PSoC® Designer™ Quick Start Guide

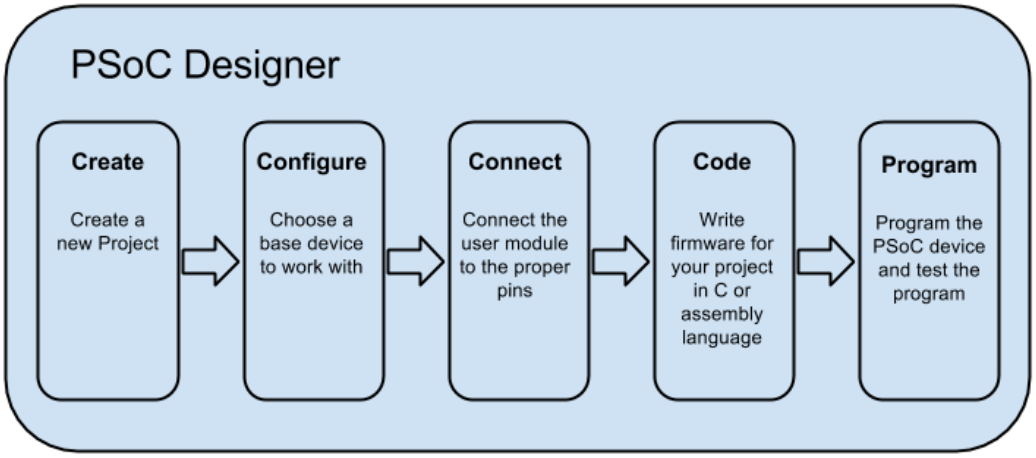
Installation

PSoC Designer is available for download at http://www.cypress.com/go/designer. You can also download an ISO image to create an installation CD. Each Starter Kit also includes an installation CD. If you need help installing PSoC Designer, call Cypress Support at 1-800-541-4736 and select 8.

After installation of PSoC Designer you can access PSoC Designer from the **Start** menu. To start PSoC Designer, click **Start > All Programs > Cypress > PSoC Designer [Version] > PSoC Designer [Version]**.

PSoC Designer Flow

Following are the steps to use PSoC Designer

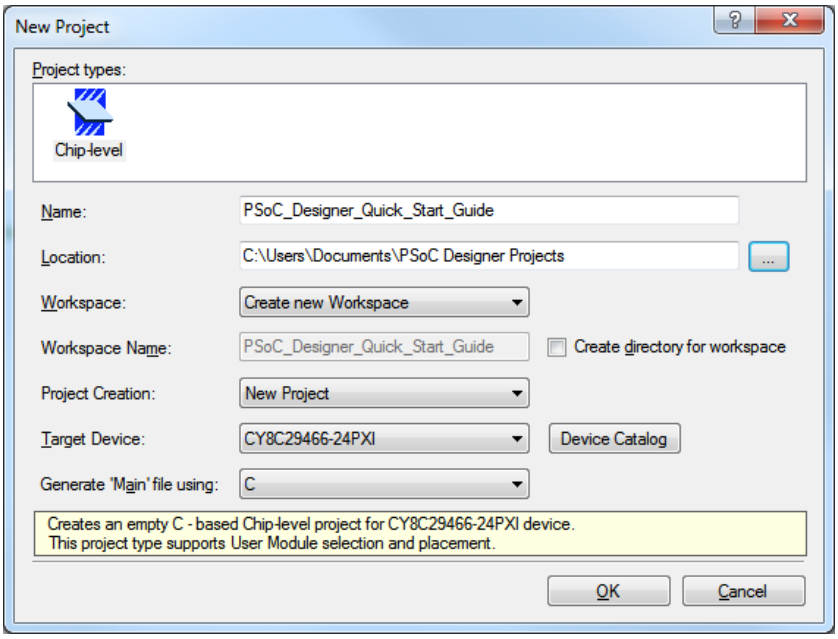


This guide helps you to program a PSoC device through a quick start project using PSoC evaluation boards such as CY3210-PSoCEval1 using a CY8C29x66 device. The project reads the voltage from a potentiometer and sets the first LED blinking depending on the voltage level. The second LED indicates that the potentiometer is near its low or high limits. When you are finished with this guide you will be familiar with the entire design process.

Step 1: Create a New Project

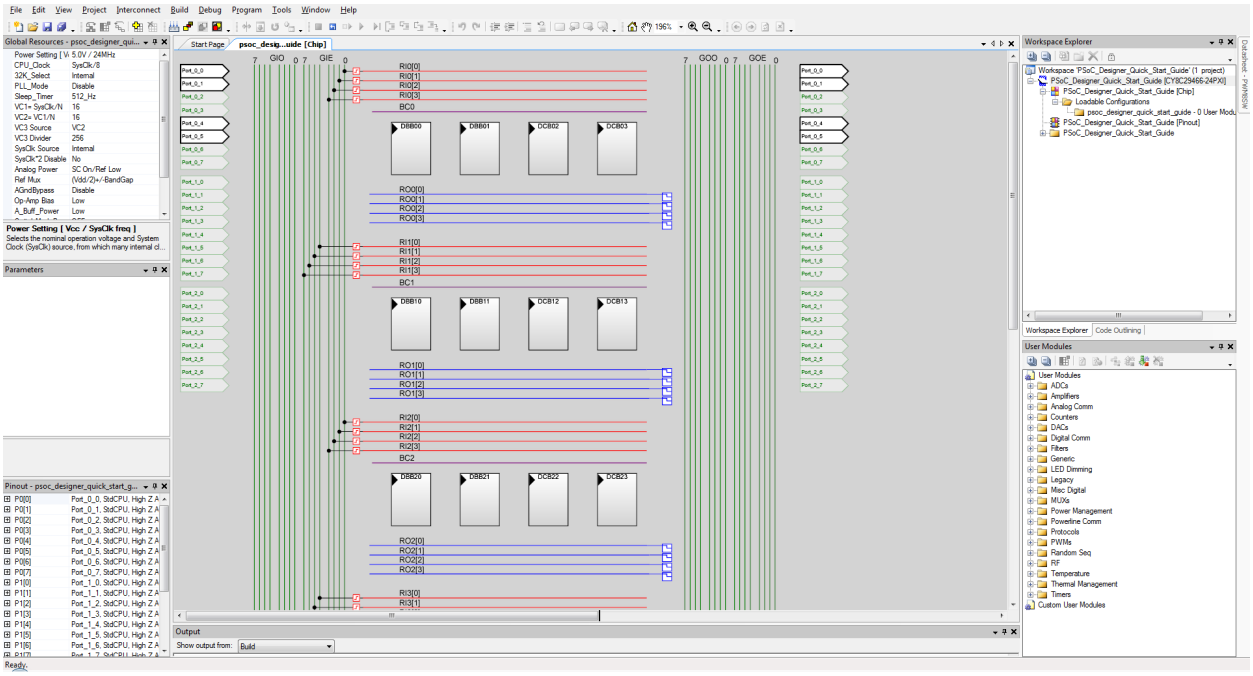
In PSoC Designer, you create a new project for your design.

1. Select **New Project…** item from the **File** menu.
2. Choose any desired name for the project and enter that under **Name.**



3. In the **Target Device** drop down menu, choose a part. CY3210-PSoCEval1 uses CY8C29466-24PXI. Consult your kit document for the proper device for your kit.
4. Under **Generate 'Main' file using**, select "C."
5. Click **OK**.

Your project opens in the **Chip Editor** view. The default view has secondary windows that show all kinds of information about your project. You can move, resize, close, and arrange all of these secondary windows to suit your preferences. The **View** menu contains the available windows. To restore a default window location you should select **Restore Default Layout** from the **Window** menu.



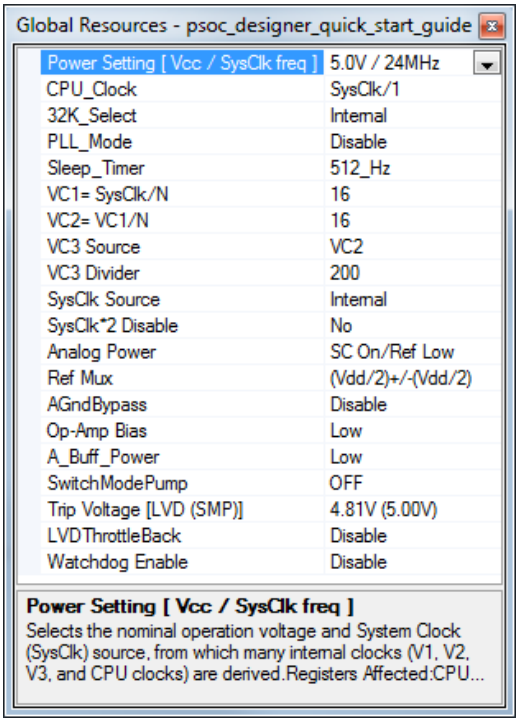
Step 2: Choose and Configure User Modules

This project consists of a potentiometer, two LEDs, and a PSoC 1 device. The first LED blinks with a rate defined by the potentiometer position. The second LED is On if the potentiometer is near its low or high limits. The limits can be changed in the code. The user modules required for this project are:

- **PGA** - A programmable gain amplifier is used to buffer the input from the potentiometer.
- **ADCINC** - An incremental ADC is used to convert the analog input from the potentiometer to a digital value that you can use for the program logic.
- **Counter8** – Two 8-bit counters are used to blink one of the LEDs periodically and turn on or off another LED.
- **LED** – The user module simplifies control of an LED or any simple device that is controlled by On and Off.

Global Resources

Global Resources are those shared by all user modules in a particular configuration. The IDE Guide (**Help > Documentation… -> Designer Specific Documents -> IDE User Guide.pdf**) contains a complete reference on the effect of each of the Global Resources.



1. Set the **CPU Clock** to “SysClk/1.”

   Since the Power Setting is at the default of the 5.0 V operation and a SysClk of 24 MHz, the CPU will also run at 24 MHz.

2. Set the **VC1** clock to “SysClk/16.”

   The setting makes VC1 = 24 MHz / 16 = 1.5 MHz.

3. Set the **VC2** clock to “VC1/16.”

   The setting makes VC2 = VC1 / 16 = 1.5 MHz / 16 = 93.75 kHz.

4. Set the **VC3 Source** to “VC2” and the **VC3 Divider** to “200.”

   The setting makes VC3 = VC2 / 200 = 93.75 kHz / 200 = 468.75 Hz.

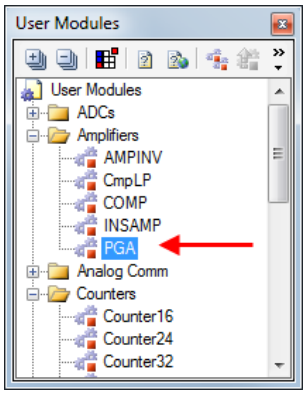
5. Set the **Ref Mux** to “(Vdd/2)+/-(Vdd/2)”.

The setting allows us to measure the voltage in the range from Vss to Vdd.

6. The rest of global parameters is unused in the example project and should be set to default values as it is shown in the figure above.

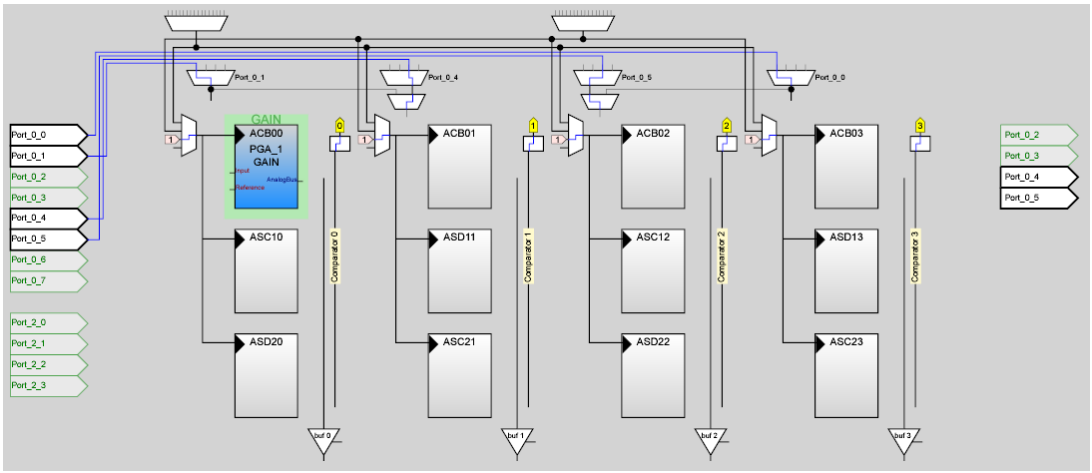
PGA

1. In the User Module Catalog (**View > User Module Catalog**), expand the **Amplifiers** folder.

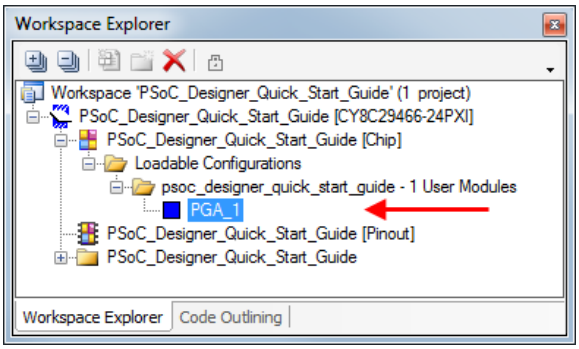


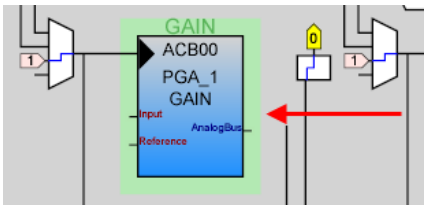
2. Double-click on the PGA user module to place it (you can do the same by right-clicking **PGA** and selecting**Place)**. *Note: you may need to scroll down in the chip editor using the side scroll bars to view the user module placement. To zoom in or out you can use your mouse, CTRL+Scroll Wheel up or down, or by using the "Zoom In" or "Zoom Out" buttons on the top menu bar.*

The PGA is placed in the first available analog block. The default placement of the user module is sufficient.

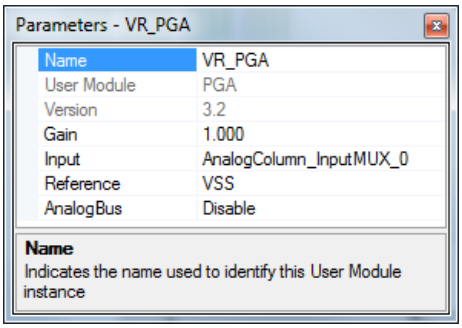


3. Click on the PGA user module in the Workspace Explorer (**View -> Workspace Explorer**) or Chip Editor (**View -> Chip Editor**) to select it.





4. Change the name of the user module to VR_PGA in the Parameters window (**View > Parameters Window**).



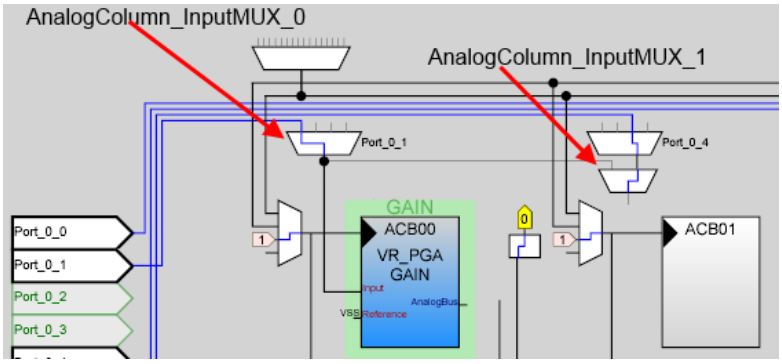
Note that the user module name in the Chip Editor updates with the new name.

5. Set the **Gain** for VR_PGA to "1.000."

   The PGA is used to buffer the input of the potentiometer, so the Gain is 1.

6. Set the **Input** to "AnalogColumn_InputMUX_0."

   The input for the PGA will come from the potentiometer routed from a pin to AnalogColumn_InputMUX_0. By default, AnalogColumn_InputMUX_0 is routed from P0[1]. You can click the mux and select one of the four pins, but the default works for this purpose.



7. For the VR_PGA **Reference**, select ground, "VSS."

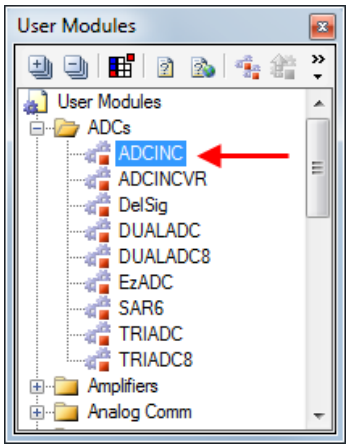
   The PGA reference is dependent on the Ref Mux global parameter. To measure voltages from Vss to Vdd, the **Ref Mux** global parameter should be set to "(Vdd/2)+/-(Vdd/2)"and PGA **Reference** should be set VSS.

8. For **Analog Bus**, choose "Disable."

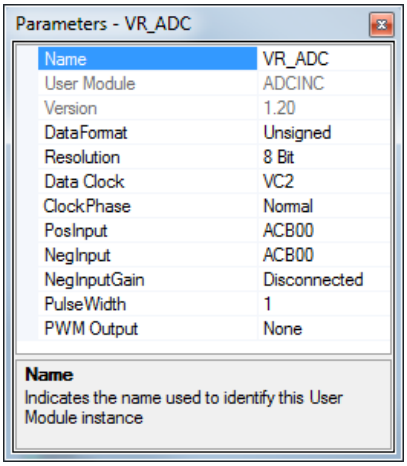
ADCINC

There are many ADCs available. When choosing an ADC for your application, consult the *ADC Selection Guide* linked at the beginning of each of the ADC datasheet. This project will use the ADCINC.

1. In User Modules, select the **ADCs** folder. Double click to place **ADCINC**.



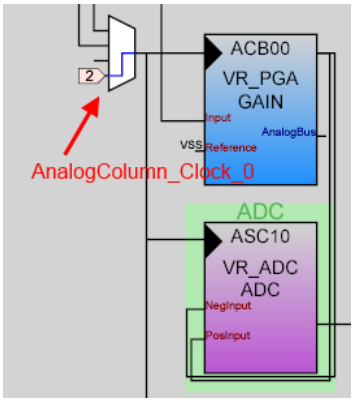
2. Select **Single Stage Modulator** and select **OK**.
3. In the Parameters Window, change **Name** of the user module to “VR_ADC.”



4. Select “Unsigned” for **DataFormat**.
5. For **Resolution**, select “8 Bit.”

   The resolution of 8 bit is enough for the example project and it matches the Counter’s resolution for easier control.
6. For **Data Clock**, select “VC2.”

   Data Clock should be the same for both digital and analog blocks. The parameter sets a clock for digital blocks only. You should click on AnalogColumn_Clock_0 and select VC2. The selection makes the ADC clock equal to 93.75 kHz.



According to the ADCINC user module we will get the following Sample Rate of the ADC:

Sample Rate = DataClock / (256 x (2^{Bits-6}+1)) = 93.75 kHz / (256 x 5) = 73 Hz

7. For **PosInput**, select “ACB00.”

   This connects the output of the **VR_PGA** GAIN block to the PosInput of the **VR_ADC**. Check the diagram to make sure that there is a line connecting them. If you are using a chip that has a different configuration of analog blocks than CY8C29466-24PXI used in the example, choose the block that contains the VR_PGA GAIN block in place of **ACB00**.

8. Set **NegInput** to “ACB00” and **NegInput Gain** to “Disconnected.”

   This sets the ADC to use a single-ended input.

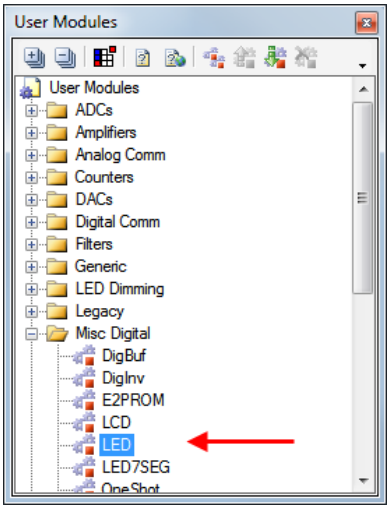
9. Set **PWMWidth** to 1 and **PWM Output** to “None.”

   We do not use the PWM output in the example project.

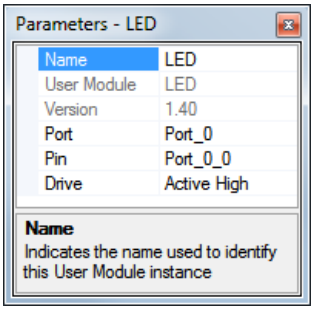
LED

The LED user module simplifies control of the output pins. It avoids writing the logical AND and OR commands to change a pin state in the main code.

1. In User Modules, select the **Misc Digital** folder, and place the **LED**.



2. In the Parameters Window, change **Name** of the user module to “LED”.



3. Select “Port_0_0” for Output Pin.
4. For **Drive**, select “Active High”.

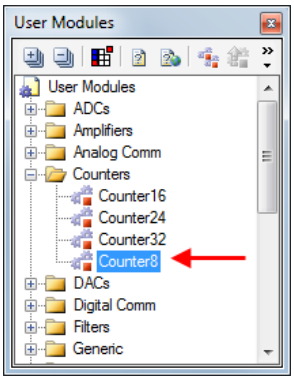
Counter8

Two 8-bit down counters are used in the project:

- the first one indicates the potentiometer’s low or high limits (using a software interrupt routine) and provides a clock for the second counter..

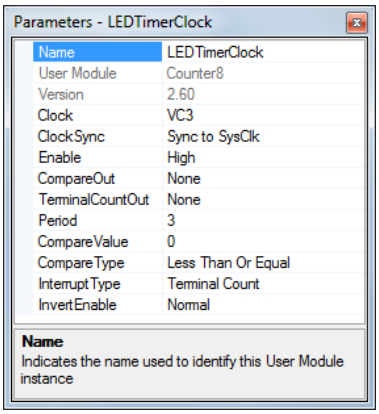
- the second one is used to flash the LED periodically.

In User Modules, select the **Counters** folder, and place two Counter8 user modules.



Adapting the first instance of Counter8 user module

1. In the Parameters Window, rename the user module to “LEDTimerClock”.



2. Set **Clock** to “VC3.”

   The counter is clocked by a VC3 system clock that is equal to 468.75 Hz.

3. Set **ClockSync** to “Sync to SysClk.”

The counter clock is synchronized to the SysClk (24 MHz) clock.

4. Set **Enable** to “High” and **InvertEnable** to “Normal”.

   The user module is enabled all time.

5. Set **CompareOut** and **TerminalCountOut** to “None”

   We do not use any of the Counter’s outputs.

6. Set **Period** to “3.”

   It provides the following output clock:

   OutClock = VC3 / (Period + 1) = 468.75 Hz / (3 + 1) = 117 Hz

   The terminal count will occur 117 times per second and each of these terminal counts compares a measured voltage with a high and low limit in the software routine.

7. Leave **CompareValue** with the default value and set any value for the **CompareType** parameter.

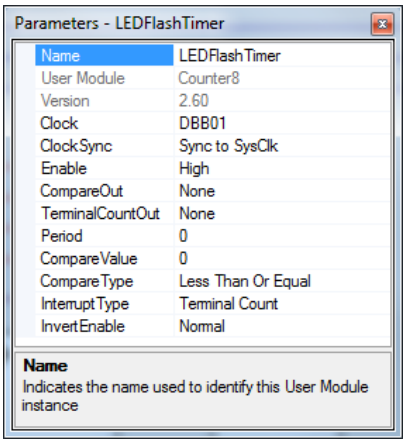
   We don’t use a compare output therefore the values have no influence on the project operation.

8. Set any value for the **InterruptType** parameter.

   Independent on the parameter value, the interrupt appears once per a Counter period.

Adapting the second instance of Counter8 user module

1. In the Parameters Window, rename the user module to “LEDFlashTimer”.



2. Set **ClockSync** to “Sync to SysClk.”

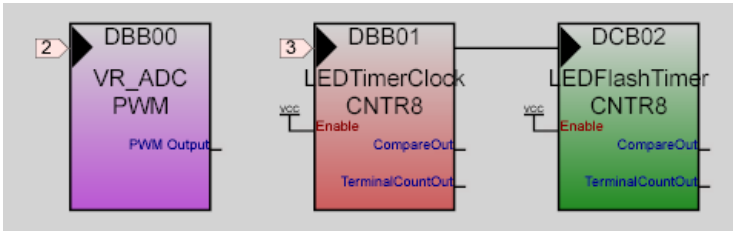
   The counter clock is synchronized to SysClk (24 MHz) clock.

3. Set **Enable** to “High” and **InvertEnable** to “Normal”.

   The user module is enabled all time.

4. Set the **Clock** to “DBB01”.

   The counter is clocked by LEDTimerClock that occupies a previous digital block and generates an output clock equal to 117 Hz.



5. Leave **Period** and **CompareValue** with default values.

   Initial values of the parameters have no influence on the project operation because the parameters are overwritten in the firmware on fly.

6. Set **TerminalCountOut** to “None”

   We do not use the Terminal Count output in the project.

7. Set any value for the **InterruptType** parameter.

   We do not use the counter‘s interrupts.

8. Set the **CompareType** to “Less Than Or Equal”.

   The parameter is more suitable for the project.

9. How to adapt the **CompareOut** parameter is described in the next section.

Step 4: Connect the User Modules

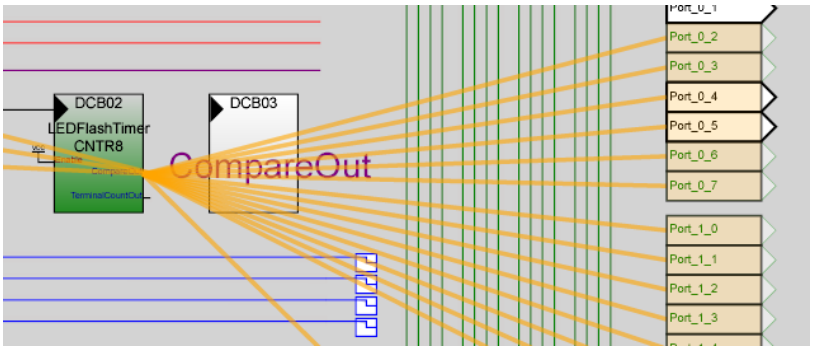
Each user module has inputs and outputs that can be routed to other user modules and pins. The output of the VR_PGA user module is routed to the VR_ADC user module in the parameter selection. So to practice routing, you will route the output of LEDFlashTimer CompareOut to a pin to flash an LED when the counter reaches the terminal count.

The low/high voltage indicating LED is connected to Port_0_0 (P0[0]), and the potentiometer input is on Port_0_1, so you will route this signal to Port_0_2. You should connect the second LED on your board to Port_0_2 as well.

There are two ways to route a block input/output to the pins – automatic and manual. Automatic is preferred in most cases, but manual provides additional flexibility for complex projects. Both of the methods are described below. For the first example you can limit by the first method, and try the second method later to get more familiar with power of the PSoC1 interconnect.

Auto routing

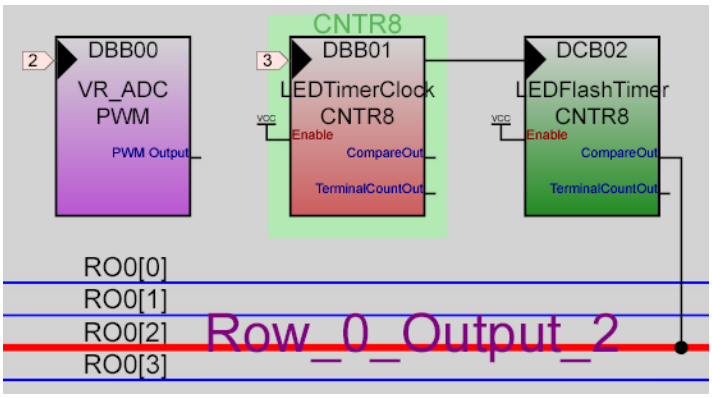
1. Press Shift and select CompareOut of LEDFlashTimer user module. When you click the output PSoC Designer shows all possible connections for the output.



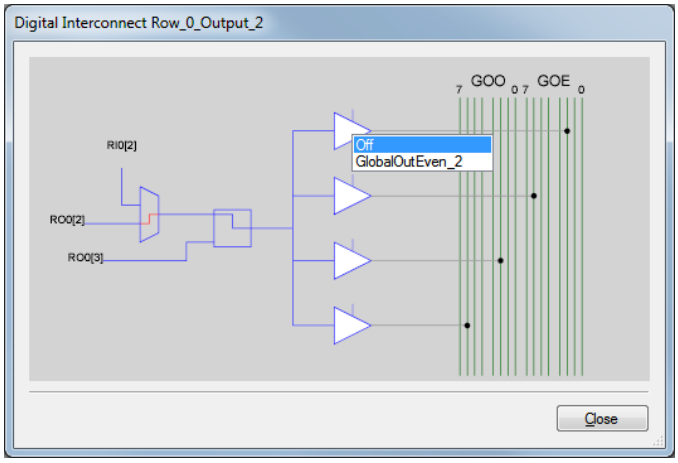
2. Click the destination of the route (Port_0_2) with pressed Shift to complete.
3. To clear an existing route, press Shift and select a block output or input of an existing route. PSoC Designer then highlights the existing routes and the remaining available outputs, pins, and resources. When you click the destination of the existing route it is disconnected.

Manual method

1. Set (connect) the **CompareOut** output of the **LEDFlashTimer** user module to “Row_0_Output_2”.
2. Click the RO0[2] (Row_0_Output_2) line in the Interconnect view. It is the blue horizontal line that your LEDFlashTimer is attached to. The line becomes red if selected.

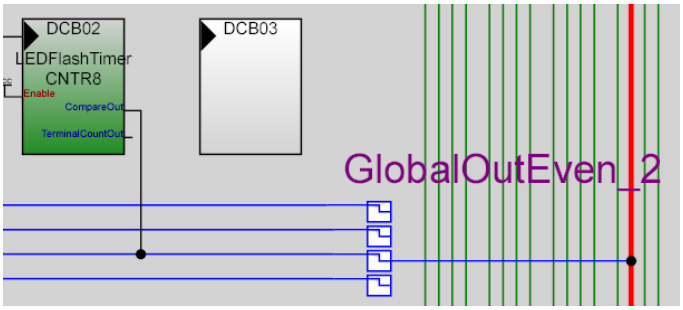


3. The Digital Interconnect dialog appears. Click the top triangle in the dialog window and select GlobalOutEven_2. Triangle fills blue if selected.

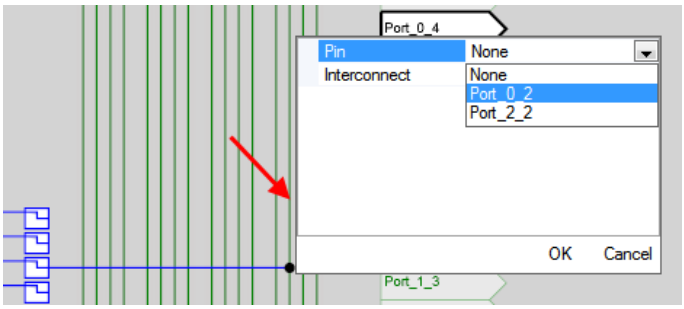


4. Click **Close**.

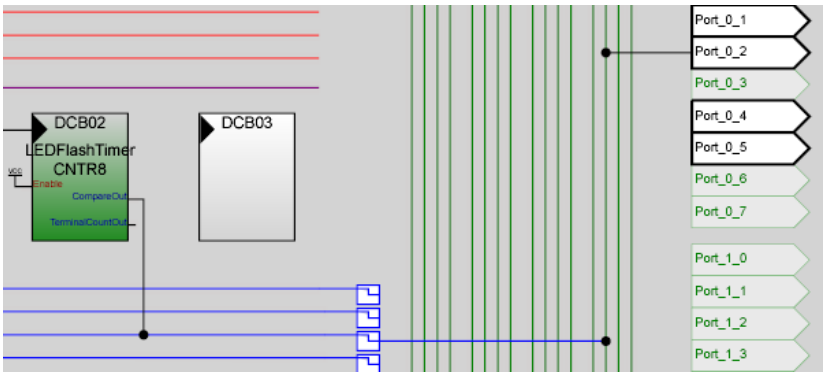
5. Chip editor will show the new connection: Row_0_Output_2 to GlobalOutEven_2 (GOE[2]) (vertical green line on the right side of the digital blocks that become red if selected)



6. Click GlobalOutEven_2 and select Port_0_2.



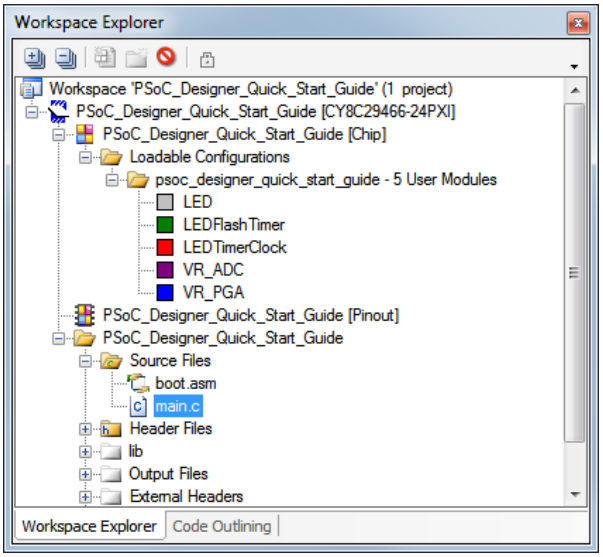
7. In the Interconnect view, there is now a line from CompareOut of the LEDFlashTimer to RO0[2], then to GlobalOutEven_2, and finally to Port_0_2. You have successfully routed Compare Out of your timer user module to a pin and configured the pin for the output attached to it.



Step 5: Write Firmware

The first step in writing the firmware is generating an application. Generating an application creates all of the source files, library files, and headers that are needed to begin writing the firmware. The APIs generated for each of the user modules you have selected are detailed in the user module data sheets associated with each of the user modules.

1. From the **Build** menu, select **Generate/Build Project**.
2. In t Workspace Explorer, double click to expand your project folder.
3. Under the Source Files folder, double click *main.c*.



4. Copy and paste the following code into *main.c*. Replace the entire contents of *main.c* with the text below.

   The code is fully commented, so you may want to look through the program logic. It starts all five user modules. The ADC begins sampling before the program goes into an infinite loop that samples the ADC and performs some simple logic on the results. After the main program, there is an interrupt service routine written in C.

```
//-----------------------------------------------------------------------------
// C main line
//-----------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

#pragma interrupt_handler LEDTimerClock_ISR
                /* Write the interrupt handler for the LEDTimerClock in C.*/

#define   ADC_LOW_LIMIT                  40
```

```
#define   ADC_HIGH_LIMIT                  210

                /*Constant definition for ADC low and high limits to turn LED On or Off*/

unsigned char ucVR_ADCResult;
                /*This global variable holds the converted output of the potentiometer (VR).*/

void main(void)
{
    M8C_EnableGInt;                       /*Enables the Global Interrupts*/

    LED_Start();            /* Turn off the LED on Port_0_0  */


    LEDTimerClock_Start();
        /*Enables the LEDTimerClock User Module. */

    LEDTimerClock_EnableInt();
        /*Enables interrupts of the LEDTimerClock User Modules that are used to check
        the potentiometer low and high limits. */


    LEDFlashTimer_Start();
        /*Enables the LEDFlashTimer User Module that blinks the led on Port_0_2. */


         VR_PGA_Start (VR_PGA_HIGHPOWER);
                 /*Performs all required initialization for the PGA User Module and sets the power
                 level for the PGA to high power (VR_PGA_HIGHPOWER).*/


         VR_ADC_Start(VR_ADC_HIGHPOWER);
                 /*Performs all required initialization for the VR_ADC User Module
                 and sets the power level to high power. */

         VR_ADC_GetSamples(0);
                 /*Sets the VR_ADC to run continuously by providing a 0 in the paramater list.*/

         while(1)                                  /* Infinte loop */
         {
                 if (VR_ADC_fIsDataAvailable() != 0)
                         /*This function checks the availability of sampled data.  The function returns
                         a non-zero value if data has been converted and is ready to read.*/
                 {
                         ucVR_ADCResult = VR_ADC_bClearFlagGetData();
                         /* This function clears the data ready flag and gets converted data as an
                         unsigned char and stores it in the variable ucVR_ADCResult.
                         This function also checks to see that data-flag is still reset.
                         If not the data is retrieved again. This makes sure that the ADC interrupt
                         routine did not update the answer while it was being collected. */

            LEDFlashTimer_WritePeriod(ucVR_ADCResult);
            /* Sets LEDFlashTimer's period equeal to a measured voltage in the ADC counts */

            LEDFlashTimer_WriteCompareValue(ucVR_ADCResult / 2);
            /* Sets LEDFlashTimer's compare value equal to a half of a period to produce a square waveform */
                 }
         }
}

/* The ISR handler of LEDTimerClock User Module*/
void LEDTimerClock_ISR(void)
{
    /* Check if an ADC result exceeds low or high limits*/
    if ((ucVR_ADCResult < ADC_LOW_LIMIT) || (ucVR_ADCResult > ADC_HIGH_LIMIT))
    {
        LED_On();
    }
    else
    {
        LED_Off();
    }
}
```

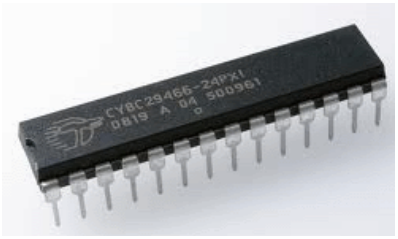
5. Save *main.c*.
6. From the **Build** menu, select **Generate/Build Project**.

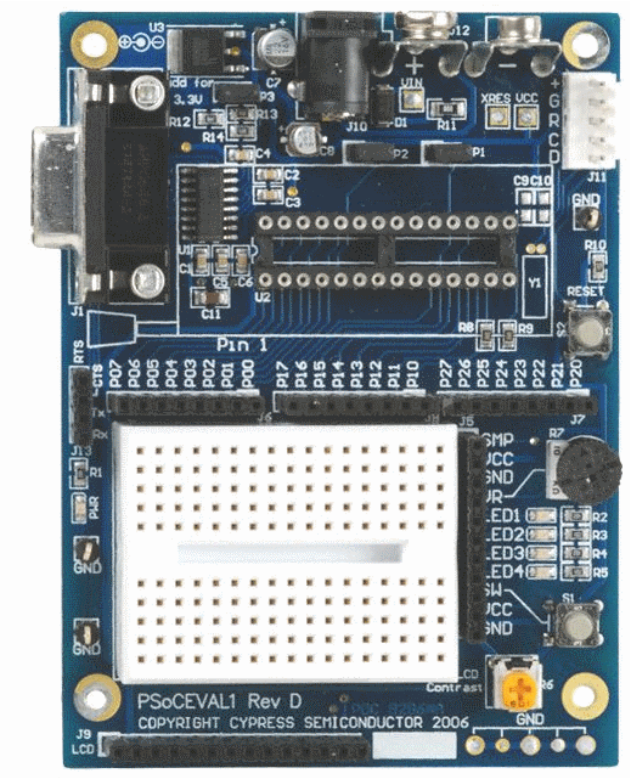
You are now ready to program your PSoC device.

Step 6: Program and Test Your PSoC Device

We will use MiniProg1 to program the CY8C29466-24PXI device and CY3210-PSoCEval1 board to test the project.

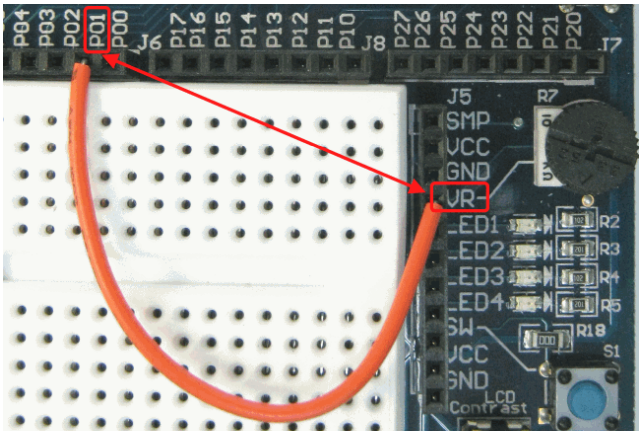
MiniProg1 is included into CY3210-PSoCEval1 kit. Also it can be ordered separately as CY3217-MiniProg1 kit.



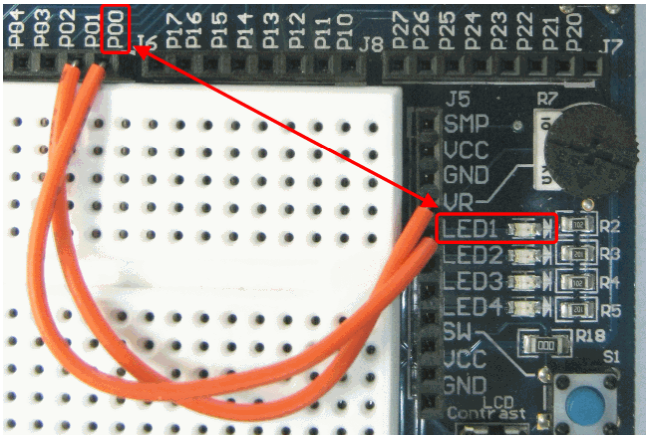


We will use wires to connect PSoC pins to LEDs and potentiometer.

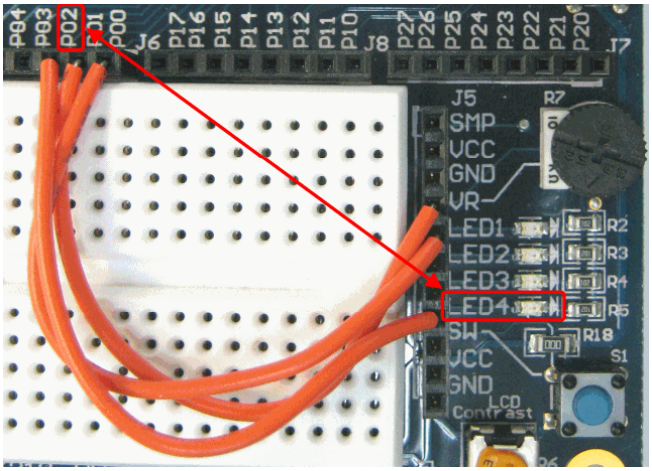
1. Connect the potentiometer on the board to the pin selected as PGA input – Port_0_1 in the example above.



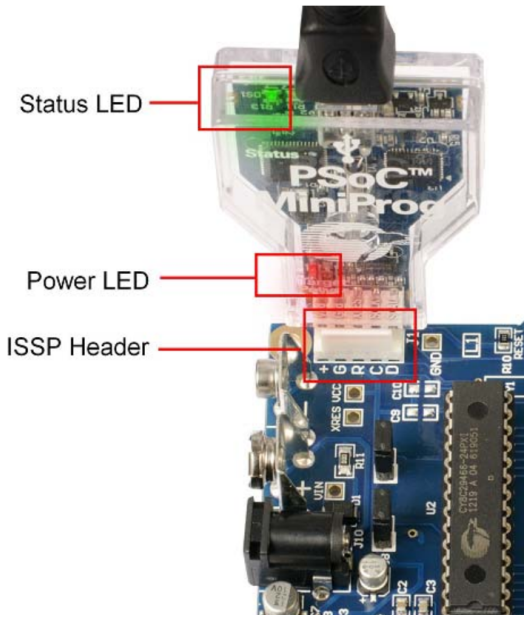
2. Connect the first LED on the board to the output pin of the LED User Module - Port_0_0 in the example above.



3. Connect the second LED on the board to the Compare output of the LEDFlashTimer User Module - Port_0_2 in the example above.

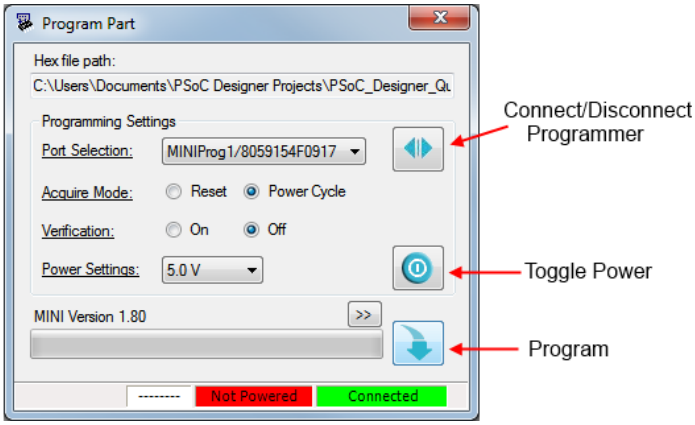


4. Place your PSoC chip on the evaluation board, and plug your programmer into the Target ISSP connector.

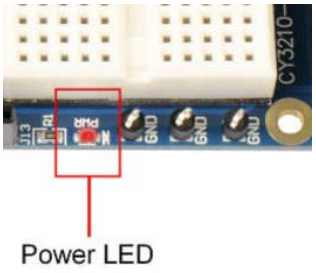


5. Connect the MiniProg1 programmer to a PC through a USB cable. The Status LED on the MoniProg1 should turn on.
6. Select **Program>Program Part** from the menu.

   The embedded programmer application opens.



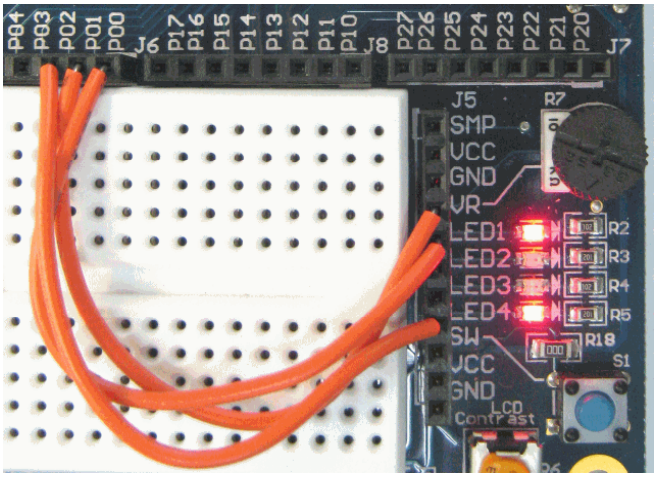
7. Click the **Program** button, and the program starts being loaded into your PSoC chip. The Power LEDs are On on both MiniProg1 and CY3210 board while programming.



8. Press the **Toggle Power** button on the programmer window to check that the program works as desired.
9. Check to see if the program works as desired:

   - the LED connected to Port_0_0 is On if the potentiometer is near to its low or high limits;

   - the LED connected to Port_0_2 blinks with rate defined by the potentiometer position.



Beyond This Example

This introductory example has familiarized you with the basic design methodology of PSoC Designer and resulted in a functional PSoC design. However, the example did not address many of the more advanced aspects of PSoC Designer, nor did it explain why some of the design decisions had been made.

More information about PSoC Designer is available in the PSoC Designer IDE Guide found in the **Help > Documentation** directory.

More training on PSoC Designer and other Cypress solutions is available at http://www.cypress.com/training/.