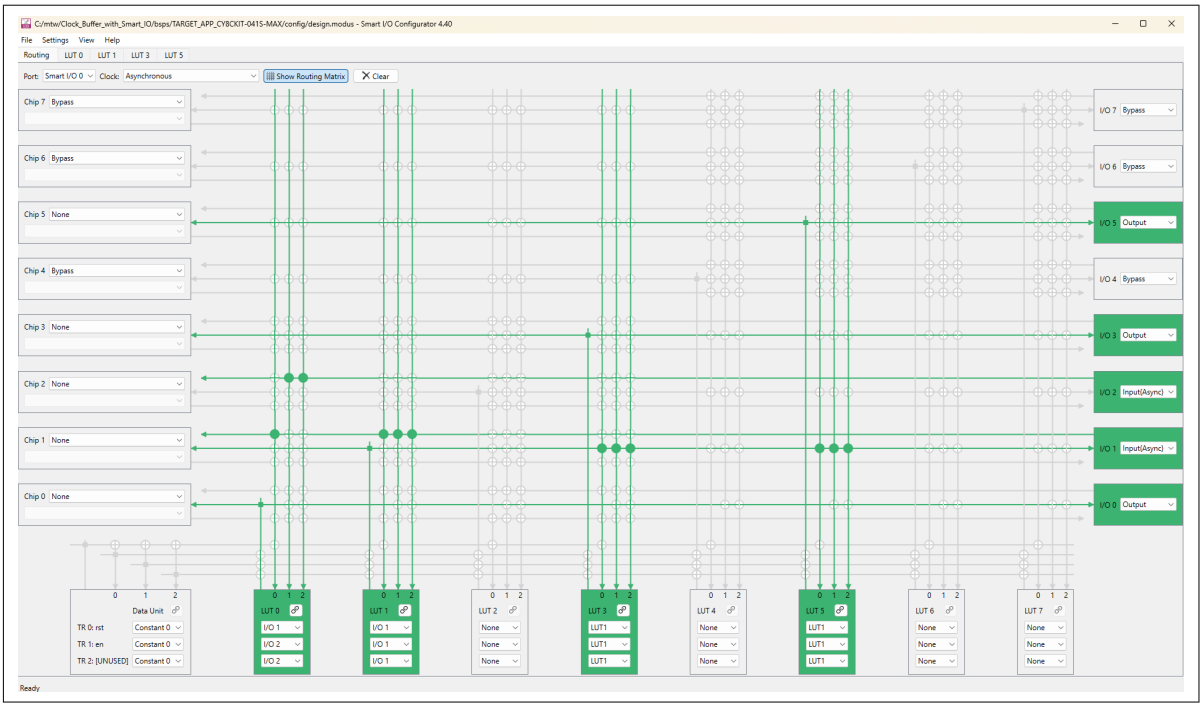


Figure 32 Smart I/O configuration for clock buffer with Smart I/O CE

7. Save and close the Smart I/O Configurator

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

```
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize and enable GPIO interrupt */
    result = Cy_SysInt_Init(&switch_intr_config, switch_isr);

    if(result != CY_SYSINT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Clearing and enabling the GPIO interrupt in NVIC */
    NVIC_ClearPendingIRQ(switch_intr_config.intrSrc);
    NVIC_EnableIRQ(switch_intr_config.intrSrc);

    smart_io_start();

    for (;;)
    {
        /* Hold override functionality must be enabled before entering deep-sleep */
        Cy_SmartIO_HoldOverride(PRGIO_PRT0, true);
        Cy_SysPm_CpuEnterDeepSleep();
        Cy_SmartIO_HoldOverride(PRGIO_PRT0, false);
        if (interrupt_flag == 1u)
        {
            interrupt_flag = 0u;
            Cy_GPIO_Write(STATUS_PIN_PORT, STATUS_PIN_NUM, 0UL);
            Cy_SysLib_Delay(ONE_SEC_DELAY);
            Cy_GPIO_Write(STATUS_PIN_PORT, STATUS_PIN_NUM, 1UL);
        }
    }
}
```

9. Build the project and program the PSOC™ 4 device
10. After programming,
    a. Observe that the signal going into IN_PIN is replicated on OUT_PIN_0 and OUT_PIN_1. These are operational during deep-sleep mode; the two signals can be externally ganged to drive a higher load
    b. Connect TRIG_PIN to VDD and then connect it back to GND. Alternatively push the external active high switch connected to TRIG_PIN. Observe that the LED connected to STATUS_PIN lights up for approximately 1 second. Observe that the OUT_PIN_0 and OUT_PIN_1 continue to operate regardless of chip power mode

The design consists of a smart I/O peripheral with only the pins on the port as its inputs and outputs. It does not use any peripherals or internal chip signals. The peripheral performs a signal replication function by taking in an external signal (such as an external clock) and driving it out to two pins. This effectively implements a signal buffer functionality. The two pins can then be ganged externally to the chip to drive a load that is higher than rated for a single GPIO pin.

The Smart I/O peripheral is operational during the device's deep-sleep mode. This design shows how multiple signals can be used in the smart I/O peripheral to trigger an interrupt that is the result of a logical operation on those signals.

For the clock buffer, an external signal is input through I/O 1, which in turn is connected to IN_PIN. This signal is repeated and output to I/O 3 and I/O 5. These are connected to OUT_PIN_0 and OUT_PIN_1 respectively.

See *CE236713 – PSOC™ 4: Clock buffer with smart I/O* code example.

6.8 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.

7 PSOC™ 4 development boards

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

***Note**: Few of the below mentioned development boards are supported by ModusToolbox™ and more will keep on adding. For more details contact Infineon support.*

Device Family	Development board
PSOC™ 4000	CY8CKIT-040 PSOC™ 4000 Pioneer Development Kit
PSOC™ 4000S/4100S	CY8CKIT-145-40XX PSOC™ 4 S-Series prototyping Kit
PSOC™ 4100S Plus	CY8CKIT-149 PSOC™ 4100S Plus prototyping Kit
PSOC™ 4100S MAX	CY8CKIT-041S-MAX PSOC™ 4100S Max prototyping Kit
PSOC™ 4200/PSOC™ 4100	CY8CKIT-042 PSOC™ 4 Pioneer Kit
PSOC™ 4200M	CY8CKIT-044 PSOC™ 4 M-Series Pioneer Kit
PSOC™ 4200L	CY8CKIT-046 PSOC™ 4 L-Series Pioneer Kit
PSOC™ 4 HV MS	KIT_PSOC4-HVMS-64K_LITE and KIT_PSOC4-HVMS-128K_LITE

References

Application notes

[1] Infineon Technologies AG: *AN88619 - PSOC™ 4 MCU hardware design considerations*; Available online
[2] Infineon Technologies AG: *AN89610 - PSOC™ Arm® Cortex® code optimization*; Available online
[3] Infineon Technologies AG: *AN86233 - PSOC™ 4 MCU low-power modes and power reduction techniques*; Available online
[4] Infineon Technologies AG: *AN57821 - Mixed-signal circuit board layout considerations*; Available online
[5] Infineon Technologies AG: *AN89056 - PSOC™ IEC 60730 class B and IEC 61508 SIL Safety Software Library*; Available online
[6] Infineon Technologies AG: *AN64846 - Getting started with CAPSENSE™*; Available online
[7] Infineon Technologies AG: *AN85951 - PSOC™ 4 and PSOC™ 6 MCU CAPSENSE™design guide*; Available online
[8] Infineon Technologies AG: *AN84858 - PSOC™ 4 programming using an external microcontroller (HSSP)*; Available online

Development tools

[9] *Pioneer kits*: CY8CKIT-040, CY8CKIT-042, CY8CKIT-044, CY8CKIT-046, and CY8CKIT-041S-MAX PSOC™ 4 Pioneer kits are easy-to-use and inexpensive development platforms
[10] *Prototyping platforms*: CY8CKIT-043, CY8CKIT-145-40XX, and CY8CKIT-149 are very low-cost prototyping platforms for sampling PSOC™ 4 devices
[11] *Kits*: The MiniProg3 (CY8CKIT-002) or MiniProg4 (CY8CKIT-005) kit provides an interface for flash programming and debug
[12] *Integrated Development Environment (IDE)*: There are two development platforms that can be used for application development with PSOC™ 4 – ModusToolbox™
[13] *Libraries*: PSOC™ 4 CAD libraries provide footprint and schematic support for common tools. IBIS models are also available

Additional resources

[14] *PSOC™ 4 portfolio*; Available online
[15] *Datasheets*: Describe and provide electrical specifications for each family; Available online
[16] *Code examples*: To demonstrate product features and usage; Available online
[17] *Reference manuals*: Provide detailed descriptions of the architecture and registers in each PSOC™ 4 device family; Available online
[18] *Programming specification*: PSOC™ 4 programming specification provides the information necessary to program PSOC™ 4 nonvolatile memory; Available online
[19] *Training videos*: Are available in Infineon website on a wide range of topics including the PSOC™ MCUs
[20] *Infineon Developer Community*: 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

***Note**: The wealth of information available on the Infineon webpage can help you select the right device and, additionally, integrate the PSOC™ device into your designs efficiently and effectively.*

Revision history

Document revision	Date of release	Description of changes
**	2025-09-03	Initial release

Trademarks

PSoC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSoC™.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-09-03

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-jha1742528712527

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.