AN72382

Using PSoC® 3 and PSoC 5LP GPIO Pins

Author: Greg Reynolds

Associated Part Family: All PSoC® 3 and PSoC 5LP parts

Software Version: PSoC Creator™ 2.1 SP1 and higher

Related Application Notes: For a complete list of related application notes, click here.

AN72382 shows you how to use GPIO pins effectively in PSoC® 3 and PSoC 5LP. Major topics include GPIO basics, configuration, mixed-signal use, registers, interrupts, and low-power behavior.

Contents

1 Introduction....1
2 GPIO Pin Basics....2
2.1 Physical Structure of GPIO Pins....2
2.2 Digital System Interconnect Overview....2
2.3 Analog Routing Overview....3
2.4 GPIO Power Structure and Limits....3
2.5 Relative Voltages of V_{DDA}, V_{DDD}, and V_{DDIO}....4
2.6 Startup and Low-Power Behavior....5
2.7 DMA Access to GPIO Pins....5
2.8 Port Interrupt Control Unit....5
3 GPIO Pins in PSoC Creator....6
3.1 PSoC Creator APIs....6
3.2 Pins Component Symbols and Macros....6
3.3 Pins Component Interrupts....6
3.4 External Terminals....8
3.5 Manual Pin Assignments....8
4 GPIO Examples, Tips, and Tricks....9
4.1 The GPIO “Hello World” Project....9
4.2 Read an Input and Write to an Output....9
4.3 Add Multiple GPIO Pins as a Logical Port....10
4.4 Configure GPIO Output Enable Logic....12
4.5 Enable the Configurable XRES Feature....13
4.6 Disable Debug Logic on GPIO Pins....14
4.7 Toggle GPIOs Faster with Data Registers....14
4.8 Use 8051 Special Function Registers....16
4.9 Use Both Analog and Digital on a GPIO....16
4.10 Control Analog Switching with Hardware....18
4.11 Use the DSI as a Clock Source....20
4.12 Change PICU Settings with Firmware....22
4.13 Gang Pins for More Drive/Sink Current....23
4.14 Level-Shift Signals....24
5 Related Application Notes....25
Appendix A. GPIO API and Register Reference....26
A.1 Component API....26
A.2 Per-Pin API....26
A.3 GPIO Registers....27
A.4 Nonvolatile Latches....28
Appendix B. PSoC Creator Settings and Registers....29
Document History....40
Worldwide Sales and Design Support....41

1 Introduction

The any-signal-to-any-pin routing available with PSoC® 3 and PSoC 5LP GPIOs helps to optimize PCB layout, shorten design time, and allow a large degree of solderless rework. However, with this freedom comes a steeper learning curve than with a traditional microcontroller. This application note introduces you to PSoC 3 and PSoC 5LP GPIO basics and demonstrates techniques for their effective use in a design.

It is assumed that you are familiar with PSoC Creator™ and the PSoC 3 and PSoC 5LP family device architecture. If you are new to PSoC, see the introductions in AN54181 – Getting Started with PSoC 3 and AN77759 – Getting Started with PSoC 5LP. If you are new to PSoC Creator, see the PSoC Creator home page.

For a list of related PSoC design resources, see the Related Application Notes section.

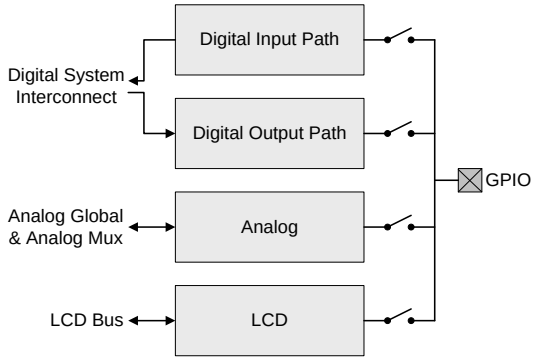
2 GPIO Pin Basics

In PSoC 3 and PSoC 5LP devices, the GPIO, SIO, and USB pins are similar. Unlike GPIO pins, though, the SIO and USB pins have different drive strengths and application-specific features. Some GPIO pins also have secondary dedicated functions, such as opamp inputs and outputs, programming and debugging interfaces, or DAC outputs. When they are not being used for special functions, all GPIO pins behave the same. Depending on the package type, PSoC devices can have as many as 62 GPIO pins.

2.1 Physical Structure of GPIO Pins

GPIO pins have eight drive modes to support the many analog and digital I/O capabilities that PSoC offers. A detailed block diagram of the GPIO structure appears in the PSoC 3 Architecture Technical Reference Manual (TRM), as well as in the PSoC 3 and PSoC 5LP family datasheets. Figure 1 shows a simplified version.

Figure 1. Simplified GPIO Block Diagram

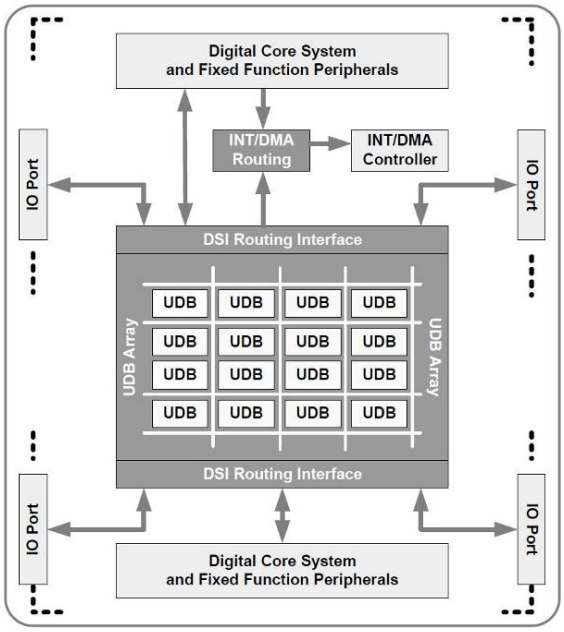


The various drive modes and their custom settings are described in detail in the Pins Component datasheet, which is available as part of PSoC Creator or as a separate download from the Cypress website.

2.2 Digital System Interconnect Overview

The PSoC 3 and PSoC 5LP digital subsystem has a programmable interconnect that allows connections between the built-in peripherals, custom logic functions (universal digital blocks, or UDBs), and any I/O pin. The digital system interconnect (DSI) routing interface allows GPIO pins to connect to any digital resource in the chip, as Figure 2 illustrates.

Figure 2. DSI Block Diagram

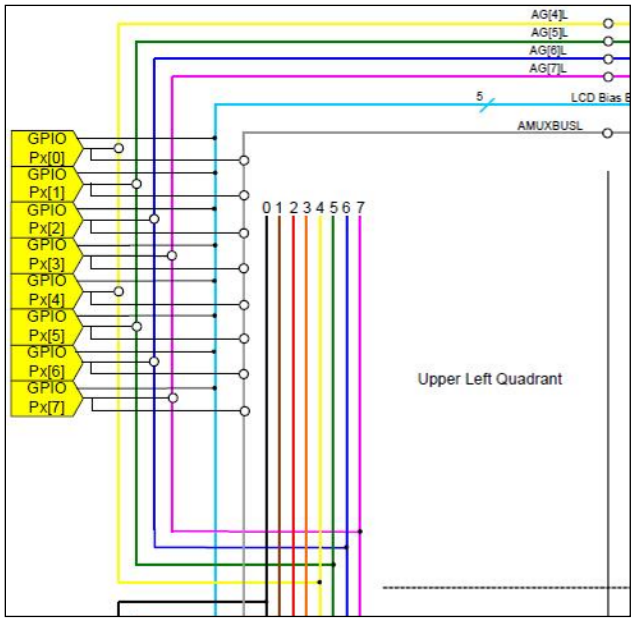


All digital resources are routed to the DSI for connection to each other or to the system core. For more details about the DSI operation, see the “UDB Array and Digital System Interconnect” section of the TRM.

2.3 Analog Routing Overview

The GPIO pins are connected to analog resources, or to each other, through a series of analog routing buses joined by switches and muxes. The two primary analog routing buses are the analog global (AG) bus and the analog mux (AMUX) bus. The AG bus is divided into four quadrants (AGL0-4, AGL4-7, AGR0-4, and AGR4-7), and the AMUX bus is divided into two halves (AMUXL and AMUXR). Figure 3 shows a portion of the analog routing diagram from the TRM.

Figure 3. Upper-Left Analog Routing Quadrant



Each AGx can connect to two of the pins on an associated port in each quadrant, while each AMUX can connect to every pin on its half of the chip. The analog buses also connect to the inputs and/or outputs of various analog resources, such as comparators, DACs, and ADCs. In addition, switches allow the left and right buses to be connected to each other.

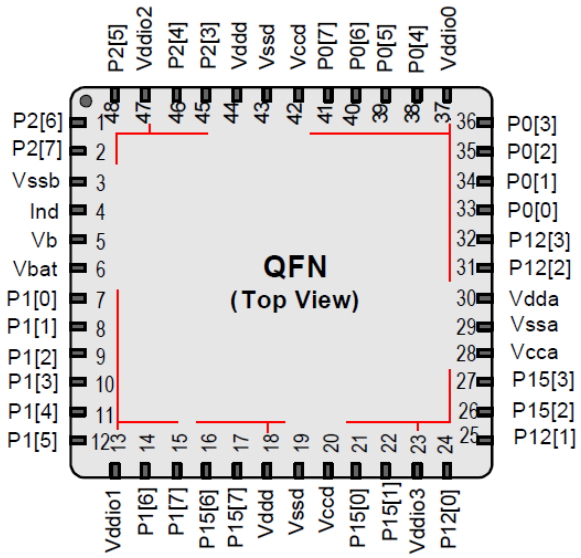
An in-depth description of the analog routing system in the PSoC 3 and PSoC 5LP devices is included in the “Analog Routing” section of the TRM. Application notes AN58304 and AN58827 discuss analog routing and pin selection in detail.

2.4 GPIO Power Structure and Limits

In general, GPIO pins can source 4 mA and sink 8 mA. They can be ganged together (shorted) to allow more current to be sourced or sunk than that which a single pin can provide, but you need to consider additional power limitations.

PSoC 3 and PSoC 5LP devices provide as many as four individual I/O voltage domains through the V_{DDIO} pins. In the PSoC 3 and PSoC 5LP datasheets, the V_{DDIO} pin that supplies power to a particular set of pins is indicated by solid lines drawn on the pinout diagrams. Figure 4 shows a 48-pin PSoC 3 device with the V_{DDIO} quadrant indicators highlighted in red.

Figure 4. Example V_{DDIO} Quadrants Highlighted in Red



The V_{DDIO} pins are often tied to the same power rail as V_{DD}. Little thought is given to how much current any individual V_{DDIO} quadrant is sourcing and sinking, but there are limitations. Table 1 shows the limits according to PSoC family and package type.

Table 1. V_{DDIO} Quadrant Current Limits

Family	Package	Source	Sink
PSoC 3 PSoC 5LP	100-pin 68-pin	100 mA per V_{DDIO}	100 mA per V_{DDIO}
	48-pin	100 mA V_{DDIO0}+V_{DDIO2} 100 mA V_{DDIO1}+V_{DDIO3}	100 mA V_{DDIO0}+V_{DDIO2} 100 mA V_{DDIO1}+V_{DDIO3}
PSoC 5	100-pin 68-pin	20 mA per V_{DDIO}	20 mA per V_{DDIO}
	48-pin	20 mA V_{DDIO0}+V_{DDIO2} 20 mA V_{DDIO1}+V_{DDIO3}	20 mA V_{DDIO0}+V_{DDIO2} 20 mA V_{DDIO1}+V_{DDIO3}
Note: Total source+sink current should not exceed 100 mA for any V_{DDIO} quadrant (or V_{DDIO} pair for the 48-pin packages).			

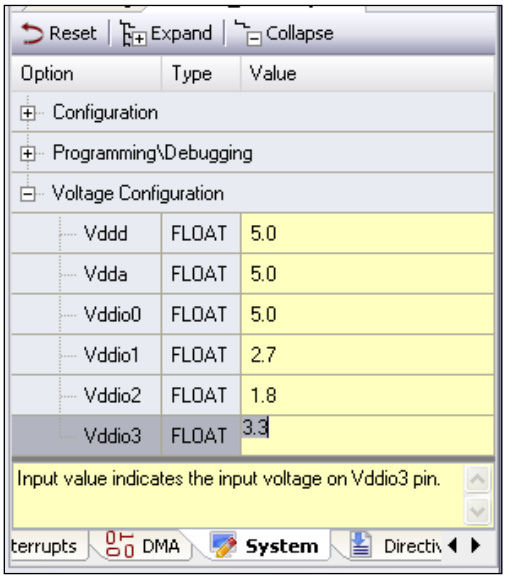
In applications for which the typical current sourced and sunk by the GPIO pins is expected to exceed 80 percent of the limit, make sure that no single quadrant of GPIO pins exceeds its maximum under the worst operating conditions. Doing so may mean that the design needs to use pins in separate V_{DDIO} quadrants to spread out the current.

2.5 Relative Voltages of V_{DDA}, V_{DDD}, and V_{DDIO}

V_{DDA} must be at the highest voltage present on the PSoC 3 or PSoC 5LP device. All other power supply pins must be less than or equal to V_{DDA}. The V_{DDD} and V_{DDIO} pins may be less than, greater than, or equal to each other.

The **System** tab of the Design-Wide Resources file includes a voltage configuration section that lets you define the voltage at which each power domain will operate. The values entered in these fields, shown in Figure 5, are used by PSoC Creator if a Component or feature is dependent on the voltage at which it is running.

Figure 5. Voltage Configuration in Design-Wide Resources



Proper voltage configuration in PSoC Creator is recommended in all cases, regardless of which Components or features are used.

2.6 Startup and Low-Power Behavior

Out of the box, all GPIO pins start up in an Analog HI-Z state, where they remain until reset is released. The initial operating configuration of each pin is loaded during boot and takes effect at that time. You can change the reset behavior of GPIOs using the PRTxRDM fields of the nonvolatile latch array, which are written when the PSoC device is programmed.

In all low-power modes, GPIO pins retain their state until the part is reset or awakened. The port interrupt logic continues to function in all low-power modes so that pins can be used as wakeup sources.

Note UDB-based Components, such as control registers, are typically not active during sleep or hibernate. They can glitch when the PSoC device enters or exits these modes. The glitch could cause a GPIO to be set at an unwanted state. To avoid that, set the pins explicitly to a HIGH or LOW logic state before the PSoC device enters a low-power mode.

2.7 DMA Access to GPIO Pins

PSoC devices have a DMA controller that connects to different internal peripherals, including the I/O interface. Because GPIO registers are memory-addressed, DMA transfers can be used to configure GPIO pins and write data to the digital output path without requiring any action by the CPU.

DMA configuration and data transfer are too complex to be covered in this application note. Several other application notes and code examples are available, including AN52705 – PSoC 3 and PSoC 5LP – Getting Started with DMA.

2.8 Port Interrupt Control Unit

PSoC 3 and PSoC 5LP have a port interrupt control unit (PICU) that manages I/O interrupts. Each GPIO pin can generate an interrupt on a rising edge, falling edge, or either edge condition. Level-sensitive interrupts are implemented by tying a cy_isr Component to the interrupt terminal of a Pins Component.

When a GPIO interrupt is triggered, the corresponding bit in that GPIO's status register is set to '1'. The bit will remain at '1' until the register is read or a chip reset occurs. The API provided by PSoC Creator manages GPIO interrupt configuration and reporting.

The individual GPIO interrupt signals within a port are ORed together, and a single PICU request is sent to the interrupt controller. The port interrupt requests are daisy-chained together to generate a single wakeup signal, which is sent to the PSoC power manager. The PICU remains active in all low-power modes, but the individual GPIO interrupts must still be managed after wakeup.

3 GPIO Pins in PSoC Creator

This section describes how to use PSoC Creator to configure and manipulate GPIO pins. PSoC Creator combines text and graphical editing interfaces so that designers can set their hardware configuration and write firmware at the same time.

3.1 PSoC Creator APIs

Cypress provides a set of APIs that you can use to dynamically control GPIOs through firmware. The APIs for the Pins Component enable access on both a component-wide and per-pin basis.

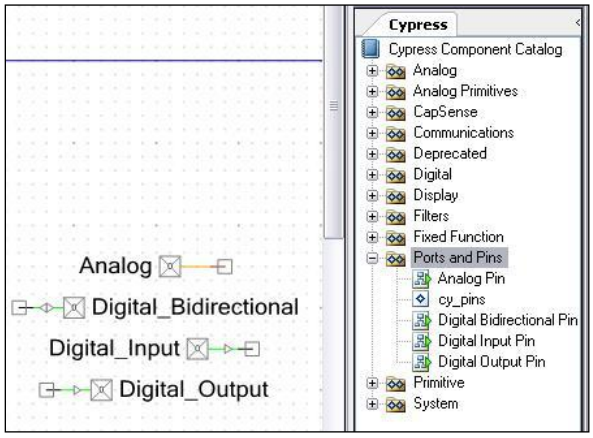
The cy_boot Component also provides functions to access chip resources. The functions in cy_boot are not part of the individual component libraries, but the libraries can use them. The per-pin APIs, which are provided as part of cy_boot in the *cypins.h* file, are documented in the “Pins” section of the PSoC Creator System Reference Guide. You can use these APIs to control the configuration registers for each physical pin.

For a summary and a simple code example for the APIs related to GPIOs, see GPIO API and Register Reference.

3.2 Pins Component Symbols and Macros

The cy_pins Component is the recommended way for internal PSoC resources to connect to a physical pin. It allows PSoC Creator to automatically place and route the signals within the PSoC device based on the chosen configuration of the pin. The standard Cypress component catalog contains four types of predefined GPIO configurations (macros) in the Ports and Pins class of symbols: analog, digital bidirectional, digital input, and digital output. Drag one of these component macros to the schematic to add a pin to the project, as Figure 6 shows.

Figure 6. Pins Component Symbol Types in PSoC Creator

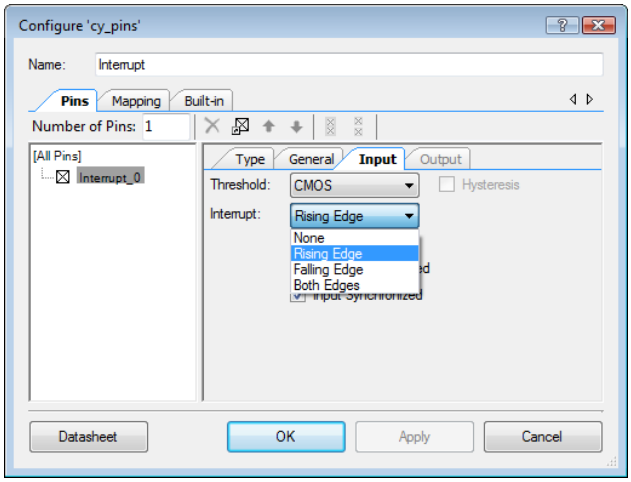


You are not confined to one type of pin configuration based on which macro symbol you choose. After you place the pin symbol on the schematic, you can configure its behavior using the component customizer options described in this document.

3.3 Pins Component Interrupts

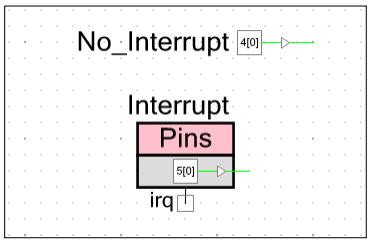
You can enable interrupts in Pins Components with the cy_pins configuration dialog in PSoC Creator, as Figure 7 shows. Double-click on the Pins Component to open it.

Figure 7. Interrupt Configuration in PSoC Creator



The Pins Component symbol changes when interrupts are enabled, as Figure 8 shows. The IRQ signal of the Pins Component will toggle when a pin interrupt is triggered. You do not need to connect the irq terminal to an isr Component to enable a pin interrupt.

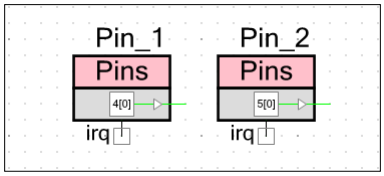
Figure 8. Pins Component Symbol Changes with Interrupts Enabled



If interrupts are enabled, you can use only one Pins Component with each physical GPIO port. The reason for this limitation is that all pin interrupts in a port are ORed together, so only one IRQ signal can be shown on the schematic.

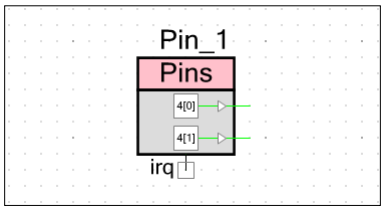
For example, consider two Pins Components with interrupts enabled, as Figure 9 shows. These Components cannot be mapped to pins in the same physical port because there are now two separate IRQ signals in the PSoC Creator schematic, but there is really only one physical PICU interrupt generated for the entire port.

Figure 9. Two Pins Components with Interrupts.



PSoC Creator will give an error if you try to assign these two Components to the same port. The accepted method is to assign multiple pins to the same Component, as Figure 10 shows. This ensures that there is only one IRQ signal in the schematic for that physical port. You can still assign each pin its own interrupt edge type. The only limitation is that the pins must be contiguous.

Figure 10. Pin Selection with Different Interrupt Edge Types.

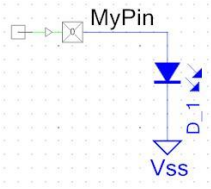


You can also use the PICUx_INTTYPEy registers to enable or change interrupts on any GPIO pin, regardless of Component settings. See the appendixes for more information on these registers.

3.4 External Terminals

The cy_pins configuration dialog offers an option to show an external terminal. This allows you to add Off-Chip Components to your schematic and show their connections to the Pin. Figure 11 shows an example of a Pins Component driving an off-chip LED**.**

Figure 11. Off-Chip Component Connection Example

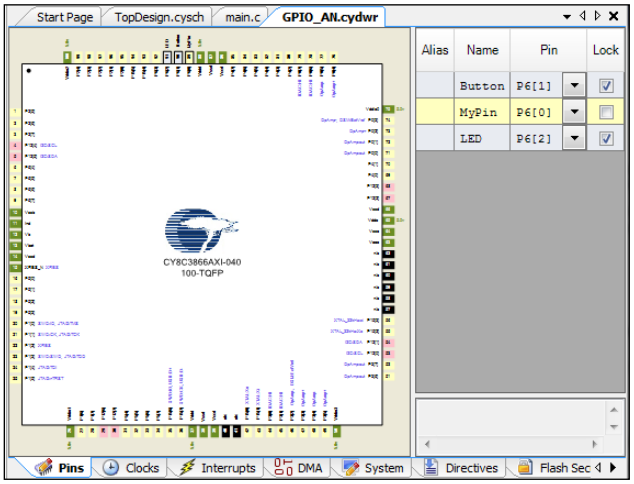


3.5 Manual Pin Assignments

A Pins Component is assigned to a physical pin through the **Pins** tab of the Design-Wide Resources interface (*cydwr*). PSoC Creator automatically assigns pins if none are chosen by the user, but this may lead to pin placement that is more difficult to route on a PCB. Also, some GPIO pins are directly connected to analog or digital resources.

Figure 12 shows three assigned pins. The pins highlighted in gray were manually assigned, and the pin highlighted in yellow was automatically assigned. Selecting the **Lock** option prevents the pin from being reassigned by PSoC Creator.

Figure 12. Pin Assignment in cydwr Window



PSoC Creator makes it simple to reassign pins as needed, but designers should consider pin selection before boards are designed. The “Analog Interconnect” diagrams in the TRM, AN58304, and AN58827 are valuable resources to help determine the optimal analog pin selection.

4 GPIO Examples, Tips, and Tricks

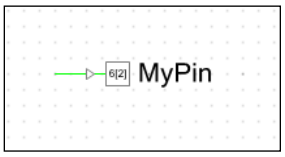
This section provides practical examples of how to use GPIO pins. The examples were generated for PSoC 3 devices, but the same techniques apply to PSoC 5LP. Both basic examples and more advanced techniques are included.

4.1 The GPIO “Hello World” Project

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

1. Place a Digital Output Pins Component, configured to Strong drive mode, in the project schematic, as Figure 13 shows.
2. Name the Component “MyPin” and assign it to P6[2].

Figure 13. Hello World Example Schematic



3. In *main.c*, use the Component API to toggle the output, as follows:

```
for(;;)
{
   /* Set MyPin output state to HIGH */
   MyPin_Write(1);

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

   /* Set MyPin output state to LOW */
   MyPin_Write(0);

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

4. Build the project and program the PSoC device.

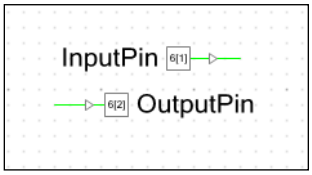
   The result is an output that toggles high/low every 500 ms.

4.2 Read an Input and Write to an Output

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

1. Place two pins in the project schematic—one digital input pin and one digital output pin—as Figure 14 shows.

Figure 14. Input and Output Example Schematic



2. Use the Component APIs to set the state of OutputPin based on InputPin as follows:

```
for(;;)
{
   /* Set OutputPin state to the
      inverse of the InputPin state */
```

```
    OutputPin_Write( ~InputPin_Read() );
}
```

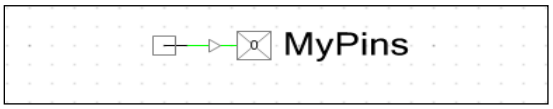
The result is that OutputPin is always at the opposite state of InputPin.

4.3 Add Multiple GPIO Pins as a Logical Port

In PSoC Creator, you can organize a group of as many as 64 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 be from separate physical ports.

1. Place a single pin symbol, as Figure 15 shows.

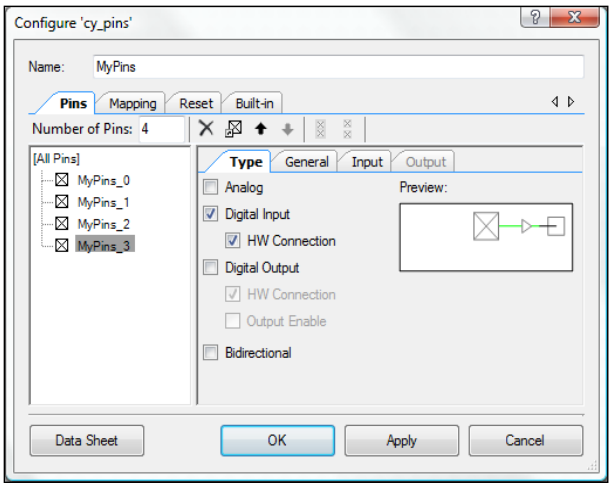
Figure 15. Single Pin Symbol Placed in a Schematic



2. Double-click on the pin symbol to open the pin customizer window.
3. Type the number of pins in the **Number of Pins** field in the configuration window.

   The pins will appear in the list below the field. Select an individual pin in the list to allow it to be customized independently of the others. Select **[All Pins]** to affect every pin in the port.
4. For this example, set three of the pins as digital output pins. Set the last as a digital input, as shown in Figure 16.

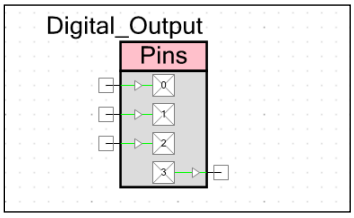
Figure 16. One of Four Pins Configured as a Digital Input



5. Click **OK** to apply the changes.

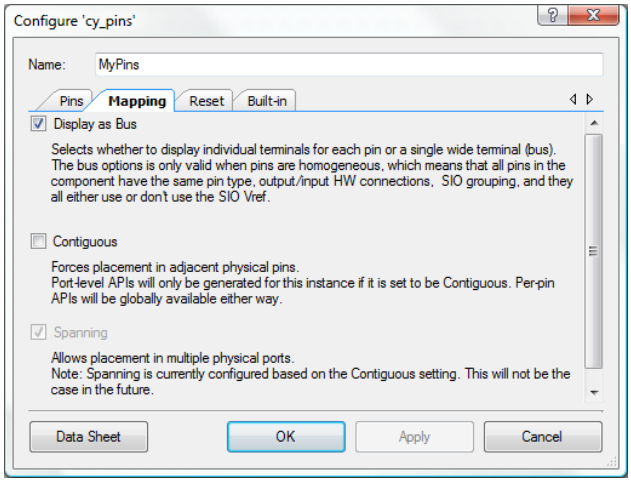
   After you define the number of pins and their types, the schematic symbol will resemble Figure 17.

Figure 17. Pins Component in Port Configuration



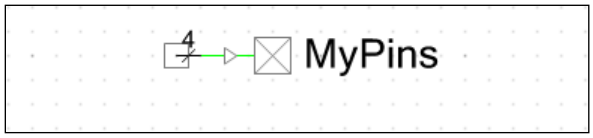
6. (Optional) Select **Display as Bus** in the **Mapping** tab of the pin configuration window to display the port as a bus symbol, as Figure 18 and Figure 19 show.

Figure 18. Display as Bus Option



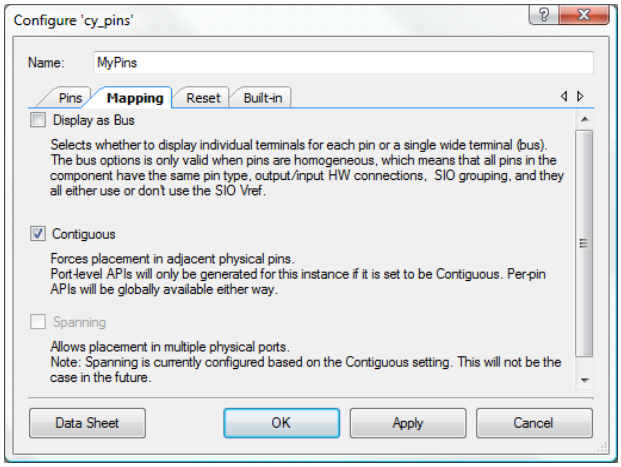
This feature does not affect the behavior of the port. **Note** All pins must be of the same type for them to be displayed as a bus.

Figure 19. Four Pins Displayed as a Port Bus Symbol



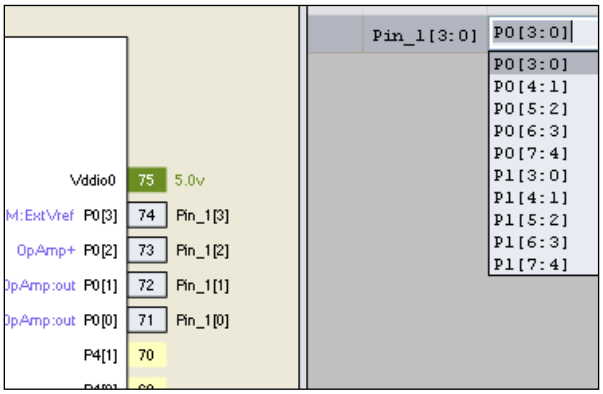
7. (Optional) Select **Contiguous** in the **Mapping** tab to force the pins to be physically adjacent, as Figure 20 shows.

Figure 20. Contiguous Pin Placement Option



When you select **Contiguous**, PSoC Creator will modify the list of available pinout options to match the port's configuration, as Figure 21 shows.

Figure 21. Pin Placement of Contiguous Port Pins



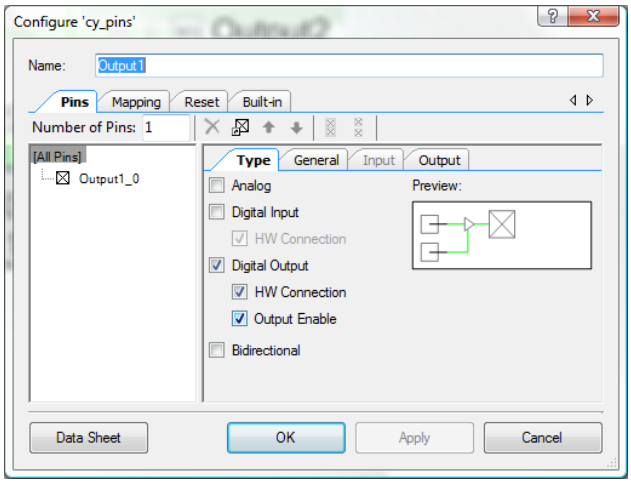
These features are described in more detail in the pin configuration window and the Pins Component datasheet.

4.4 Configure GPIO Output Enable Logic

This example demonstrates how to configure and use the output enable logic of a GPIO pin.

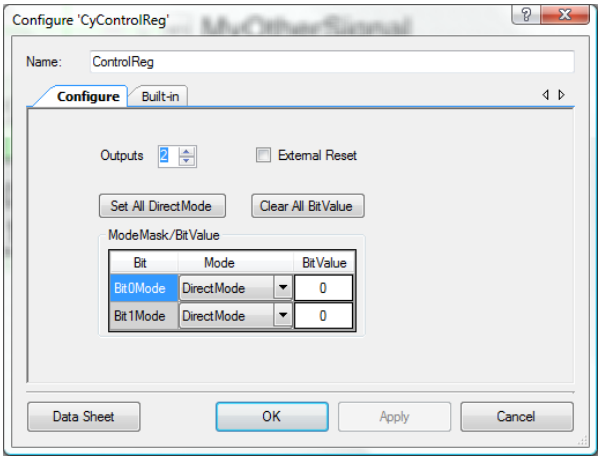
1. Place two digital output pins in the project schematic.
2. Open the configuration dialog for each pin and check the **Output Enable** box, as Figure 22 shows.

Figure 22. Output Enable Selection



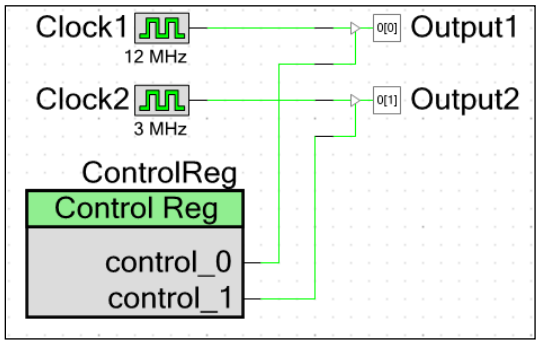
3. Place a control register in the schematic.
4. Configure the control register for two outputs, as Figure 23 shows.

Figure 23. Control Register Configured with Two Outputs



5. Add two Clock Components, configured in any way.
6. Connect the clocks to the pins, as Figure 24 shows.

Figure 24. Control Register Driving Pins' Output Enable



7. Add the following code to the *main.c* file:

```
for(;;)
{
    for( i=0; i<=3; i++ )
    {
        ControlReg_Write(i);
        CyDelay(500);
    }
}
```

8. Compile and program the PSoC 3 or PSoC 5LP device.

   The result is the output of the two pins gated by the state of ControlReg.

   This same method can be used for the creation of a bi-directional data bus. Place as many pins as required for the data bus and check the **Output Enable** box in each Pin Component. Connect all the output enable signals to a single Control Register output. When Output Enable is enabled the GPIO pins are strongly driven out. When the Output Enable is disabled the GPIO pin input logic state can be read.

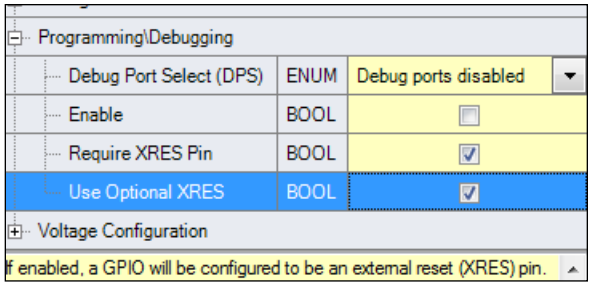
4.5 Enable the Configurable XRES Feature

This example demonstrates how to enable the configurable XRES feature. You can configure Pin P1[2] as an optional XRES pin to support an external reset for small packages. The feature is also available in the larger packages.

1. Open the **System** tab in the Design-Wide Resources file, as shown in Figure 25.

2. Select the **Use Optional XRES** option to enable the optional XRES logic. If this box is selected, P1[2] stops functioning as a GPIO pin and is configured as an active LOW input with an internal pull-up.

Figure 25. Optional XRES Pin Enable



3. Program the PSoC device to write the setting to the nonvolatile array. It will take effect after the next power on.
4. Deselect the **Use Optional XRES** option and reprogram the PSoC device to restore normal GPIO functionality.

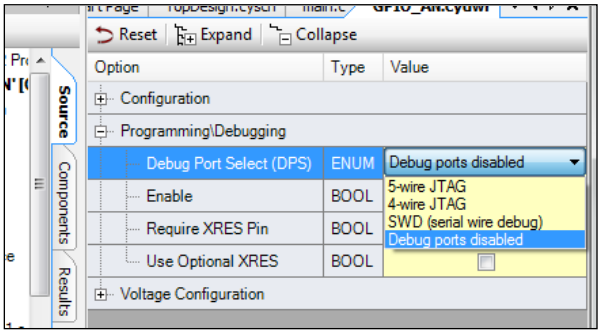
Note that all PSoC 3 and PSoC 5LP devices come from the factory with the optional XRES feature disabled. Using the configurable XRES pin does not change the functionality of a dedicated XRES pin.

4.6 Disable Debug Logic on GPIO Pins

This example demonstrates how to disable the debug logic associated with the port 1 pins. If the debug port feature is enabled, the PSoC device will enter debug mode if it detects activity on these pins at boot time.

1. Open the Design-Wide Resources file and click the **System** tab.
2. Select **Debug ports disabled** from the drop-down menu, as Figure 26 shows.

Figure 26. Debug Ports Disabled



3. Compile and program the PSoC 3 or PSoC 5LP device.

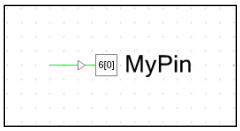
Note that the debug port must be manually enabled again if debugging is needed. Disabling the debug interface does not affect the ability to program the device.

4.7 Toggle GPIOs Faster with Data Registers

This example demonstrates how to use port data registers and masks to quickly toggle pins. While the Component API is the easiest way to control GPIO pins, the number of processor cycles needed to update the pin can affect how fast a toggle can occur. The register definitions and masks in the *<pin_name>.h* file that is created for each Component can be used to more quickly update pins.

1. Place a digital output pin in the schematic and name it “MyPin” for convenience.
2. Configure the Component with no hardware connection and assign it to a physical pin (this example uses P6[0]), as Figure 27 shows.

Figure 27. Pin Placed in Schematic

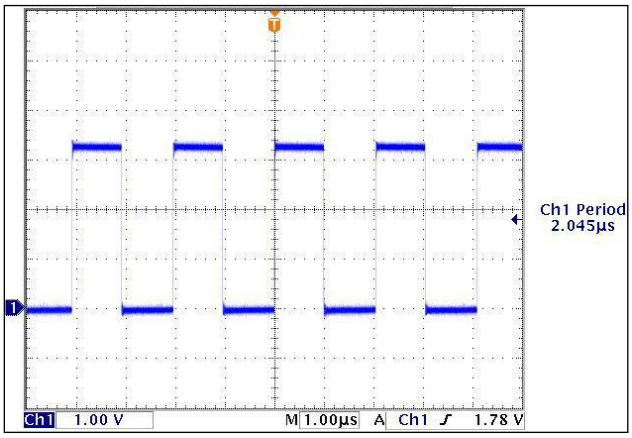


3. Add the following code to the *main.c* file:

```
for(;;)
{
   // These are API functions
   MyPin_Write(1); //set MyPin output
   MyPin_Write(0); //clear MyPin output
}
```

4. Observe the output of P6[0] using the API, as Figure 28 shows.

Figure 28. Pin Toggle Using API Switch Method

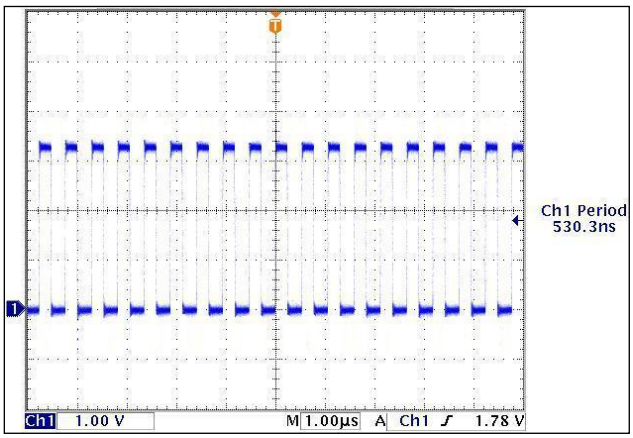


5. Replace the previous code in *main.c* with this code:

```
for(;;)
{
   MyPin_DR |= MyPin_MASK; //Set MyPin
   MyPin_DR &= ~MyPin_MASK; //Clear
}
```

6. Observe the output of P6[0] using the fast switching method, as Figure 29 shows.

Figure 29. Pin Toggle Using Fast Switching Method



A pin can toggle almost four times faster using the fast switching method rather than the API functions. This code also has the advantage of being portable. If the pin assignment is changed during development, you do not have to write to a specific physical pin's registers.

4.8 Use 8051 Special Function Registers

The 8051 in PSoC 3 has a set of special function registers (SFRs) that allow faster access to a limited set of PSoC registers. You can use two of those registers to quickly toggle GPIO pins.

1. Place a Digital Output Pins Component in the project schematic and assign it to a physical pin, just as you did in the previous example. This example also uses P6[0].
2. Add the following code to the *main.c* file:

```
/* Enable SFR access for P6[0]. */
/* Only done once in the beginning. */
SFRPRT6SEL |= 0x01;
/* Toggle GPIO pin. */
for(;;)
{
     /* Switch on P6[0] */
     SFRPRT6DR |= 0x01;
     /* Switch off P6[0] */
     SFRPRT6DR &= ~0x01;
}
```

3. Alternatively, use this method:

```
for(;;)
{
     /* Toggle P6[0] */
     SFRPRT6DR ^= 0x01;
}
```

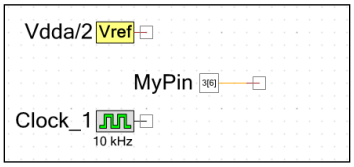
Either method will result in very fast pin toggles. For more information on the SFRs, see the PSoC 3 Architecture TRM.

4.9 Use Both Analog and Digital on a GPIO

This example demonstrates how to configure and use a pin for both analog and digital functions. Assume that a GPIO pin needs to output a 10-kHz clock signal for a short time, switch to a reference voltage for a short time, and then switch back to the 10-kHz signal.

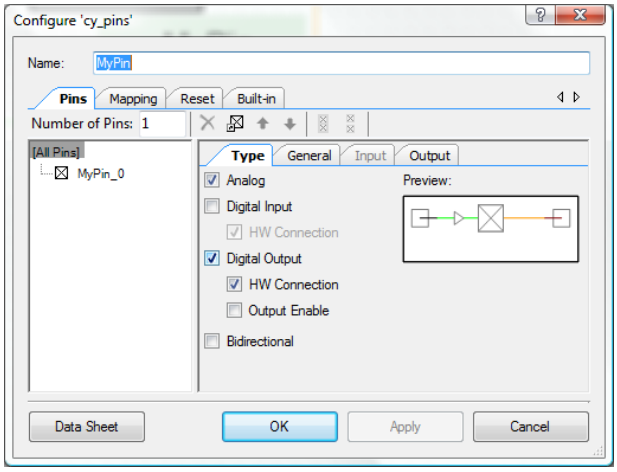
1. Place an analog pin, a V_{REF}, and a clock in the schematic.
2. Assign the Pins Component to a physical pin (this example uses P3[6]), as Figure 30 shows.

Figure 30. Basic Components Placed in the Schematic



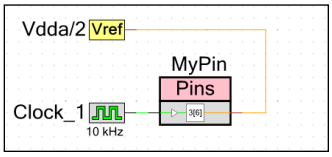
3. Configure the pin with both analog and digital output settings, as Figure 31 shows.

Figure 31. MyPin Configured as Both Analog and Digital



4. Connect the clock to the digital terminal and the V_{REF} to the analog terminal, as Figure 32 shows.

Figure 32. PSoC Creator Schematic of Analog and Digital Switching Scheme



5. Compile the project to create the API necessary to determine the analog routing that PSoC Creator uses.
6. Open the *cyfitter_cfg.c* file and look for either `CYREG_PRT3_AG` (the analog global enable) or `CYREG_PRT3_AMUX` (the analog mux bus enable). In this case, the routing tool has chosen to use the AG bus for Port 0, as follows.

```
CY_SET_REG8(CYREG_PRT3_AG, 0x40);
```

Note that the analog routing may change whenever the project is rebuilt. If any changes are made to the project, you must check the routing.

7. Add the following code to the *main.c* file:

```
for(;;)
{
    /* Set pin to Analog */
    // Set P3[6] to Analog HI-Z
    CyPins_SetPinDriveMode(CYREG_PRT3_PC6, PIN_DM_ALG_HIZ);

    // Make AG connection for P3[6]
    CY_SET_REG8(CYREG_PRT3_AG, CY_GET_REG8(CYREG_PRT3_AG) | 0x40);

    // Wait for 100 ms while driving signal
    CyDelay(100);

    /* Set pin to digital */
    // Break AG connection for P3[6]
    CY_SET_REG8(CYREG_PRT3_AG, CY_GET_REG8(CYREG_PRT3_AG) & 0xBF);

    //Set P3[6] to Strong Drive mode
    CyPins_SetPinDriveMode(CYREG_PRT3_PC6, PIN_DM_STRONG);

    // Wait for 100 ms while driving signal
    CyDelay(100);
}
```

8. Compile and program the PSoC 3 or PSoC 5LP device.

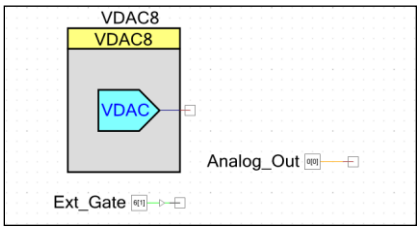
The result is an output that alternates every 100 ms between the clock signal and the reference voltage.

4.10 Control Analog Switching with Hardware

This example shows how an external signal is used to gate the output of an analog pin without CPU intervention.

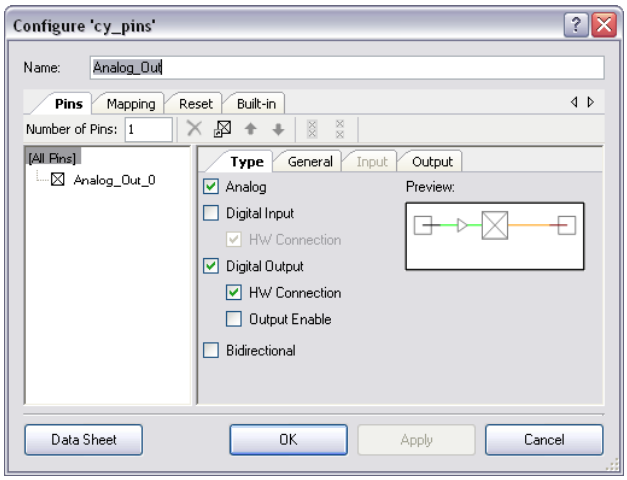
1. Place a digital input pin (Ext_Gate in this example), an analog pin (Analog_Out), and an analog source (VDAC8) in the project schematic, as Figure 33 shows.

Figure 33. Components for Hardware-Controlled Gate



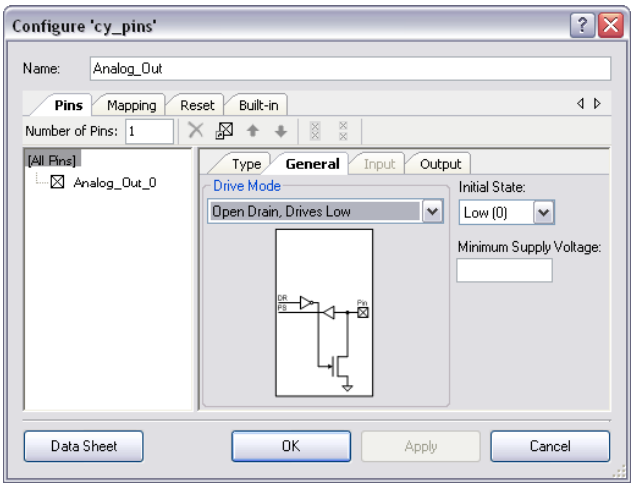
2. Configure the Analog_Out pin with both analog and digital properties, as Figure 34 shows.

Figure 34. Analog_Out Pin Configuration



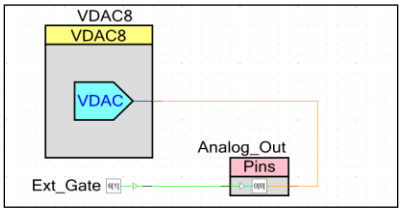
3. Configure the drive mode of the Analog_Out pin as “Open Drain Drives Low,” as Figure 35 shows.

Figure 35. Analog_Out Pin Drive Mode



4. Connect the Components, as Figure 36 shows.

Figure 36. Analog Pin with Hardware Gate



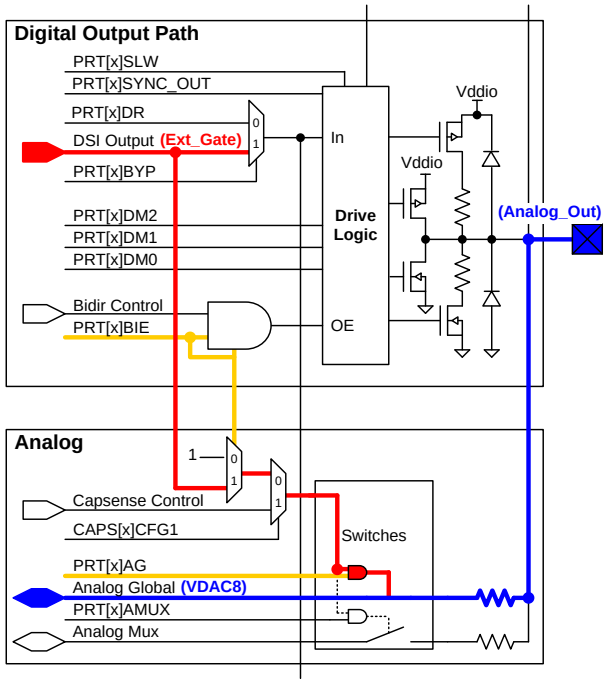
5. Add the following line of code to the *main.c* file. In this example, it sets the bidirectional bit for pin 0 in the Port 0 configuration registers:

```
// Set P0[0] to bidirectional mode
CY_SET_REG8(CYDEV_IO_PRT_PRT0_BIE, 0x01);
```

6. Assign the Analog_Out pin to P0[0] to match the previous code.
7. Compile and program the PSoC 3 or PSoC 5LP device.

Figure 37, which is taken from the detailed GPIO block diagram in the PSoC 3 and PSoC 5LP datasheets, shows how the GPIO control logic is used to implement this technique.

Figure 37. Highlighted GPIO Block Diagram from PSoC 3 and PSoC 5LP Datasheets



The Ext_Gate signal is routed through the DSI to the digital portion of the Analog_Out pin. The signal from the DSI (red) is routed to the analog switches because the port bidirectional bit and the analog global select bit are set (yellow). The VDAC output (blue) is switched on or off depending on the logic state of the Ext_Gate signal.

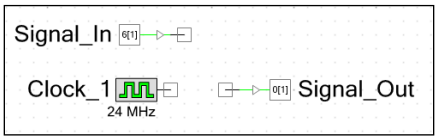
For more details on the analog switching available in PSoC 3 and PSoC 5LP devices, see the “Analog Routing” section of the TRM. Another good resource is the application note AN58827 – PSoC 3 and PSoC 5LP Internal Analog Routing Considerations.

4.11 Use the DSI as a Clock Source

This example demonstrates how to use a digital signal routed through the DSI as a clock source. As many as eight digital and four analog clocks can be created from an arbitrary DSI signal. Also, the PSoC device can use an arbitrary digital signal as an input source for the PLL.

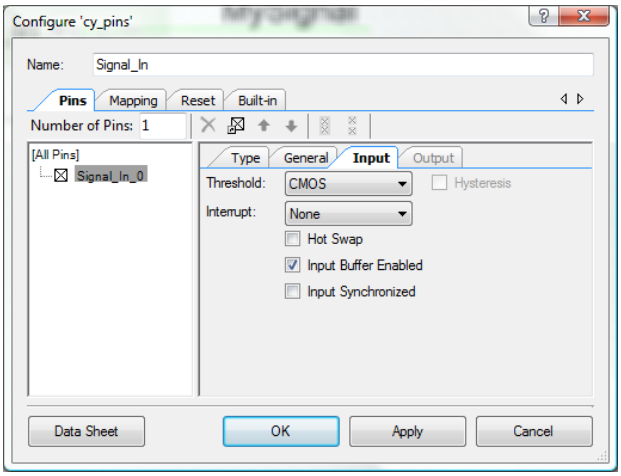
1. Place a digital input pin, a digital output pin, and a clock in the schematic, as Figure 38 shows.

Figure 38. Basic Components for DSI Clock Example



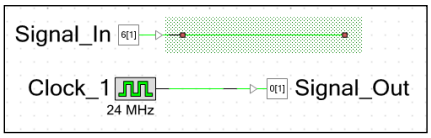
2. Open the configuration dialog for Signal_In and deselect the **Input Synchronized** option in the pin configuration window, as Figure 39 shows. This is necessary to prevent the signal from trying to sync to itself.

Figure 39. Input Synchronized Setting Disabled



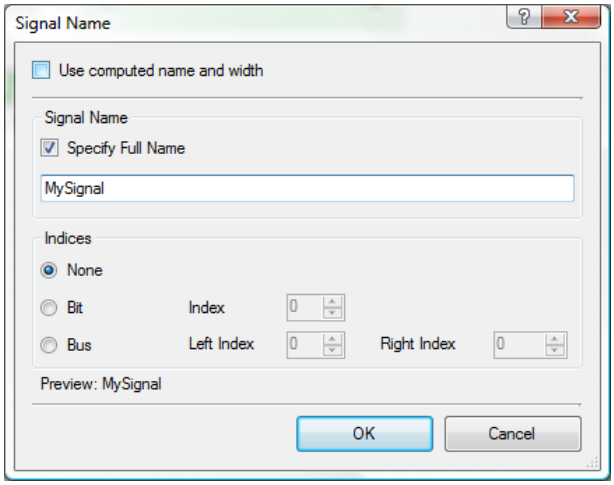
3. Connect the clock to Signal_Out with the wire tool.
4. Use the wire tool to create a signal for the DSI source that is connected only to Signal_In. Start away from the terminal of Signal_In to create the wire, as shown in Figure 40.

Figure 40. DSI Clock Source Signal



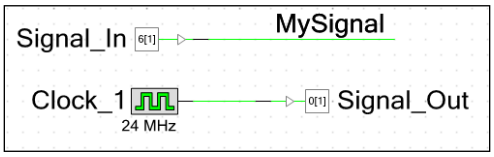
5. Right-click on the wire and select **Edit Name And Width** from the pop-up menu that appears.
6. Deselect the **Use computed name and width** option and type a unique name (“MySignal” in this example) in the **Signal Name** field, as Figure 41 shows.

Figure 41. Signal Name Configuration Window



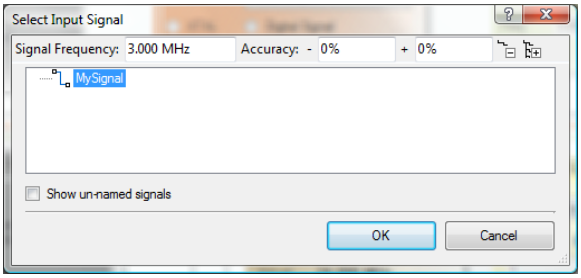
After these configurations have been set, the schematic will resemble Figure 42.

Figure 42. Modified Components for DSI Clock Example



7. Open the project's Design-Wide Resources file (*<Project Name>.cydwr*) and select the **Clocks** tab.
8. Double-click on any of the system clocks to open the **Configure Systems Clocks** window.
9. Select the **Digital Signal** option and click the "**…**" button to open the **Select Input Signal** window.
10. Select "MySignal" and enter the signal frequency (3 MHz in this example) and accuracy, as Figure 43 shows.

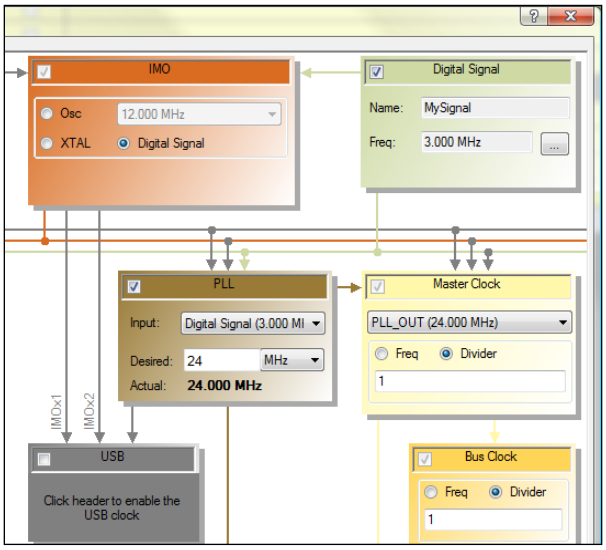
Figure 43. Digital Signal Configuration Window



Note that PSoC Creator uses this information to perform calculations needed to configure the system clocks. Make sure that the signal information is accurate.

11. Set the input source for the internal main oscillator (**IMO**) and **PLL** to "Digital Signal," as Figure 44 shows.

Figure 44. DSI Signal as the Source for the IMO and PLL



The PLL uses the 3-MHz input to generate a 24-MHz output, which is routed to the Master and Bus clocks to generate the Signal_Out clock.

For more details on the DSI system clocking, see the datasheet, TRM, and application note AN60631 – PSoC 3 and PSoC 5LP Clocking Resources.

4.12 Change PICU Settings with Firmware

Dynamic configuration of the PICU is done through a write to bits [1:0] of the `PICUx_INTTYPEy` register, where “x” corresponds to the port number and “y” corresponds to the pin number (see Table 2). You can change the configuration at any time to enable or disable pin interrupts.

Table 2. PICU Interrupt Types and Bit Settings

Bits 1:0	Name	Description
00	Disable	Interrupts disabled
01	Rising Edge	Trigger on rising edge
10	Falling Edge	Trigger on falling edge
11	Change Mode	Trigger on any edge

In this example, pin P6[0] is configured as a rising-edge interrupt, and P6[1] is configured as a falling-edge interrupt.

1. Place two digital input pins in the project schematic.
2. Assign the pins to P6[0] and P6[1].
3. Configure P6[0] as a resistive pull-down pin, or add an external pull-down.
4. Configure P6[1] as a resistive pull-up pin, or add an external pull-up.
5. Add the following code to the *main.c* file:

```
//Set P6[0] to PICU rising-edge trigger
CY_SET_REG8(CYREG_PICU6_INTTYPE0, 0x01);

//Set P6[1] to PICU falling-edge trigger
CY_SET_REG8(CYREG_PICU6_INTTYPE1, 0x02);

//Sleep and wait for PICU interrupt
//Sleep again if not P6[1] PICU wakeup
do
{
```

```
        //Save clocks and enter sleep
        CyPmSaveClocks();
        CyPmSleep(PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_PICU);
        CyPmRestoreClocks();

        //Stay awake for two seconds
        CyDelay(2000);
    }
    while (!(CY_GET_REG8(CYREG_PICU6_INTSTAT) & 0x02));

    //Disable P6[1] PICU trigger
    CY_SET_REG8(CYREG_PICU6_INTTYPE1, 0x00);
```

6. Compile and program the PSoC 3 or PSoC 5LP device.

The PSoC device will wake from sleep on any PICU interrupt, but it will return to sleep again unless P6[1] was the trigger. You are not required to disable the interrupts after wakeup. They can be used during normal operation like any other interrupt source.

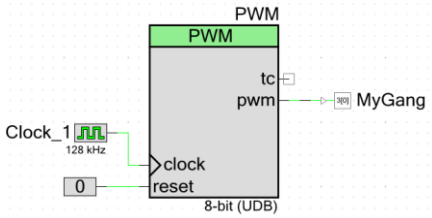
The PSoC 3 or PSoC 5LP TRM contains further information about the PICU, including block diagrams and functional descriptions. Another good resource is the application note AN54460 – PSoC 3 and PSoC 5LP Interrupts.

4.13 Gang Pins for More Drive/Sink Current

To increase the total source/sink capabilities of the circuit, GPIO pins can be ganged (shorted together). The limitations of V_{DDIO} quadrants still apply. This example demonstrates driving a PWM signal with four GPIO pins.

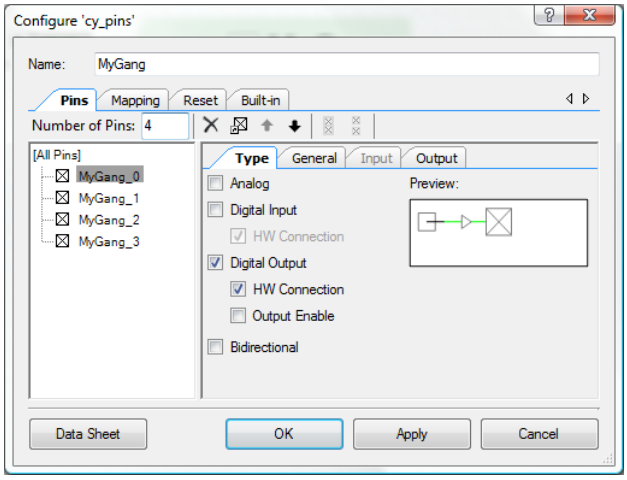
1. Place and configure a PWM and a Clock Component.
2. Place a single Digital Output Pins Component and connect it to the PWM output terminal, as Figure 45 shows.

Figure 45. Single Pin Placed in Schematic



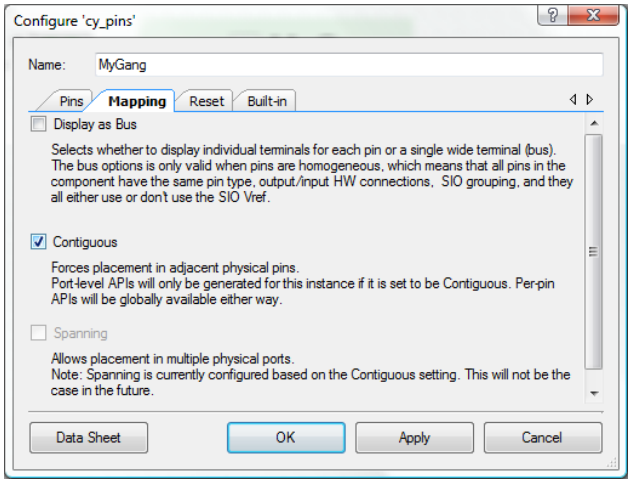
3. Open the Pins configuration dialog and set the number of pins accordingly, as Figure 46 shows. This example uses four GPIO pins.

Figure 46. Configure Multiple Pins in the Component



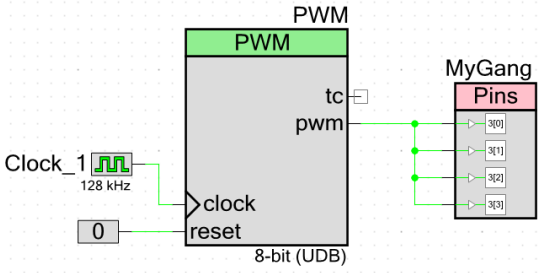
4. Optionally, set the pin mapping to **Contiguous** for easier PCB routing, as Figure 47 shows.

Figure 47. Enable Contiguous Mapping



5. Assign the Pins Component to physical pins.
6. Connect the signal source (PWM in this example) to each of the pin terminals in the Component, as shown in Figure 48.

Figure 48. Ganged Pins Symbol



7. Compile and program the PSoC 3 or PSoC 5LP device.

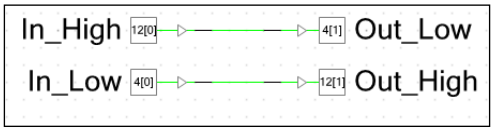
The output of the PWM will be driven on all four GPIOs. The pins can be shorted externally on the PCB and connected to the external circuit as needed.

4.14 Level-Shift Signals

The GPIO pins can be used for level-shifting of signals by powering the V_{DDIO} pins at different voltages. The only limitation is that no V_{DDIO} quadrant may be at a higher voltage than V_{DDA}. This example demonstrates how to create a simple 5-V/1.8-V level shifting configuration in the PSoC 3 or PSoC 5LP device.

1. Place two High-Z digital input pins and two Strong Drive digital output pins in the project schematic.
2. Connect one of the inputs to one of the outputs. Connect the remaining input to the remaining output.
3. For convenience, give the pin symbols meaningful names similar to the ones shown in Figure 49.

Figure 49. Level-Shifting with GPIO Pins

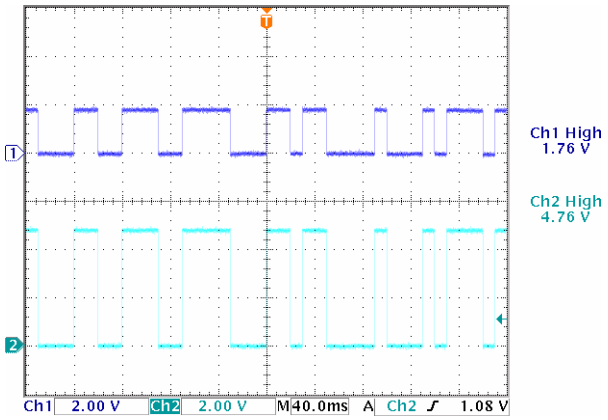


4. Assign the 5-V signals to pins in one V_{DDIO} quadrant, and the 1.8-V signals to pins in another quadrant. For this example, $V_{DDIO}3$ (5.0 V – P12[1:0]) and $V_{DDIO}0$ (1.8 V – P4[1:0]) were chosen. See the device datasheet for V_{DDIO} distributions.
5. On the development board, connect V_{DDD}, V_{DDA}, $V_{DDIO}1$, $V_{DDIO}2$, and $V_{DDIO}3$ to a 5-V supply.

6. Connect $V_{DDIO}0$ to a 1.8-V supply.
7. Compile and program the PSoC 3 or PSoC 5LP device.

Any 5-V signal applied to the high-side input will appear at 1.8 V on the low-side output. Likewise, any 1.8-V signal applied to the low-side input will appear at 5 V on the high-side output, as Figure 50 shows.

Figure 50. Level-Shifted Signals



In addition to shifting the voltage of the signal, the PSoC device can read and manipulate the data as it passes through. Remember that all GPIO pins in a V_{DDIO} quadrant will be at the same voltage.

5 Related Application Notes

- AN54181 – Getting Started with PSoC 3
- AN77759 – Getting Started with PSoC 5LP
- AN60631 – PSoC 3 and PSoC 5LP Clocking Resources
- AN58304 – PSoC 3 and PSoC 5LP – Pin Selection for Analog Designs
- AN58827 – PSoC 3 and PSoC 5LP Internal Analog Routing Considerations
- AN54460 – PSoC 3 and PSoC 5LP Interrupts
- AN60580 – SIO Tips and Tricks in PSoC 3/PSoC 5LP
- AN77900 – PSoC 3 and PSoC 5LP Low-power Modes and Power Reduction Techniques
- AN52705 – PSoC 3 and PSoC 5LP – Getting Started with DMA

About the Author

Name:	Greg Reynolds
Background:	Greg Reynolds has been with Cypress in several roles for more than a decade.

Appendix A. GPIO API and Register Reference

A.1 Component API

The Pins Component API is used to access all physical pins associated with a Component. The instance name of the Component, either assigned by the user or generated by PSoC Creator, is used as the prefix for the function name. The Component APIs are documented in detail in the Pins Component datasheet. Table 3 lists the Component functions.

Table 3. Component API Reference

API	Description	Example (using a Component name of MyPin)
<Pin_Name>_Read	Returns the current value for all pins in the Component.	`myVar = MyPin_Read();`
<Pin_Name>_Write	Writes the value to the Component pins.	`MyPin_Write(1);`
<Pin_Name>_ReadDataReg	Returns the current value for all pins in the Component.	`myVar = MyPin_ReadDataReg();`
<Pin_Name>_SetDriveMode	Sets the drive mode for each of the Component's pins.	`MyPin_SetDriveMode(MyPin_DM_ALG_HIZ);`
<Pin_Name>_ClearInterrupt	Clears any active interrupts on the port into which the Component is mapped. Returns the value of interrupt status register.	`myVar = MyPin_ClearInterrupt();`

A.2 Per-Pin API

You can access individual physical pins by using the global per-pin API macros. The physical pins do not need to be associated with a Pins Component because the macros directly access the pin configuration registers. Using per-pin APIs can result in undefined behavior in any physical pin associated with a Pins Component due to conflicts with the Component configuration. The per-pin APIs are documented in detail in the PSoC Creator System Reference Guide. Table 4 lists the per-pin functions.

Table 4. Per-Pin API Reference

API	Description	Example (using P1[2])
CyPins_ReadPin	Reads the current value on the pin.	`myVar = CyPins_ReadPin(CYREG_PRT1_PC2);`
CyPins_SetPin	Sets the output value for the pin to logic HIGH.	`CyPins_SetPin(CYREG_PRT1_PC2);`
CyPins_ClearPin	Sets the output value for the pin to a logic LOW.	`CyPins_ClearPin(CYREG_PRT1_PC2);`
CyPins_SetPinDriveMode	Sets the drive mode for the pin.	`CyPins_SetPinDriveMode(CYREG_PRT1_PC2, PIN_DM_ALG_HIZ);`
CyPins_ReadPinDriveMode	Reads the drive mode for the pin.	`myVar = CyPins_ReadPinDriveMode(CYREG_PRT1_PC2);`
CyPins_FastSlew()	Sets the slew rate for the pin to fast edge rate.	`CyPins_FastSlew(CYREG_PRT1_PC2);`
CyPins_SlowSlew()	Sets the slew rate for the pin to slow edge rate.	`CyPins_SlowSlew(CYREG_PRT1_PC2);`

A.3 GPIO Registers

The following registers are used to configure the GPIOs. They are accessible by firmware during normal operation. Further details and register maps are in the PSoC 3 and PSoC 5LP Registers TRM documents that are available for free from the Cypress website. Table 5 lists the GPIO registers.

Table 5. GPIO Registers

Register	Name	Description
PRTx_PCy[7:0]	Port Pin Configuration	This register accesses several configuration or status bits of a single I/O port pin at once.
PRTx_DR[7:0]	Port Data Output	This register is used to set the output data state for the corresponding GPIO port.
PRTx_PS[7:0]	Port Pin State	This register holds the logical pin state for the corresponding GPIO port. If the drive mode for the pin is set to High-Z Analog, the state will always read 0.
PRTx_DM2[7:0] PRTx_DM1[7:0] PRTx_DM0[7:0]	Port Drive Mode	These registers' combined value determines the unique drive mode of each bit in a GPIO port.
PRTx_SLW[7:0]	Port Slew Rate Control	This register is used to set a fast or slow edge rate for any strong drive mode GPIO pin.
PRTx_BYP[7:0]	Port Bypass Enable	This register selects whether the output data for the corresponding GPIO is sourced from the DSI or port logic data register.
PRTx_BIE[7:0]	Port Bidirection Enable	This register is used to enable dynamic bidirectional control through the DSI.
PRTx_INP_DIS[7:0]	Input Buffer Disable Override	This register is used to force the input buffers off.
PRTx_CTL[0]	Port Wide Control Signals	This register is used to select the internal buffer trip point.
PRTx_PRT[7:5,3:1]	Port Wide Configuration	This register accesses several available configuration registers on a port-wide basis with a single bit write.
PRTx_BIT_MASK[7:0]	Bitmask for Aliased Register Access	This register allows or blocks access to the data registers from the aliased register address space.
PRTx_AMUX[7:0]	Port Analog Global Mux Bus Enable	This register controls the analog global mux switch for the corresponding GPIO port.
PRTx_AG[7:0]	Port Analog Global Enable	This register controls the analog global switch for the corresponding GPIO port.
PICUx_INTTYPE[1:0]	Port Interrupt Control Type	This register configures the type of interrupt for the corresponding GPIO pin.
PICUx_INTSTAT[7:0]	Port Interrupt Control Status	This register shows posted interrupts for the corresponding GPIO port.
PICUx_SNAP[7:0]	Port Interrupt Control Snapshot	This register shows the state of input pins at the last read of the PICUx_INTSTAT register.
PICUx_DISABLE_COR[0]	Disable Status Register Clear on Read Feature	This register disables the "clear on read" feature of the PICUx_INTSTAT register.

A.4 Nonvolatile Latches

PSoC 3 and PSoC 5LP have an array of nonvolatile latches (NVLs) that are used to configure the device behavior in reset. The following latches are used to configure the GPIOs. They are not accessible from firmware during normal operation. Further details and register maps for the NVL array are included in the “Nonvolatile Latches” section of the PSoC 3 and PSoC 5LP datasheets and TRM. Table 6 lists the NVL latches.

Table 6. NVL Latches Relating to GPIOs

NVL Latch	Name	Description
PRTxRDM[1:0]	Port Reset Drive Mode	Controls reset drive mode of the corresponding I/O port.
XRESMEN[0]	Optional XRES Enable	Controls whether pin P1[2] is used as a GPIO or as an external reset.
DPS[1:0]	Debug Port Select	Controls the use of various Port1 pins as a debug port.

Appendix B. PSoC Creator Settings and Registers

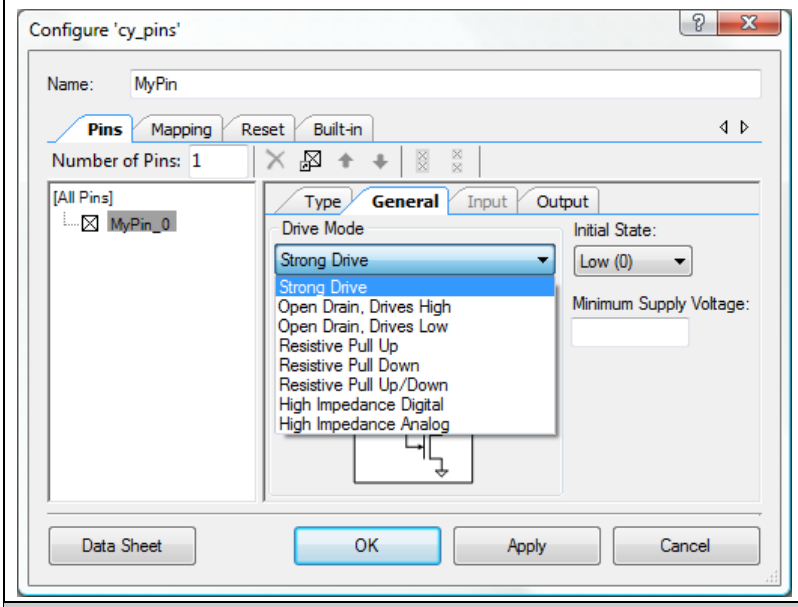
The GPIO settings established in PSoC Creator are part of the cy_boot startup code and take effect during initial device configuration. The tables in this appendix show the relationship between the settings in the Pins Component configuration window and the GPIO registers. In addition, simple code examples demonstrate how to perform the same function in the firmware during normal operation, if applicable. The configuration window may look slightly different, depending on which version of PSoC Creator you are using.

Table 7. Drive Mode Parameter

Drive Mode

This parameter configures the pin to provide one of the eight available pin drive modes. Pin type determines default settings.

PSoC Creator Configuration Window



Component APIs

<PinName>_SetDriveMode()

Per-Pin APIs

CyPins_SetPinDriveMode()
CyPins_ReadPinDriveMode()

Associated Registers

PRTx_PCy[3:1]
PRTx_DMy[7:0]
PRTx_BIE[7:0]

Code Examples

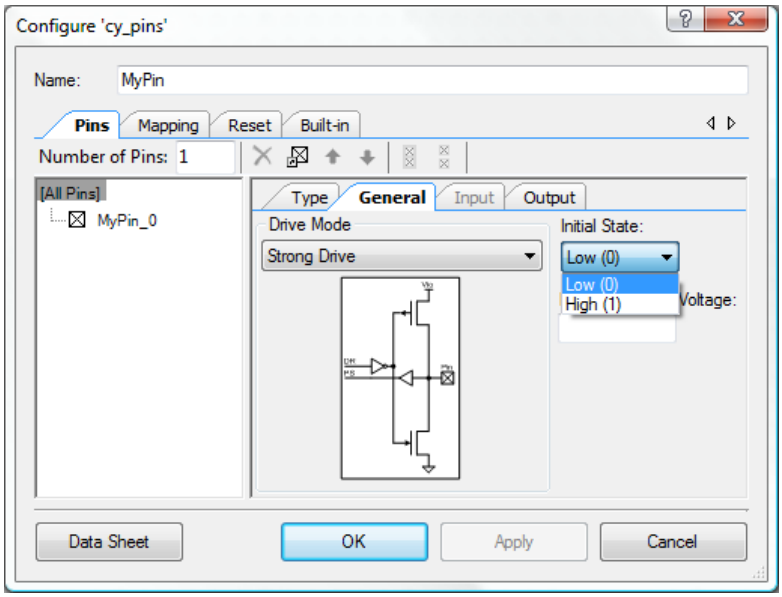
```
/* Set pin to Resistive Pull-up Using Component API */
MyPin_SetDriveMode(MyPin_DM_RES_UP);

/* Set pin to Resistive Pull-up Using Per-Pin API */
CyPins_SetPinDriveMode(CYREG_PRT1_PC2,PIN_DM_RES_UP);

/* Read Drive Mode Using Per-Pin API */
myVar = CyPins_ReadPinDriveMode(CYREG_PRT1_PC2);

/* Set pin to Resistive Up/Down Using Register Write */
CY_SET_REG8(CYREG_PRT1_PC2, CY_GET_REG8(CYREG_PRT1_PC2) | 0x07);
```

Table 8. Initial State Parameter

Initial State	
This parameter specifies the value written to the pin's data register after power-on reset (POR).	
PSoC Creator Configuration Window	**Component APIs**
	<PinName>_Write() **Per-Pin APIs** CyPins_SetPin() CyPins_ClearPin() **Associated Registers** PRTx_DR[7:0]
Code Examples	

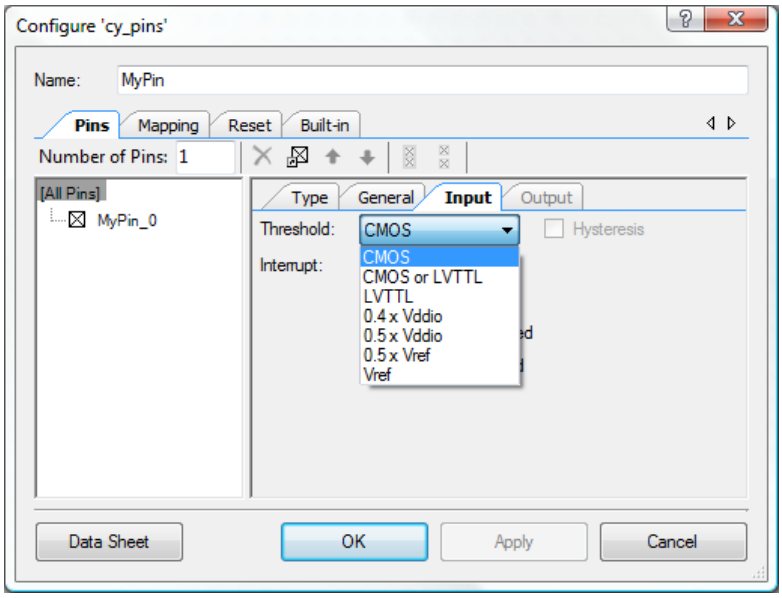
```
/* Set pin to logic state HIGH output */
       MyPin_Write(1);

/* Set pin P1[2] to logic state HIGH output */
       CyPins_SetPin(CYREG_PRT1_PC2);

/* Set pin P1[2] to logic state LOW output */
       CyPins_ClearPin(CYREG_PRT1_PC2);

       /* Set pin P1[2] output to logic state LOW using a register write */
       CY_SET_REG8(CYREG_PRT1_DR, CY_GET_REG8(CYREG_PRT1_DR) | 0xFB);
```

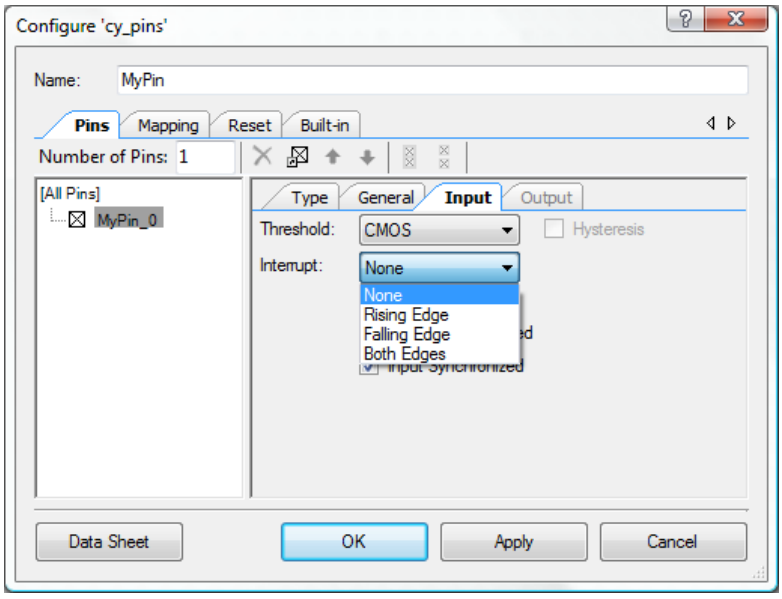
Table 9. Threshold Parameter

Threshold	
This parameter selects the threshold levels that define a logic HIGH level (1) and a logic LOW level (0). The setting applies to all physical pins in the port. Only CMOS and LVTTL settings are valid for GPIO pins.	
PSoC Creator Configuration Window	**Component APIs**
	N/A **Per-Pin APIs** N/A **Associated Registers** PRTx_CTL[0]
Code Examples	

```
/* Set port 1 logic threshold to CMOS using a register write */
       CY_SET_REG8(CYREG_PRT1_CTL, 0);

/* Set port 1 logic threshold to LVTTL using a register write */
       CY_SET_REG8(CYREG_PRT1_CTL, 1);
```

Table 10. Interrupt Parameter

Interrupt	
This parameter selects the interrupt type of the GPIO.	
PSoC Creator Configuration Window	**Component APIs**
	N/A
	Per-Pin APIs
	N/A
	Associated Registers
	PICUx_INTTYPEy[1:0]

Code Examples

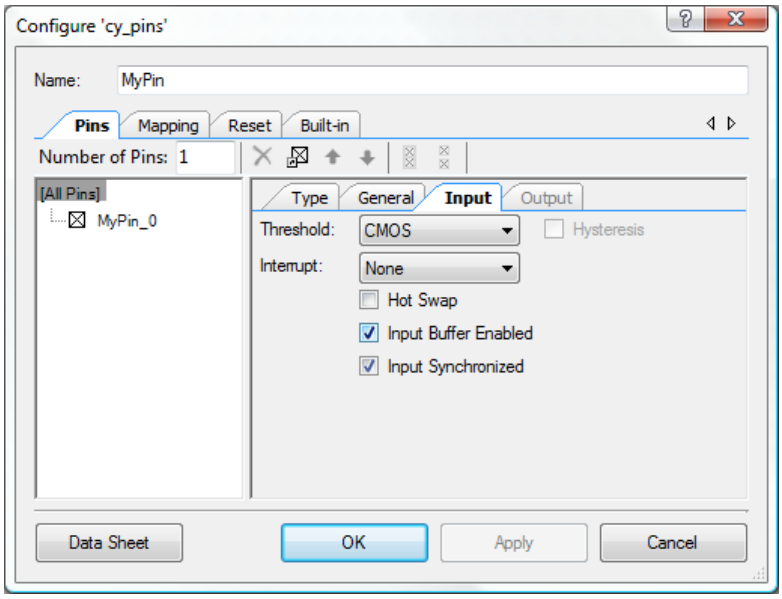
```
/* Disable interrupt on P1[2] */
CY_SET_REG8(CYREG_PICU1_INTTYPE2, 0x00);

/* Enable rising-edge interrupt on P1[2] */
CY_SET_REG8(CYREG_PICU1_INTTYPE2, 0x01);

/* Enable falling-edge interrupt on P1[2] */
CY_SET_REG8(CYREG_PICU1_INTTYPE2, 0x02);

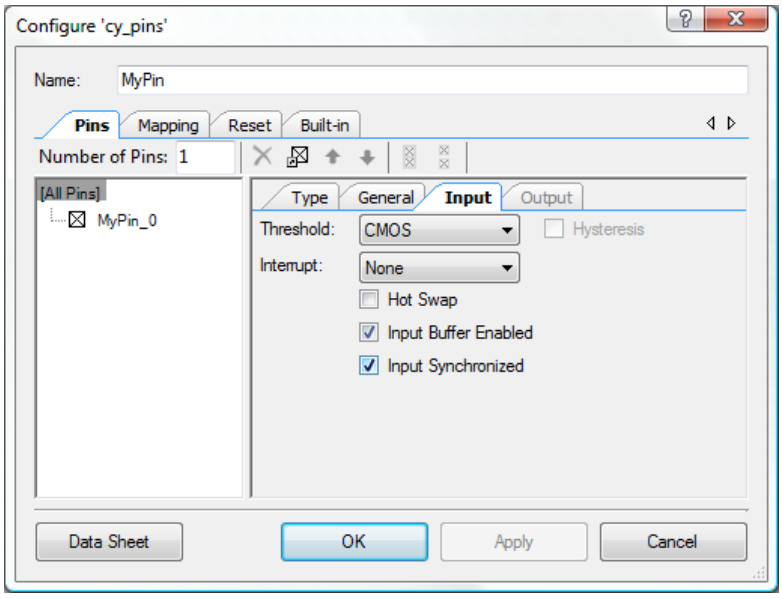
/* Enable any edge interrupt on P1[2] */
      CY_SET_REG8(CYREG_PICU1_INTTYPE2, 0x03);
```

Table 11. Input Buffer Enabled Parameter

Input Buffer Enabled	
This parameter determines if the pin's digital input buffer is enabled.	
PSoC Creator Configuration Window	**Component APIs**
	N/A
	Per-Pin APIs
	N/A
	Associated Registers
	PRTx_INP_DIS[7:0]
Code Examples	

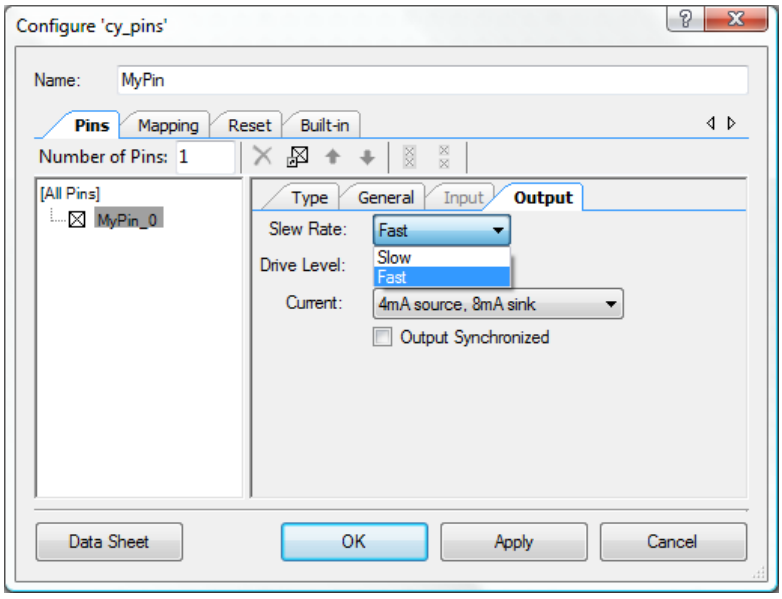
```
/* Disable Input Buffer on P1[2] using register write. */
CY_SET_REG8(CYREG_PRT1_INP_DIS, CY_GET_REG8(CYREG_PRT1_INP_DIS) | 0x04);
```

Table 12. Input Synchronized Parameter

Input Synchronized	
This parameter enables synchronization of the input of the pin to the bus clock.	
PSoC Creator Configuration Window	**Component APIs**
	N/A
	Per-Pin APIs
	N/A
	Associated Registers
	PRTx_DBL_SYNC_IN[7:0]
Code Examples	

```
/* Sync input of P1[2] to bus_clk using register write. */
      CY_SET_REG8(CYREG_PRT1_DBL_SYNC_IN, CY_GET_REG8(CYREG_PRT1_DBL_SYNC_IN) | 0x04);
```

Table 13. Slew Rate Parameter

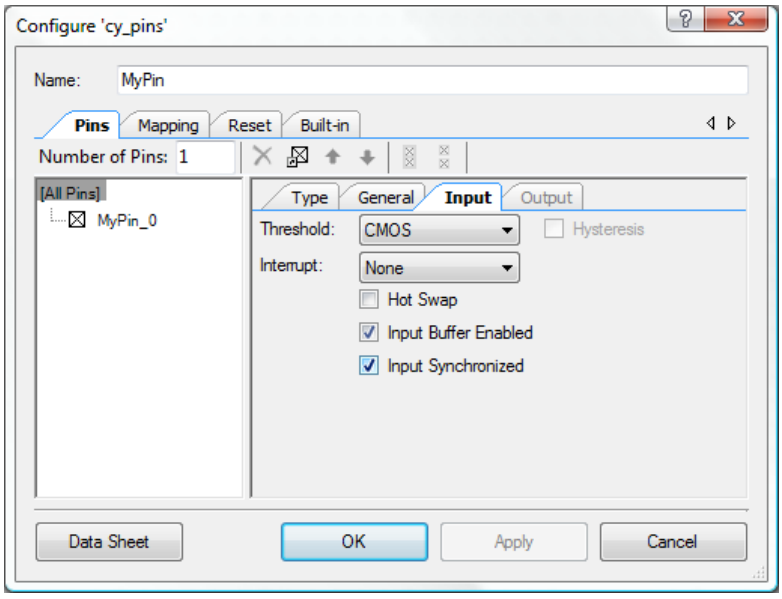
Slew Rate	
This parameter determines the rise and fall ramp rate for the pin as it changes output logic levels.	
PSoC Creator Configuration Window	**Component APIs**
	N/A **Per-Pin APIs** CyPins_FastSlew(CYREG_PRTx_PCy) CyPins_SlowSlew(CYREG_PRTx_PCy) **Associated Registers** PRTx_SLW[7:0]
Code Examples	

```
/* Set fast edge rate for P1[2] */
CyPins_FastSlew(CYREG_PRT1_PC2);

/* Set slow edge rate for P1[2] */
CyPins_SlowSlew(CYREG_PRT1_PC2);

/* Set slow edge rate for P1[2] */
       CY_SET_REG8(CYREG_PRT1_SLW, CY_GET_REG8(CYREG_PRT1_SLW) |= 0x02);
```

Table 14. Output Synchronized Parameter

Output Synchronized	
This parameter synchronizes the output drivers of the pin to the bus clock.	
PSoC Creator Configuration Window	**Component APIs**
	N/A
	Per-Pin APIs
	N/A
	Associated Registers
	PRTx_SYNC_OUT[7:0]
Code Examples	

```
/* Sync output of P1[2] to bus_clk using register write. */
      CY_SET_REG8(CYREG_PRT1_SYNC_OUT, CY_GET_REG8(CYREG_PRT1_SYNC_OUT) | 0x04);
```

Table 15. POR Parameter

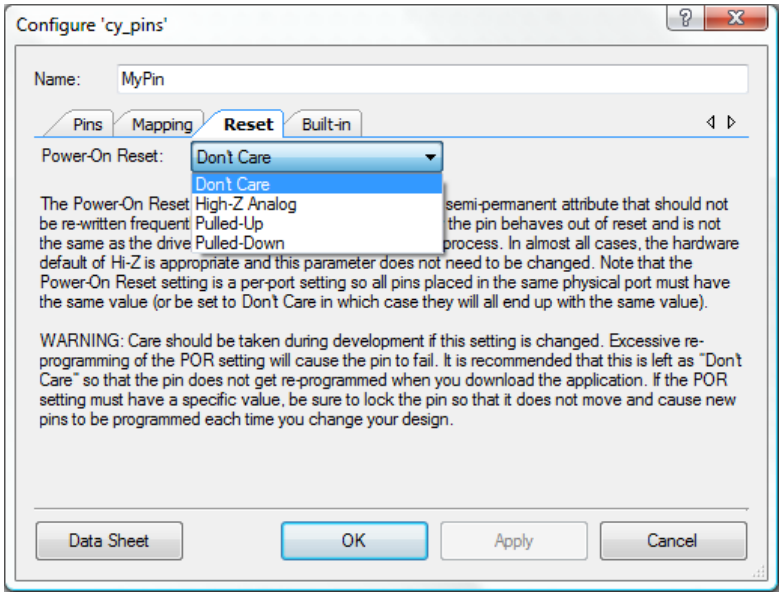
Power-On Reset (NVL Array)	
This parameter determines how the pin behaves during reset. It is not the same as the operating drive mode, which is configured during the boot process. Note that the POR setting is a per-port setting, which requires all pins placed in the same physical port to have the same value. This register is part of the NVL array and cannot be changed during normal operation.	
PSoC Creator Configuration Window	**Component APIs**
	N/A
	Per-Pin APIs
	N/A
	Associated Registers
	PRTxRDM[1:0]
Code Examples	
See the PSoC 3 Device Programming Specifications or PSoC 5LP Device Programming Specifications documents for details and instructions on programming the NVL array.	

Table 16. Enable Optional XRES Parameter

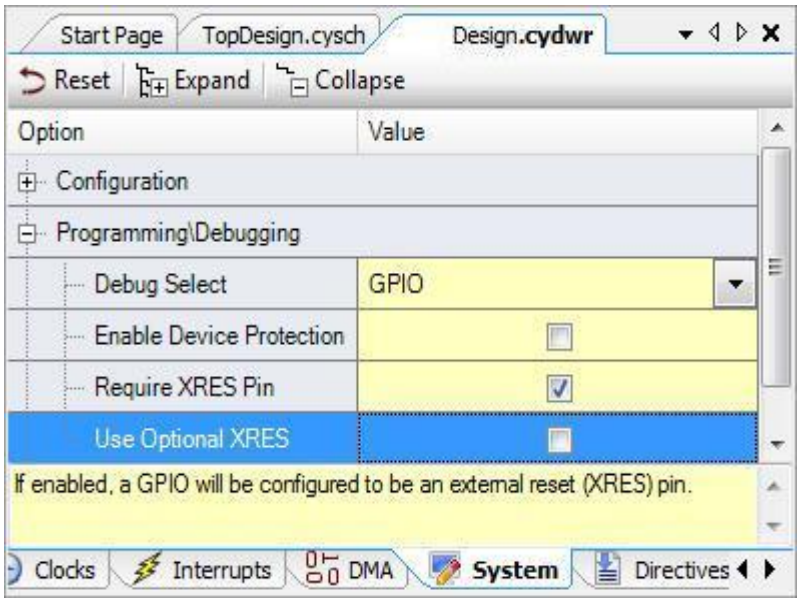
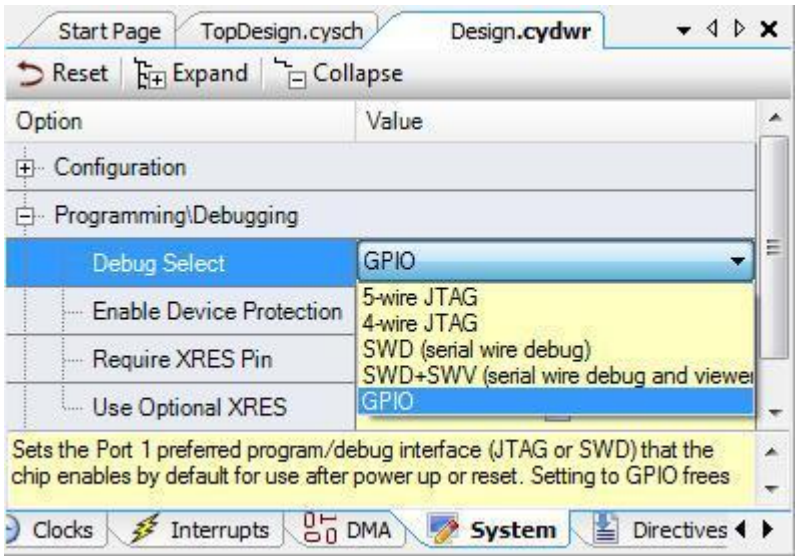
Enable Optional XRES (NVL Array)	
This parameter determines whether P1[2] will be configured as an external reset (XRES) pin. This register is part of the NVL array and cannot be changed during normal operation.	
PSoC Creator Configuration Window	**Component APIs**
	N/A **Per-Pin APIs** N/A **Associated Registers** XRESMEN[0]
Code Examples	
See the PSoC 3 Device Programming Specifications or PSoC 5LP Device Programming Specifications documents for details and instructions on programming the NVL array.	

Table 17. Debug Port Select Parameter

Debug Port Select (NVL Array)	
This parameter sets the preferred programming and debugging interface. This register is part of the NVL array and cannot be changed during normal operation.	
PSoC Creator Configuration Window	**Component APIs**
	N/A **Per-Pin APIs** N/A **Associated Registers** DPS[1:0]
Code Examples	
See the PSoC 3 Device Programming Specifications or PSoC 5LP Device Programming Specifications documents for details and instructions on programming the NVL array.	

Document History

Document Title: AN72382 - Using PSoC® 3 and PSoC 5LP GPIO Pins

Document Number: 001-72382

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3470167	GIR	12/20/2011	New application note
*A	3477593	GIR	12/28/2011	Updated Table 1 and the following text to reflect recent PSoC 5
*B	3526406	GIR	02/28/2012	Updated GPIO Power Structure and Limits section Updated Table 1, VDDIO Quadrant Current Limits Added Toggle section Updated Change PICU Settings with Firmware section.
*C	3820218	MKEA	11/26/2012	Updated for PSoC 5LP and PSoC Creator 2.1 SP1
*D	3884724	GIR	01/25/2013	Added GPIO interrupt sections – Port Interrupt Control Unit and Pins Component Interrupt. Added Special Function Registers section. Added additional text to the Introduction. Minor corrections and changes for overall look and feel.
*E	3942353	MKEA	03/22/2013	Added External Terminals section. Corrected some references to other application notes.
*F	4592891	GJV	12/10/2014	Removed reference to hidden pin component. Updated referenced application note names.
*G	5702158	BENV	04/19/2017	Updated logo and copyright
*H	6033709	GJV	01/16/2018	Added information on creation of a bi-directional data bus. Updated template

Worldwide Sales and Design Support

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

Products

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

PSoC® Solutions

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

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2011-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.