CY8CKIT-001

PSoC® Development Kit Guide

Doc. # 001-48651 Rev. **
June 16, 2009

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

Copyrights

Flash Code Protection

Cypress products meet the specifications contained in their particular Cypress PSoC Data Sheets. Cypress believes that its family of PSoC products is one of the most secure families of its kind on the market today, regardless of how they are used. There may be methods, unknown to Cypress, that can breach the code protection features. Any of these methods, to our knowledge, would be dishonest and possibly illegal. Neither Cypress nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Cypress is willing to work with the customer who is concerned about the integrity of their code. Code protection is constantly evolving. We at Cypress are committed to continuously improving the code protection features of our products.

Contents

1. Introduction

1.1 Kit Overview

The CY8CKIT-001 PSoC® Development Kit provides you a common development platform where you can prototype and evaluate different solutions using any one of the PSoC, PSoC3, or PSoC5 architectures. This guide and kit gives you a practical understanding of PSoC technology. In addition, the kit gives several example projects with step-by-step instructions to enable you to easily get started developing PSoC solutions. This kit includes PSoC CY8C29 and PSoC3 CY8C38 Family Processor Modules.

1.2 Kit Contents

The CY8CKIT-001 PSoC® Development Kit includes:

- PSoC development board
- PSoC3 CY8C38 Family processor module
- PSoC CY8C29 Family processor module
- MiniProg3 Programmer and Debug tool
- USB cable
- 12V power supply adapter
- Wire pack
- Printed documentation
  - Quick Start
  - Schematic PSoC development board design
- A kit CD, which includes
  - PSoC Designer™ IDE for PSoC
  - PSoC Creator™ IDE for PSoC3/PSoC5
  - PSoC Programmer™ software
  - CY8C29 and CY8C38 data sheets
  - Architecture technical reference manuals
  - Kit release notes
  - Software release notes
  - Example project files, firmware, and documentation

1.3 Installation

Everything you need to use the PSoC Development Kit is included, but you only need to install the software for the processor module you plan to use.

1.3.1 Before You Begin

All Cypress software installations require administrator privileges but is not required to run the installed software.

Shut down any currently running Cypress software.

Disconnect any ICE Cube or MiniProg devices from your computer.

1.3.2 Prerequisites

PSoC Creator and PSoC Designer both use the Microsoft .NET Framework, Adobe Acrobat Reader, and a Windows Installer. If these are not on your computer, the installation automatically installs it for you.

1.3.3 Installing PSoC Development Software

To use the CY8C29x66 Family Processor Module (PSoC) you need:

- PSoC® Designer™ 5.0 SP4.5 or higher
- PSoC® Programmer™ 3.10 or higher

If PSoC Designer 5.0 is currently installed, uninstall it. Click **Start**, click **Control Panel**, and then double click **Add or Remove Programs**.

PSoC Designer 5.0 SP4.5 is not presently compatible with Microsoft Internet Explorer 8 beta (any beta version). If you have Internet Explorer 8 beta, uninstall it and replace it with the released version of Microsoft Internet Explorer 8.

The CD includes PSoC Programmer 3.10 Beta, which is an optional update for PSoC users.

Insert the CD and, using the menu, install “PSoC Designer for PSoC” and “PSoC Programmer” (if required).

After installation, user guides and key documents are located in the `\Documentation` subdirectory of the PSoC Designer installation directory. The default location is:
`C:\Program Files\Cypress\PSoC Designer 5\Documentation`

1.3.4 Installing PSoC3 Development Software

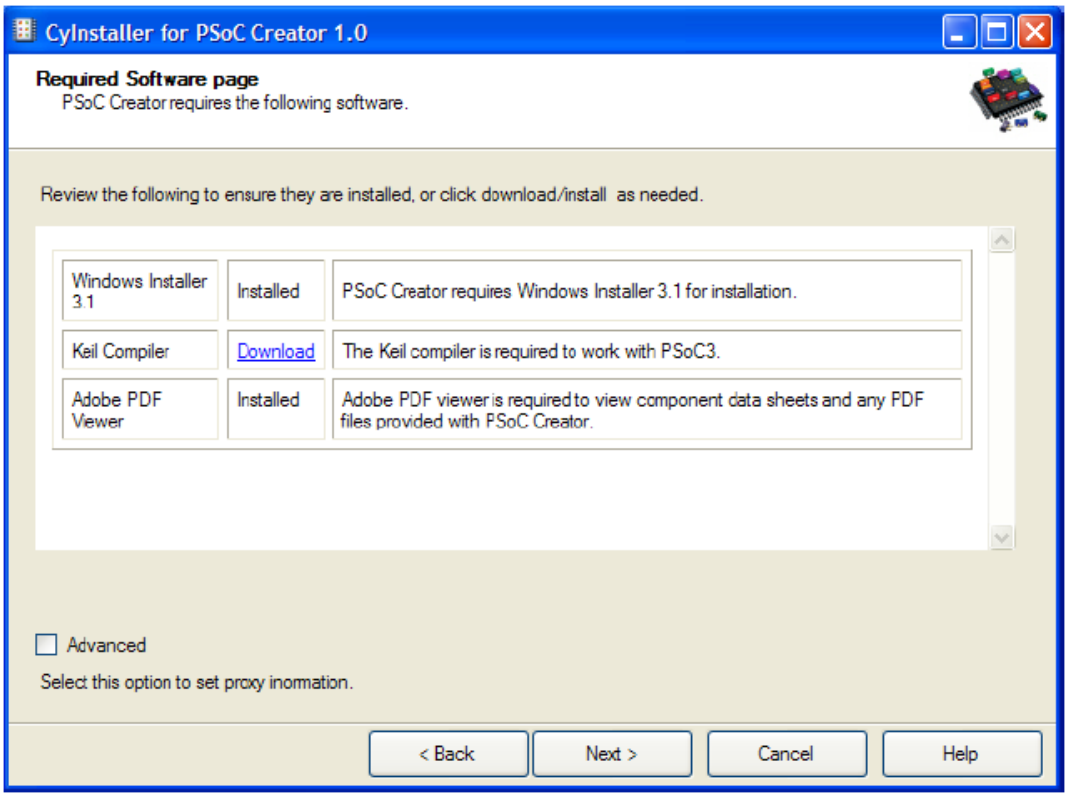
To use the CY8C38 Family Processor Module (PSoC3) you need:

- PSoC® Creator™ 1.0 Beta 2 or higher
- PSoC® Programmer™ 3.10 Beta or higher
- PSoC Development Kit example files

Insert the CD and, using the menu, install the PSoC Development Kit for PSoC3/PSoC5. This option installs all three required software packages. The installers for PSoC programmer and PSoC Creator automatically start before the Kit examples are installed.

For each installation, select **Typical** on the **Installation Type** page.

PSoC Creator uses the Keil CA51 compiler to build PSoC3 applications. You can download it, during the PSoC Creator installation, from the Required Software page:



Click the **Download** link and follow the prompts. You are asked to provide some basic user information.

Note The Keil compiler is distributed with a free license. You must activate this license within 30 days of installation. Once the Cypress software installation is complete, and you run PSoC Creator, activate the compiler license from the **Keil Registration** option under the **Tools** menu.

After installing PSoC Creator and PSoC Programmer, refer to the documentation as needed:

- **PSoC Creator → Help → Getting Started**
- **Programmer → User Guide**

Other documents included with this release are located in the `\Documentation` subdirectory of the PSoC Creator installation directory. The default location is:
`C:\Program Files\Cypress\PSoC Creator\1.0\PSoC Creator\documentation`

You can access this directory from within PSoC Creator under **Help → Documentation → Reference Material**. Documents include (but are not limited to)

- *PSoC Creator Component Author Guide*
- *Warp Verilog Reference Guide*
- *Customization API Reference*

1.4 PSoC Development Board

The CY8CKIT-001 PSoC Development Board is designed to aid hardware, firmware, and software developers in building their own systems around Cypress 8-bit and 32-bit PSoC devices. The flexibility to configure the power domains is one of the foremost features of this board. Input power to the board is from one of two sources:

- 12V 1A wall wart power supply
- 9V alkaline battery (not included)

This full featured board incorporates three onboard linear regulators that power peripherals and PSoC modules at voltages between 1.72V and 5.5V. These regulators include a fixed 5V 1A linear regulator, a fixed 3.3V 300 mA linear regulator, and a 1.5V to 5.5V 300 mA adjustable regulator. The board also provides the ability to separate the PSoC core VCC rail into two separate rails, analog and digital. In addition, the board is able to separate the I/O VCC rails, giving the flexibility to power the I/O ports at different voltages.

The board is equipped with a 2x16 alphanumeric LCD module capable of 1.8V to 5.5V I/O. In addition, there is a mini-B full speed USB interface and a female DB9 serial communications interface. Also included is a x-pin wireless radio module interface which can be used to develop CyFi™ Low-Power RF or other embedded RF solutions with this kit. The board also has a prototyping area containing a small bread board complete with I/O port sockets nearby, multipurpose LEDs, mechanical push buttons, and a multipurpose variable resistor. In addition, three capacitive sensing elements (two buttons and a five segment slider) are included on the board to allow evaluation of CapSense™ applications.

The board has four GPIO expansion slots, allowing the I/O to expand to external boards.

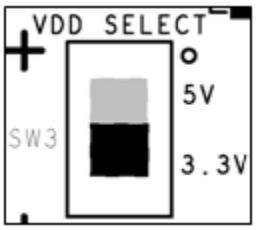
The board was designed with modularity in mind and, as a result, supports removable processor modules. This allows you to plug different PSoC modules into the board based upon the desired features of both 8-bit and 32-bit PSoC devices.

1.4.1 Default Switch and Jumper Settings

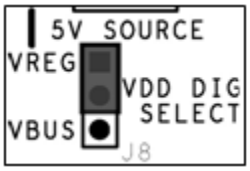
Jumpers on the CY8CKIT-001 PSoC Development Main Board have a default setting for 3.3V operation. For Default configuration, each of the jumpers must be set according to these instructions.

Note: All CY8C29x66 family processor module example projects are configured for 5V. Do configure the board to 5V, before creating the example projects.

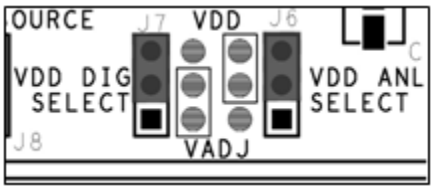
SW3 - VDD Select. Default Position: 3.3V (down position)



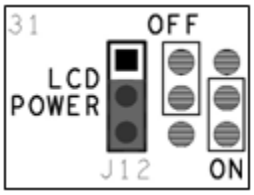
J8 - 5V Source. Default Position: VREG (upper two pins)



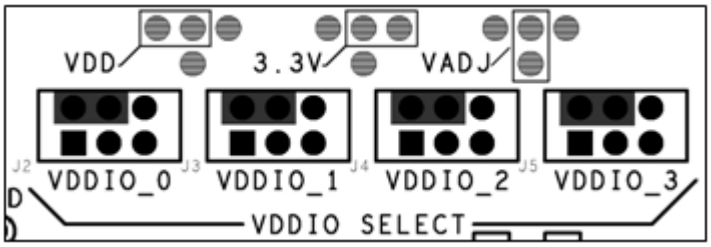
J7, J6 - VDD Digital, VDD Analog. Default Position: VDD (upper two pins, both headers)



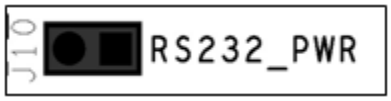
J12 - LCD Power. Default Position: ON (lower two pins)



J2-J5 - VDDIO Power Select. Default Position: VDD (upper left two pins)



J10 - RS232 Power (Serial Communications). Default Position: Installed



J14 - Radio Power. Default Position: Installed

J11 - Variable Resistor Power. Default Position: Installed

1.5 Conventions

These conventions are used throughout this guide.

Table 1-1. Documentation Conventions

Convention	Usage
`Courier New Size 12`	Displays file locations and source code: `C:\ …cd\icc\.`
Italics	Displays file names and reference documentation: *sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
Bold → With → Arrows	Represents menu paths, user entered text: **File → New Project → Clone**
Bold	Displays commands and selections, and icon names in procedures: Click the **Debugger** icon, and then click **Next**.
Note:	Displays functionality unique to PSoC Designer, PSoC Creator, or the PSoC device.
WARNING:	Displays cautions that are important to the subject.

1.6 Document Revision History

Document Title: CY8CKIT-001 PSoC Development Kit Guide Document Number: 001-48651			
Revision	**Issue Date**	**Origin of Change**	**Description of Change**
**	6/16/09	AESA	New Guide

2. Loading My First PSoC Project

The CY8CKIT-001 PSoC Development Kit supports projects across the PSoC, PSoC3, and PSoC5 architectures. This section walks you through the high level design process for opening, building, programming, and running your first PSoC projects using this kit.

Before beginning, follow each of these steps to make certain that your software and hardware environments are properly configured and ready for these projects:

1. Install PSoC Designer using the steps listed in Installing PSoC Development Software on page 4.
2. Install PSoC Creator using the steps listed in Installing PSoC3 Development Software on page 4.
3. Close any open PSoC Creator or PSoC Designer applications and projects.
4. Configure the PSoC Development Board (jumper settings and switches) in its default configuration, as described in Default Switch and Jumper Settings on page 6.
5. Use the PSoC CY8C29 Family Processor Module for the PSoC version of your first PSoC project (My First PSoC Project on page 10).
6. Use the PSoC CY8C38 Family Processor Module for the PSoC3 version of your first PSoC project (My First PSoC3 Project on page 12).
7. Configure the PSoC Development Board prototyping board and I/O:

- CY8C29 Family Processor Module
  - Connect P0[7] to LED1
  - Connect P1[7] to LED2
- CY8C38 Family Processor Module
  - Connect P1[6] to LED1
  - Connect P1[7] to LED2

8. Connect the MiniProg3 into your PC using the supplied USB cable. When you connect the MiniProg3, Microsoft Windows® may indicate that it has found new hardware. All required drivers were installed as part of the PSoC Programmer installation process; however, if Windows opens the driver installation dialog boxes, accept the defaults and allow Windows to automatically find the appropriate driver.
9. For a PSoC project, use the ISSP header on the PSoC CY8C29 Family Processor Module and connect the MiniProg3 ISSP port.
10. For a PSoC3 project, use the JTAG ribbon cable. Connect the ribbon cable to the MiniProg3 and the PSoC CY8C38 Family Processor Module into the header labeled PROG on the processor module.

    Note The MiniProg3 should not be “hot plugged” into processor modules that are attached to the PSoC Development Board. In other words, do not plug the ribbon cable of the MiniProg3 into the processor module while code is actively running on the module. Doing so may cause the PSoC device to unintentionally reset. Power down the PSoC Development Board and module by unplugging the power supply from the Development Board before attaching the MiniProg3 device to the module board. Once the ribbon cable is attached to the module board, power can then be

applied to the system by plugging in the power supply to the PSoC Development Board. This will avoid any undesirable PSoC device resets.

11. Power the PSoC Development Board using the 12V AC Power Adapter

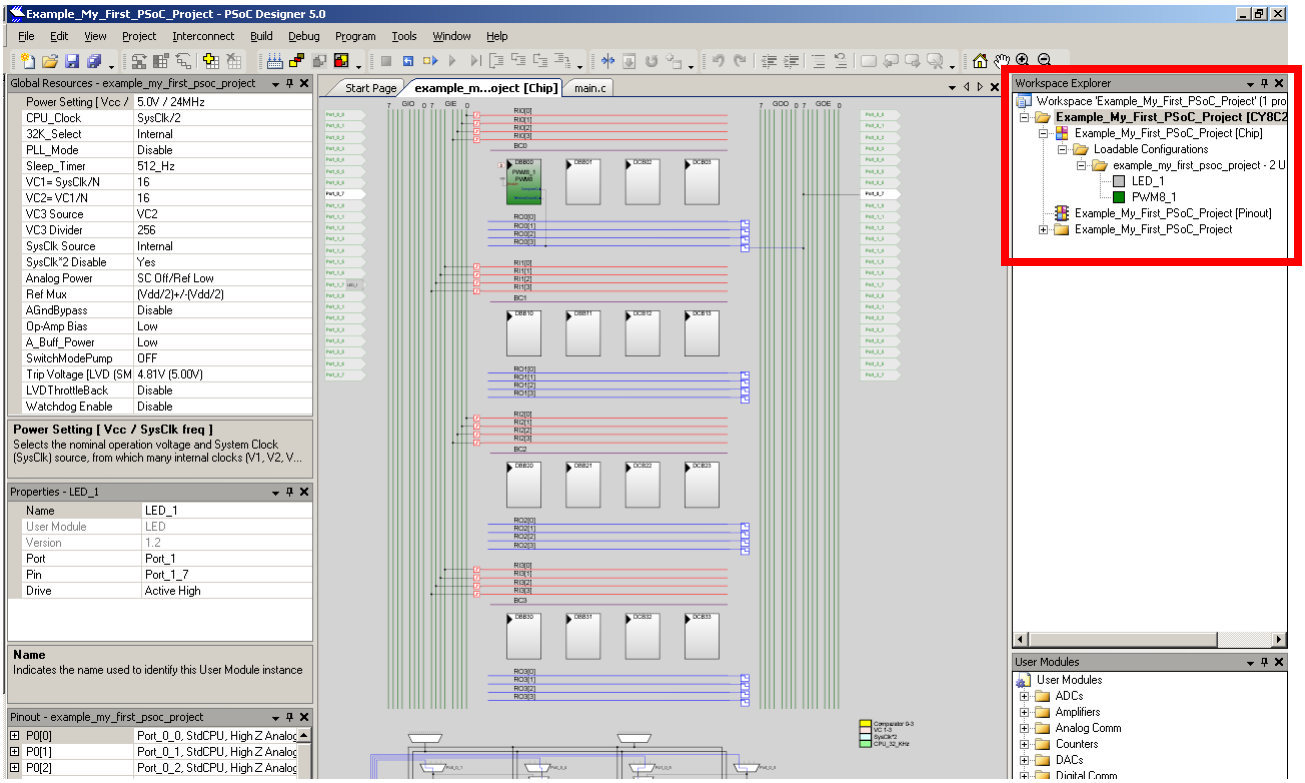
2.1 My First PSoC Project

This is a simple PSoC project using a PWM peripheral programmed from inside the PSoC device to control the blinking rates of two different LED outputs. For this project, be sure you have the PSoC CY8C29 Family Processor Module inserted into the PSoC Development Board and the appropriate software installed. This section walks you through the high level steps of opening, building, and programming a project.

2.1.1 Loading My First PSoC Project

1. Open PSoC Designer.
2. In the **Start Page**, navigate to **File** → **Open Project/WorkSpace**
3. Navigate to the PSoC project directory `C:\Cypress\CY8CKIT-001\CY8C29 Projects\`
4. Open the folder `Example_My_First_PSoC_Project`.
5. Double click **Example_My_First_PSoC_Project.app**.
6. The project opens in the Chip Editor view. All the project files are in the **Workspace Explorer**.

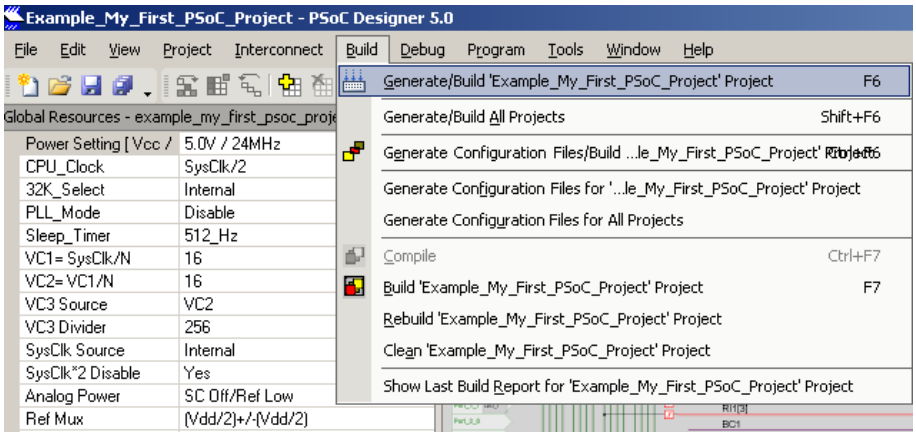
Figure 2-1. Chip Editor View



2.1.2 Building My First PSoC Project

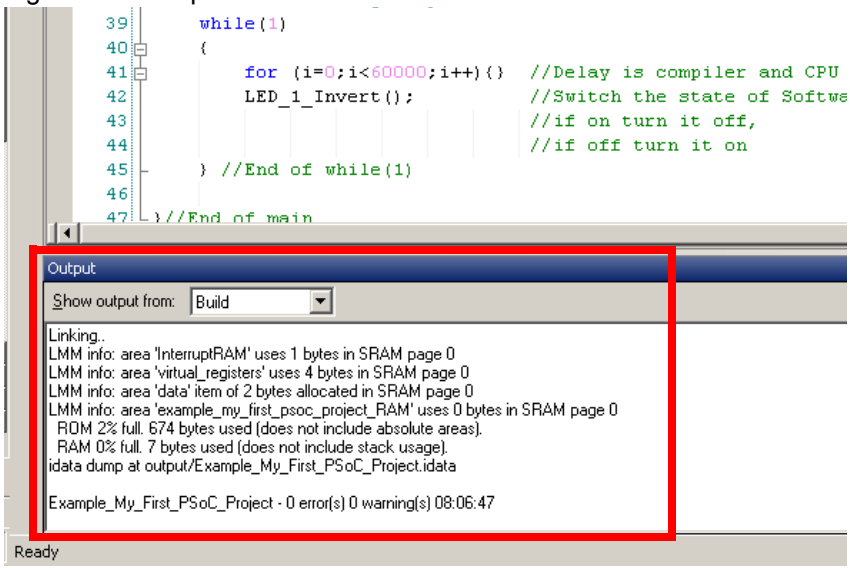
1. Select **Build** → **Build 'Example_My_First_PSoC_Project' Project**.

Figure 2-2. Build Project



2. PSoC Designer builds the project and displays comments in the **Output** window. When you see the message that the project built with 0 errors and 0 warnings you are ready to program the device.

Figure 2-3. Output Window



2.1.3 Programming My First PSoC Project

Note The CY8C29 Family Processor Module is designed to accommodate the use of the CY3215-DK In-Circuit Emulator (ICE Cube). When using the ICE Cube debugger, make certain that PSoC Designer is configured so that the ICE Cube DOES NOT provide power to the processor module. Within the **PSoC Designer** application, select **Project** > **Settings** and select **Debugger** from the tree list. Make sure **External only** is selected under the **Pod Power Source** section.

1. Open PSoC Programmer from within PSoC Designer by selecting **Program** → **PSoC Programmer**.
2. In PSoC Programmer, make sure that **MiniProg3** is selected in the **Port Selection** box.
3. In PSoC Programmer, set **Programming Mode** to **Reset**.

4. In PSoC Programmer, set **Verification** to **On** so that the software verifies that the downloaded program's checksum matches the actual checksum of the flash memory after programming. This is a precautionary check to verify that there is no data corruption during programming.
5. In PSoC Programmer, set **AutoDetection** to **On** to enable the software to automatically detect and configure for the target device family and device. **Note** If PSoC Programmer is properly configured, AutoDetection reports a device family of 29x66 and device of CY8C29466.
6. Set the protocol to ISSP.
7. With these settings configured, click **Program** to program your PSoC device.
8. Wait until programming is complete before continuing.

2.1.4 Running My First PSoC Project

1. Now that the PSoC device is programmed, reset the PSoC Development Board by pressing and releasing the reset switch (SW4).
2. Verify that LED1 and LED2 are blinking based on the project's use of the PWMs. LED1 blinks approximately once every second and LED2 blinks about three times a second.
3. For more details regarding this project, see the detailed step-by-step project instructions in My First PSoC Project on page 17.

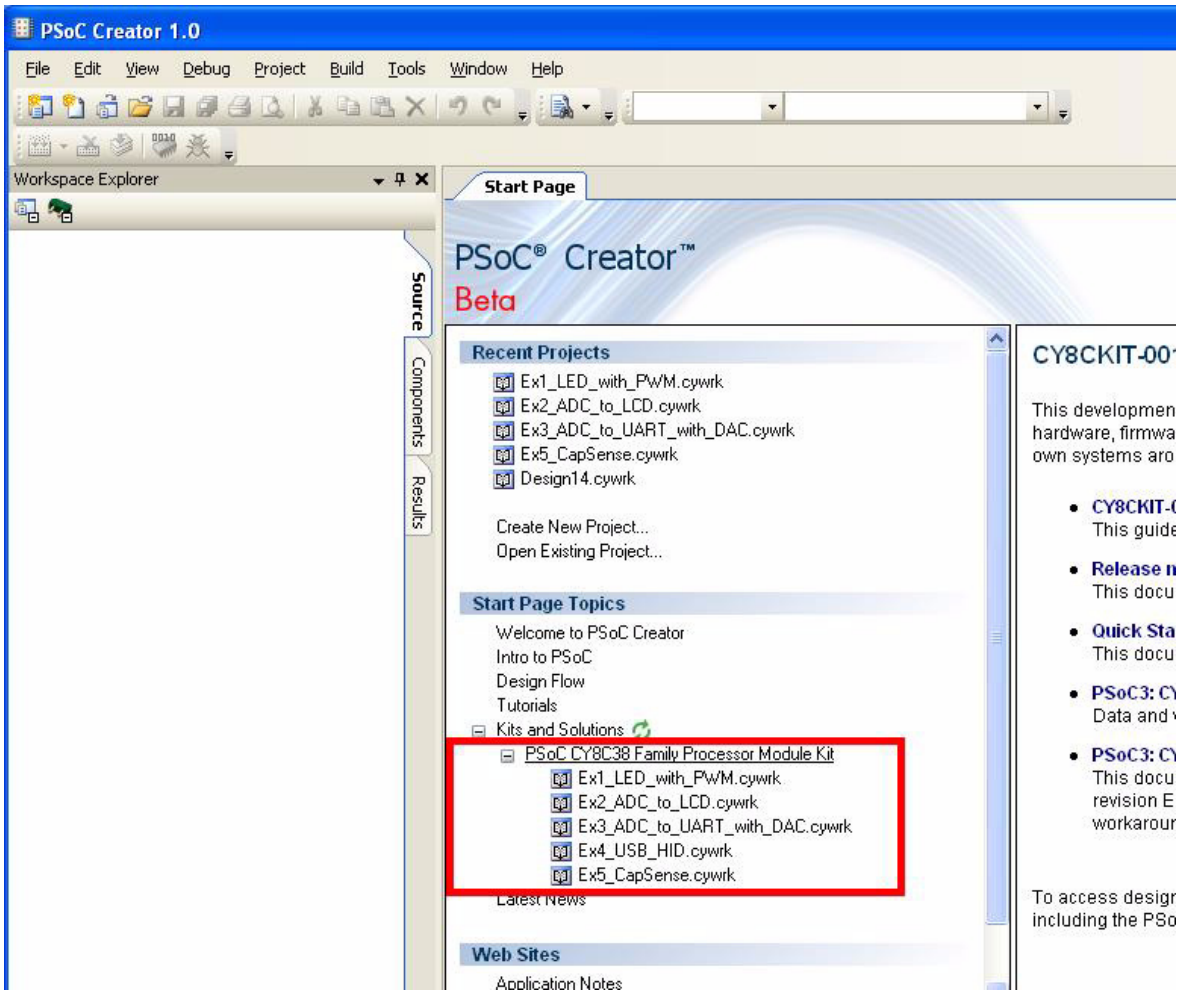
2.2 My First PSoC3 Project

This is a PSoC3 project using a PWM peripheral programmed from inside the PSoC device to control the blinking rates of two different LED outputs. For this project, insert the PSoC CY8C38 Family Processor Module in the PSoC Development Board and install the appropriate software. This section shows you the high level steps of opening, building, and programming a project.

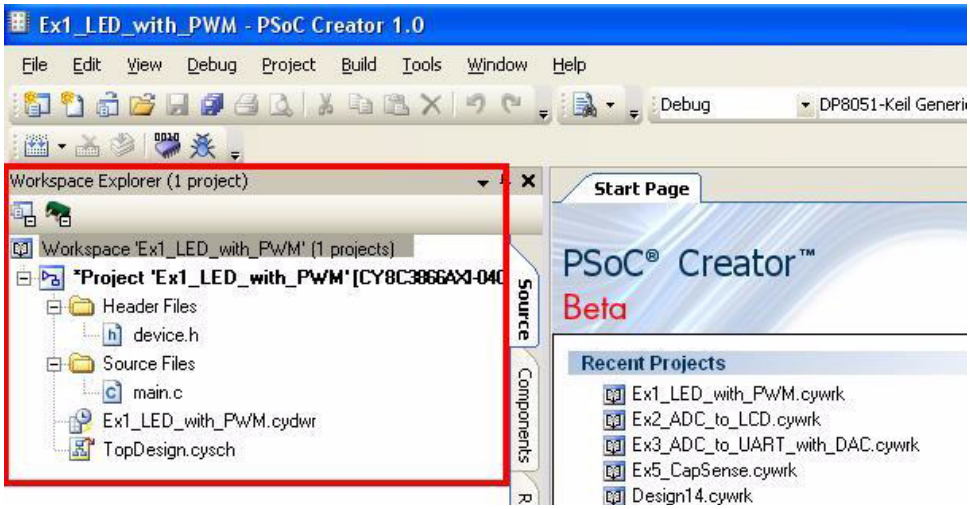
2.2.1 Loading My First PSoC3 Project

1. Open PSoC Creator.
2. In the **Start Page**, under **Start Page Topics** expand **Kits and Solutions**.
3. Under **Kits and Solutions**, expand **PSoC Development Kit**.

4. Click **Ex1_LED_with_PWM.cywrk** to open the project.



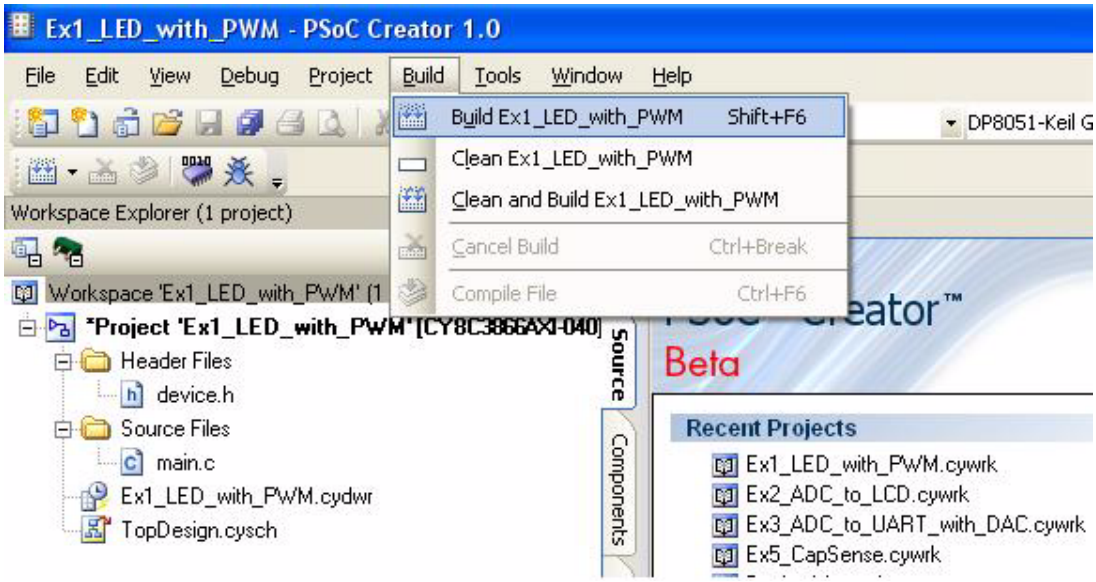
5. Select the directory in which to store the project.
6. After the project opens, you can see the project files in **Workspace Explorer**.



2.2.2 Building My First PSoC3 Project

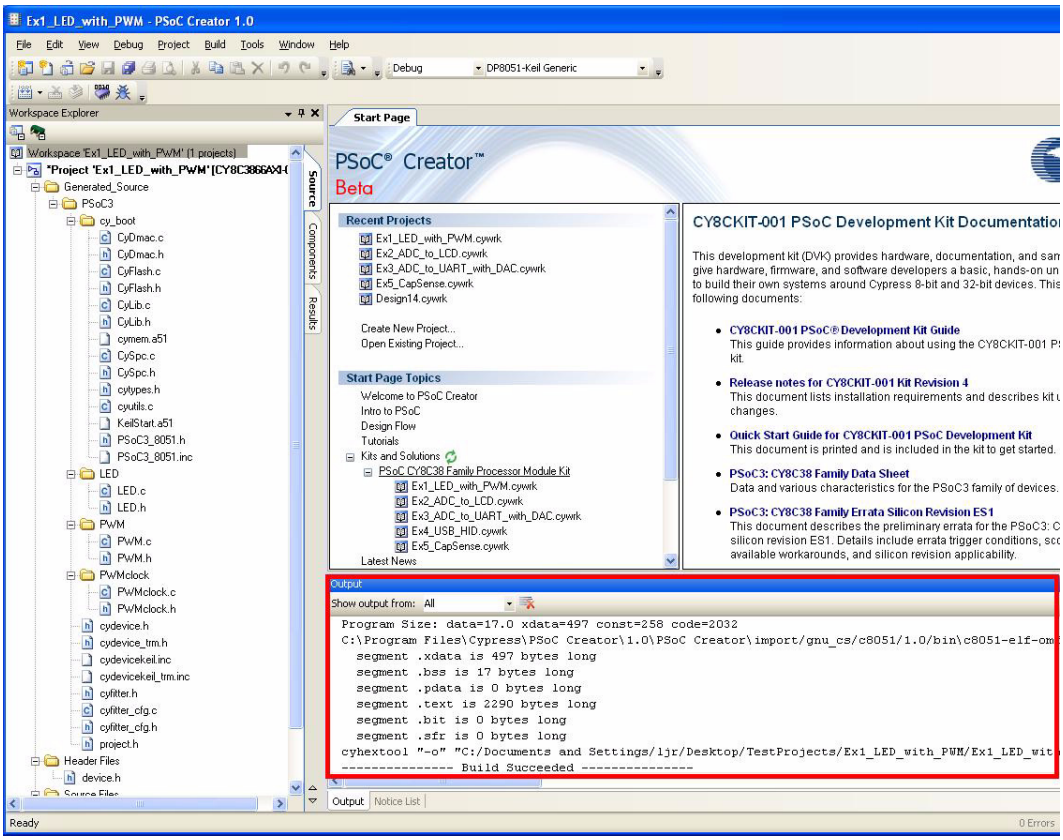
1. Select **Build** → **Build Ex1_LED_with_PWM**.

Figure 2-4. Build Window



2. PSoC Creator builds the project and displays the comments in the **Output** window. When you see the message “Build Succeeded” you are ready to program the device.

Figure 2-5. Output Window



2.2.3 Programming a Device

1. If this is your first time running PSoC Creator, follow these steps to configure the MiniProg3 device for these PSoC Development Kit projects. If these configurations are set skip to the next step below and begin programming.

   Note VTARG of the MiniProg3 is wired exclusively to VDDIO1 of the chip on the PSoC CY8C38 Family Processor Module. Because of this the user cannot perform powercycle mode programming.

   - From the **Tools** menu in PSoC Creator, click **Options**. The **Options** window opens.
   - In the **Options** window, select **Debugging** → **MiniProg3** from the tree list.
     - Set **Applied Voltage** to **3.3V**
     - Set **Transfer Mode** to **SWD**
     - Set **Active Port** to **10 Pin**
     - Set **Acquire Mode** to **Reset**
     - Set **Clock Speed** to **3.2 MHz**
     - Click **OK**.
   - From the **Debug** menu, select **Select Debug Target**. The **Select Debug Target** dialog box opens.
   - Expand the tree under **MiniProg3** and click **Port Acquire**.
   - Select the appropriate device and click **Connect**.
   - Click **Close**.
2. In PSoC Creator, from the **Debug** menu, click **Program**.
3. The PSoC Creator Status Bar indicates that the device is programming.
4. Wait until programming is complete before continuing.

2.2.4 Running My First PSoC3 Project

1. Reset the PSoC Development Board by pressing and releasing the reset switch (SW4).
2. Verify that LED1 and LED2 are blinking based on the project's use of the PWMs. LED1 blinks approximately once every second and LED2 blinks about three times a second.
3. For more details regarding this project, review the detailed step-by-step project instructions in My First PSoC Project on page 48.

3. Sample Projects

This chapter shows you how to create the sample projects included with this kit.

Read these precautions before you create example projects:

- All CY8C29x66 family processor module example projects are configured for 5V.
- All CY8C38 family processor module example projects are configured for 3.3V.
- Do close any open project in PSoC Creator before loading or creating an example project
- When working with example projects, use 12V wall wart power supply
- Remove power before changing board jumpers for each example project. Reapply power after you place jumpers in the bread board.
- When you complete each project make certain to save the project

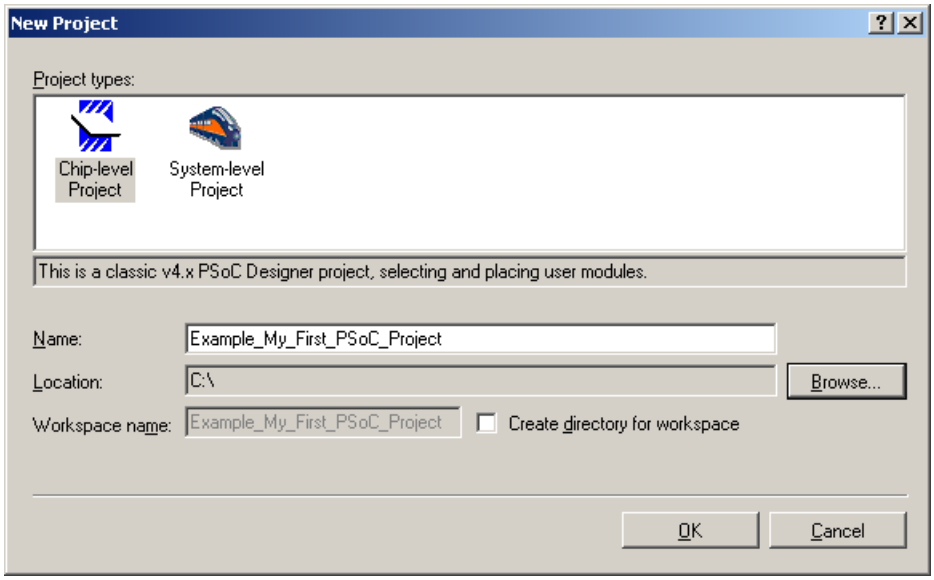
3.1 CY8C29x66 Family Processor Module Example Projects

3.1.1 My First PSoC Project

3.1.1.1 Creating My First PSoC Project

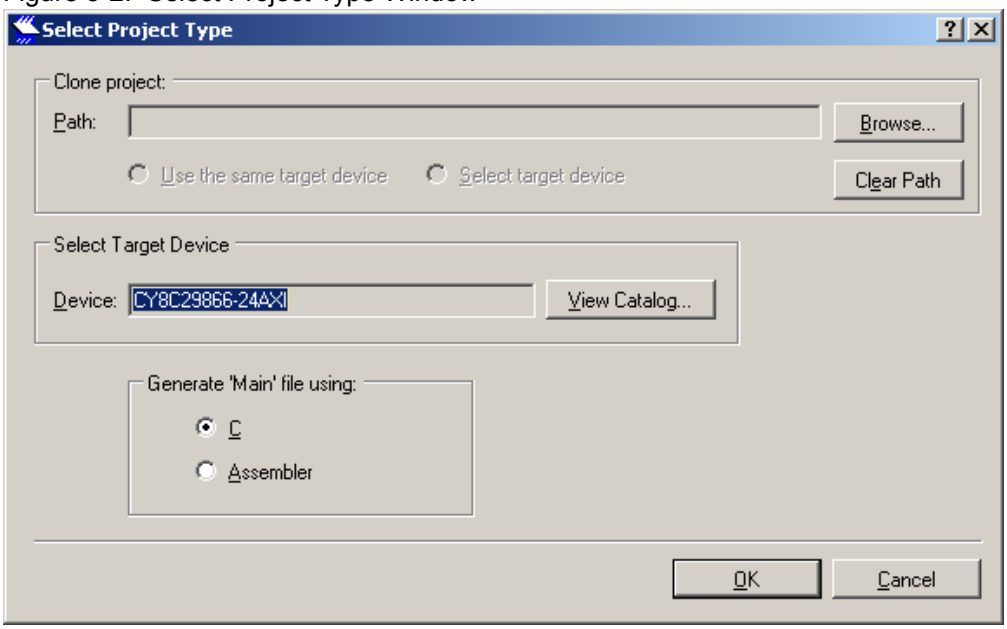
1. Open PSoC Designer 5.0
2. To create a new project, click **File** → **New Project**. The **New Project** window opens.
3. In the **New Project window**, select the **Chip-Level Project**. **Name** the project **Example_My_First_PSoC_Project**.
4. In **Location**, click **Browse** and navigate to the appropriate directory.

Figure 3-1. New Project Window



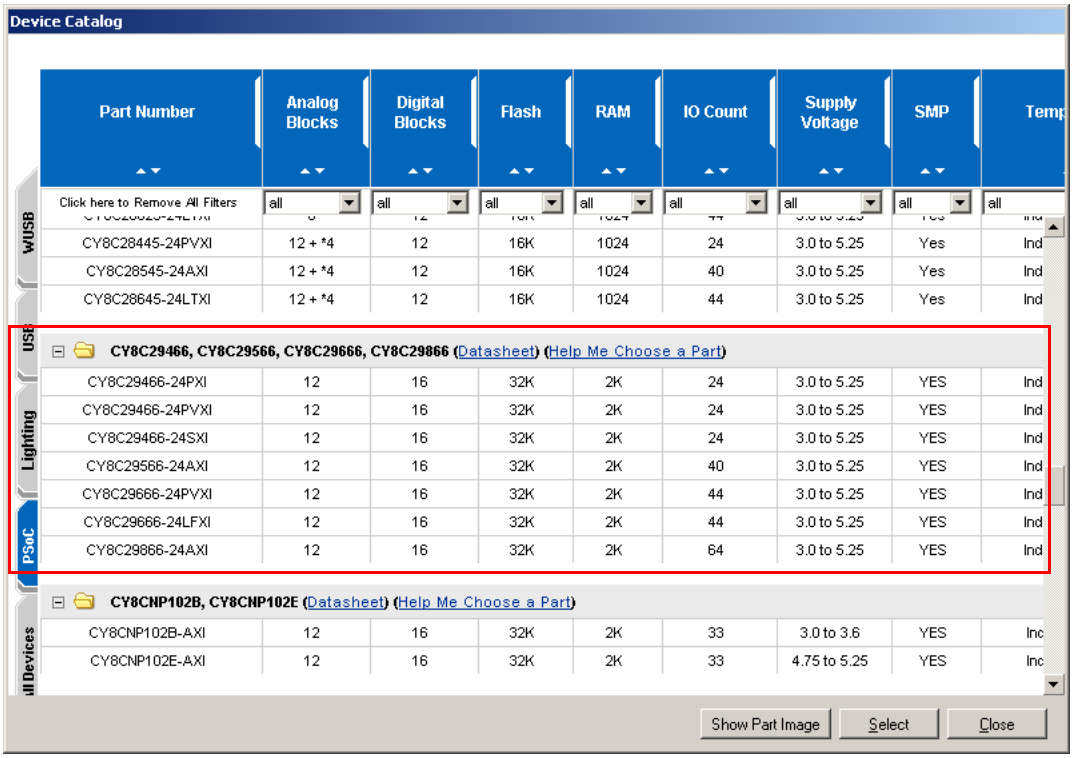
5. Click **OK**. The **Select Project Type** window opens.

Figure 3-2. Select Project Type Window



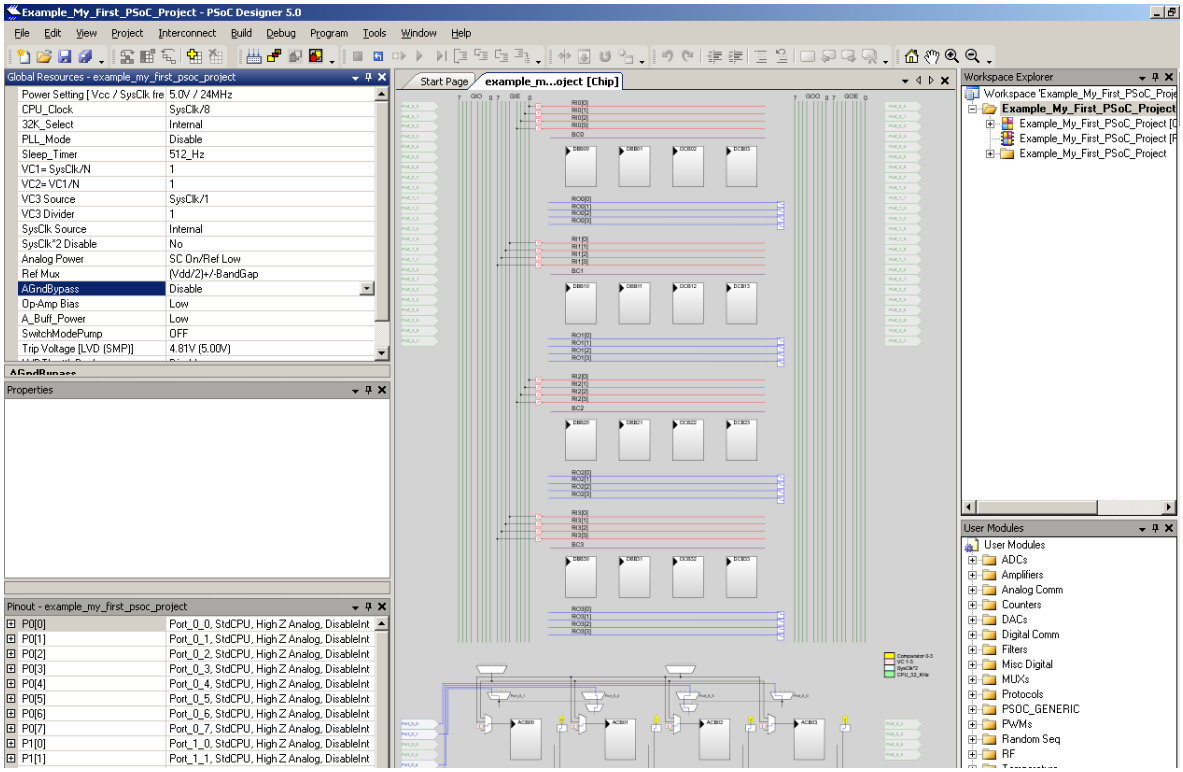
6. In this window under **Select Target Device**, click **View Catalog**.
7. The **Device Catalog** window opens. Click on the **PSoC** tab, and scroll down to the **CY8C29466, CY8C29566,…** section.
8. For this project click any device in this section and then click **Select**.

Figure 3-3. Device Catalog Window



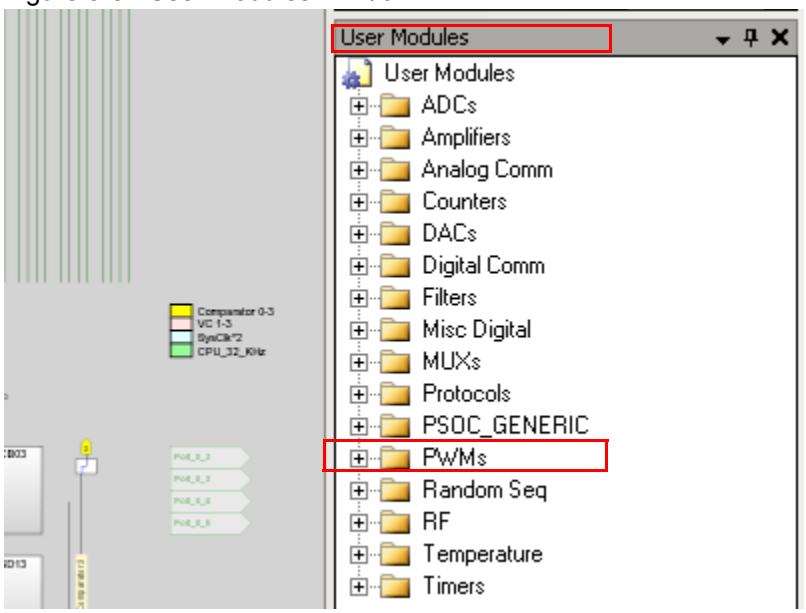
9. Under **Generate 'Main' File Using:**, for this project select **C**, then click **OK**.
10. By default, the project opens to chip view

Figure 3-4. Default View.



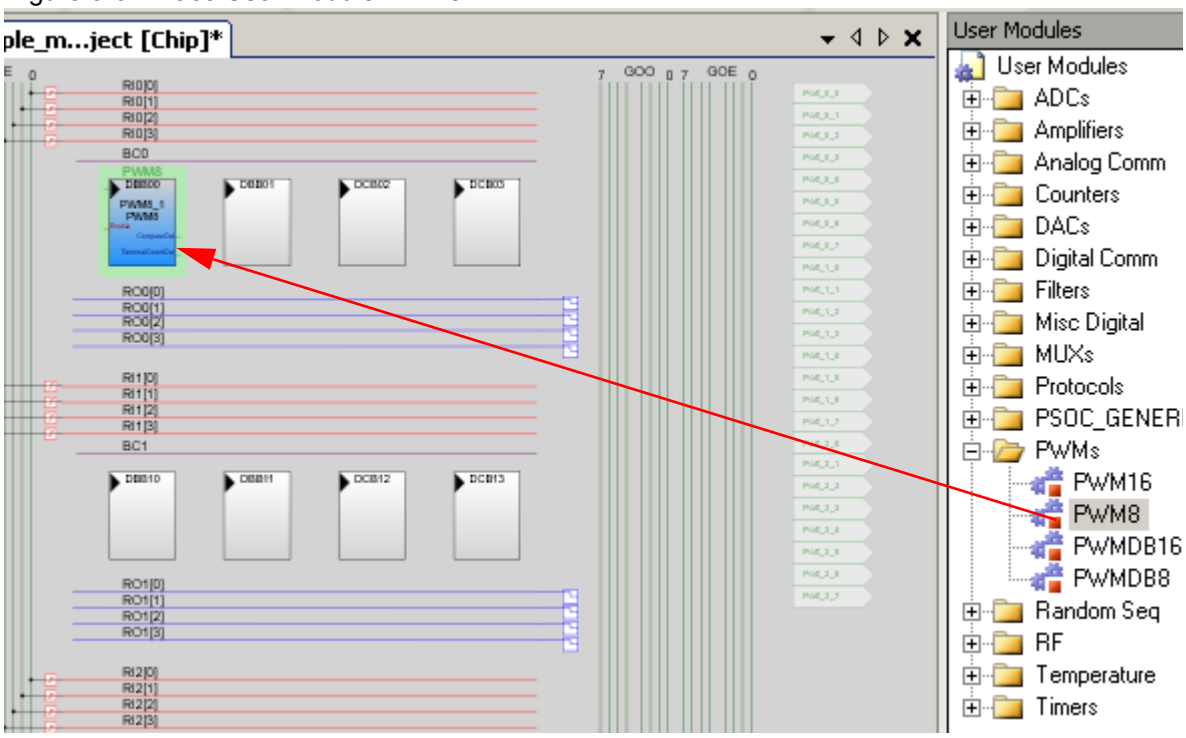
11. In the **User Modules** window, expand the **PWMs** folder.

Figure 3-5. User Modules Window



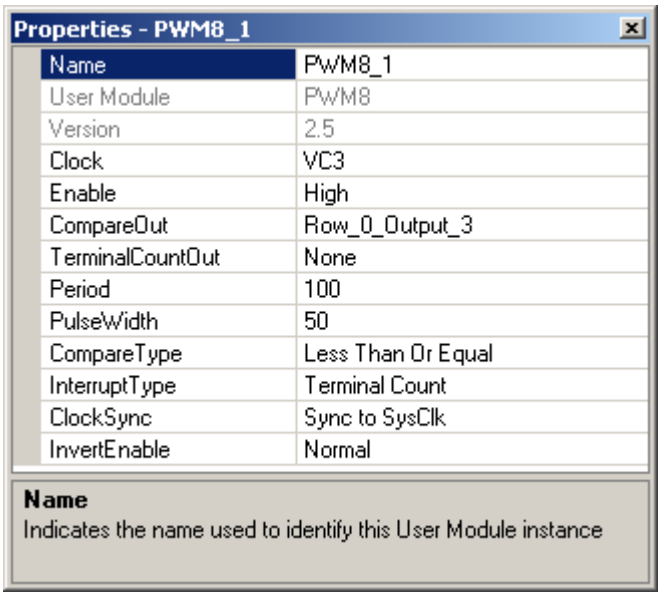
12. In this folder right click on a **PWM8** and select place. The User Module (UM) is placed in the first available digital block.

Figure 3-6. Place User Module PWM8

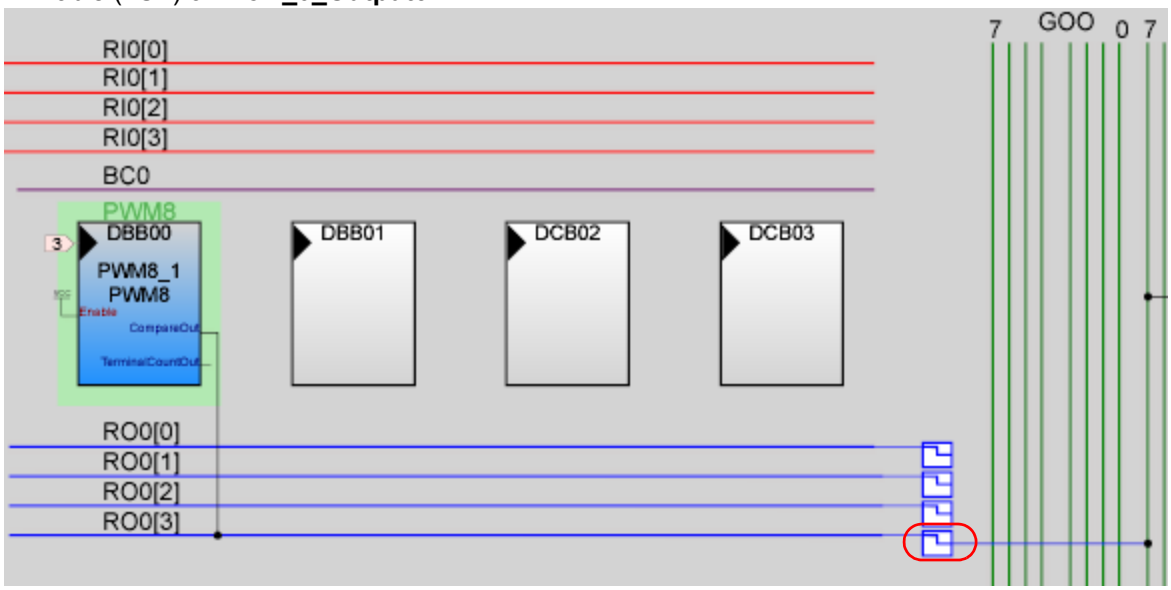


13. Double click the placed PWM8_1 UM; the **Properties** window opens on the left side of the screen. Configure the PWM with the settings as in the following figure. If the **Properties** window does not appear, click **View** → **Properties** Window.

Figure 3-7. Properties Window

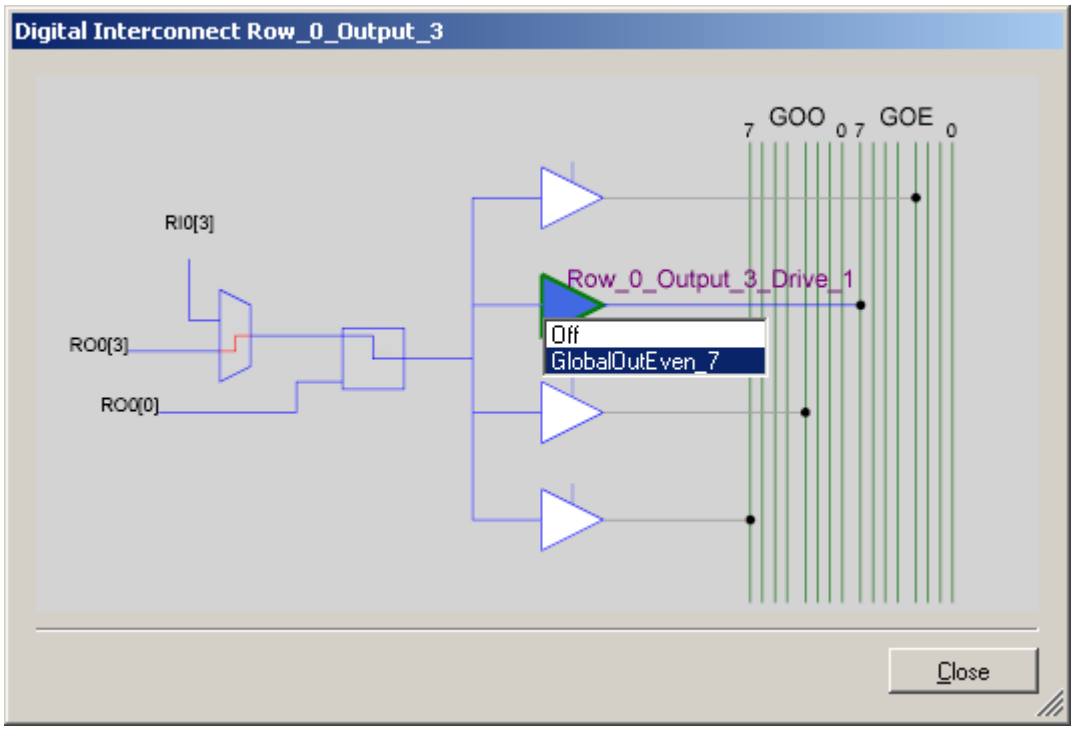


14. Next, route the PWM **CompareOut** signal to P0_7. The first step is to configure the Look Up Table (LUT) on **Row_0_Output3**.



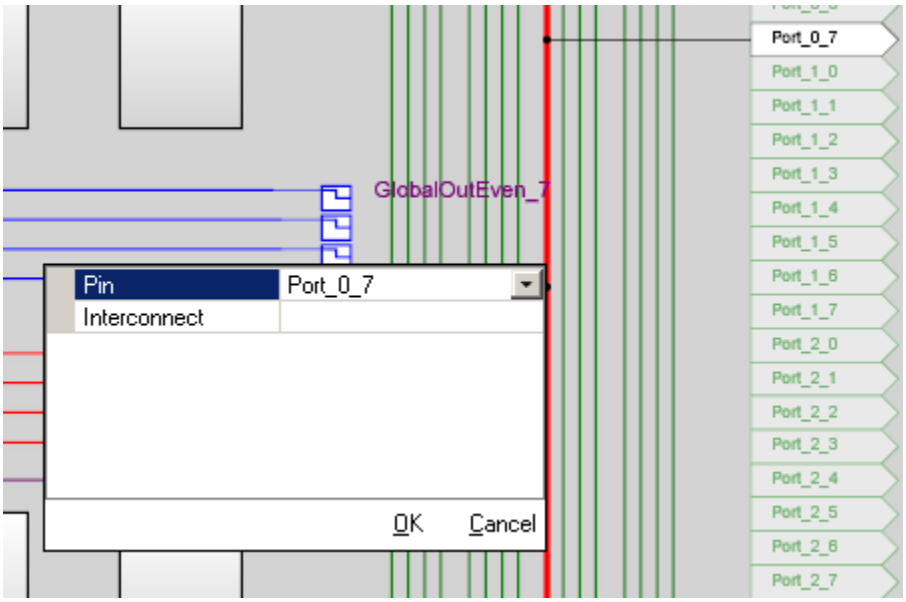
15. Double click the **LUT**, the **Digital Interconnect** window opens.
16. In this window enable **Row_0_Output_3_Drive_1** to connect to **GlobalOutEven_7**.

Figure 3-8. Digital Interconnect Window



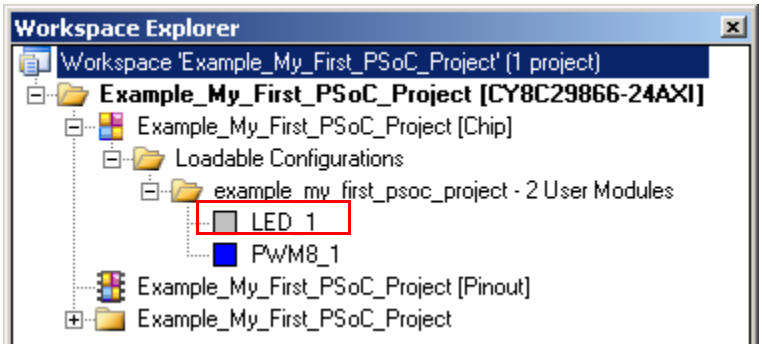
17. Click **Close**.

18. Click on **GlobalOutEven_7**. A window appears, in this window configure **PIN** for **Port_0_7**.



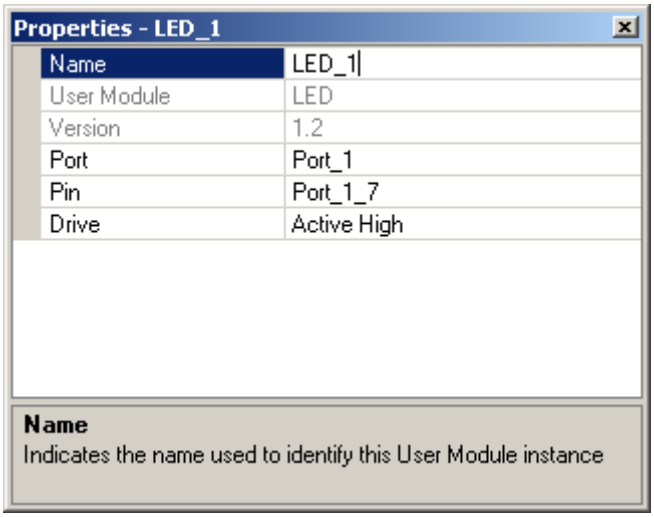
19. Click **OK** to continue.
20. In the **User Modules** window expand the **Misc Digital** folder. In this folder right click the **LED** and select place, this adds the UM to the project. This UM does not use digital or analog blocks. It appears in the **Workspace Explorer** → **Example_My_First_PSoC_Project[CY8C29x66]** → **Example_My_First_PSoC_Project[Chip]** → **Loadable Configurations** → **example_my_first_psoc_project - 2 User Modules**.

Figure 3-9. Workspace Explorer

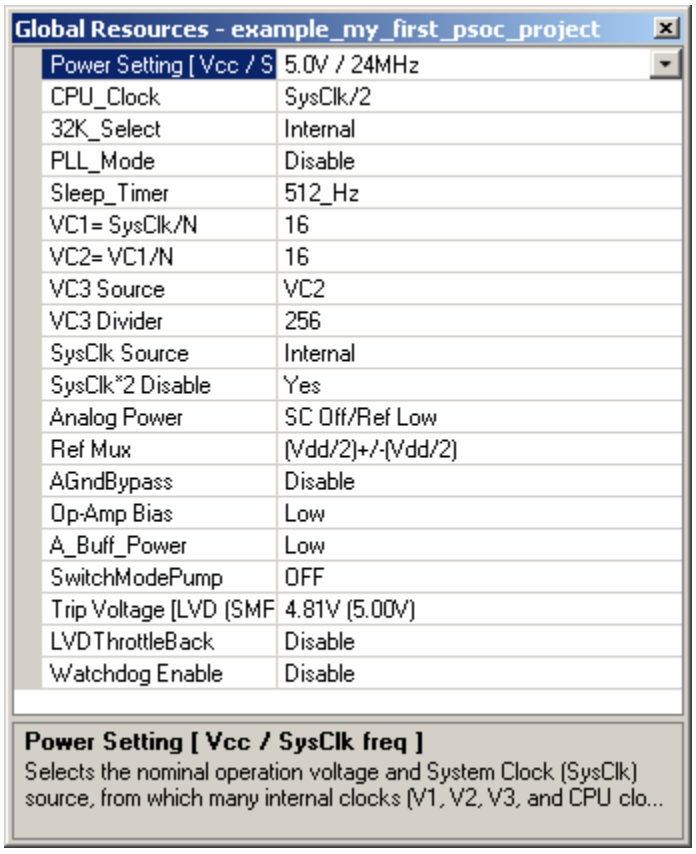


21. Double click the **LED_1** UM and navigate to the **Properties** window. Configure the LED for **P_1_7.**

Figure 3-10. .Properties Window

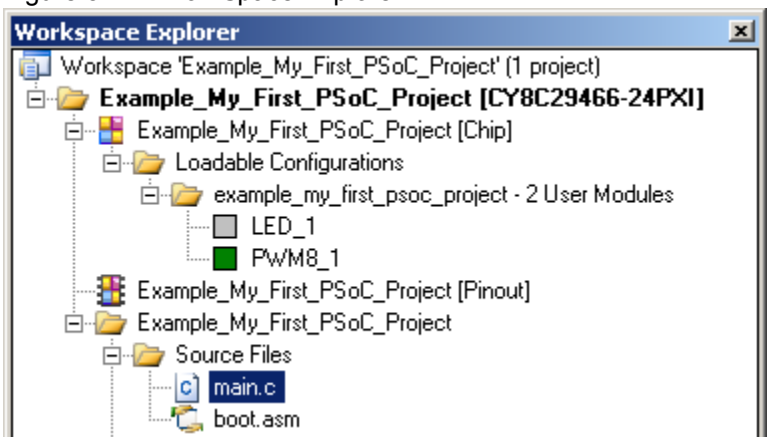


22. Configure the **Global Resources** window to match the following figure.



23. Open your *main.c* file and copy the example code located in section 3.1.1.2 on page 25 to the existing *main.c*. **main.c** is in the Workspace Explorer.

Figure 3-11. WorkSpace Explorer



24. Save the project.
25. Build the project. **Build → Generate/Build 'Example_My_First_PSoC_Project' Project**.
26. Disconnect power to the board.
27. Configure the DVK bread board SW3 to 5V.
28. Configure the DVK bread board using the included jumper wires:
    - P0_7 to LED1
    - P1_7 to LED2
29. Reapply power to the board.
30. Use PSoC Designer as described in Programming My First PSoC Project on page 11 to program the device.
31. Reset the DVK, and observe the blinking LEDs.
32. Save and close the project.

3.1.1.2 main.c

```
/*****************************************************************************
* File Name: main.c
*
* Description:
*  This file provides source code for My First PSoC Project example.
*  The firmware blinks one LED at about 3.6Hz with a PWM, and blinks
*  another LED with a software timing loop.
*
*****************************************************************************/

/*****************************************************************************
*   PWM Settings:
*
*     Input Clock      = VC3        //VC3 = 24MHz/16/16/256 =366.2Hz
*     Enable           = High
*     CompareOut       = ROW_0_Output_3
*     TerminalCountOut = None
*     Period           = 100    Output period = (Period+1)*(1/Input
*                        Clock) = 101/366.2 = .275sec or 3.6Hz
*     PulseWidth       = 50
*     CompareType      = Less Than Or Equal
*     InterruptType    = Terminal Count
*     ClockSync        = Sync to SysClk
*     InvertEnable     = Normal
*
*****************************************************************************/

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

unsigned int i;          // Variable used for delay

void main()
{
    PWM8_1_Start();      // Turn on the PWM to blink LED on P0_7
    LED_1_Start();       // Enable Software controlled LED on P1_7


   // The following loop controls the software LED connected to P1.7
      while(1)
      {
         for (i=0;i<60000;i++){}  //Delay is compiler and CPU dependant
                  LED_1_Invert();//Switch the state of Software LED,
         //if on turn it off,
         //if off turn it on
      } //End of while(1)

}//End of main
```

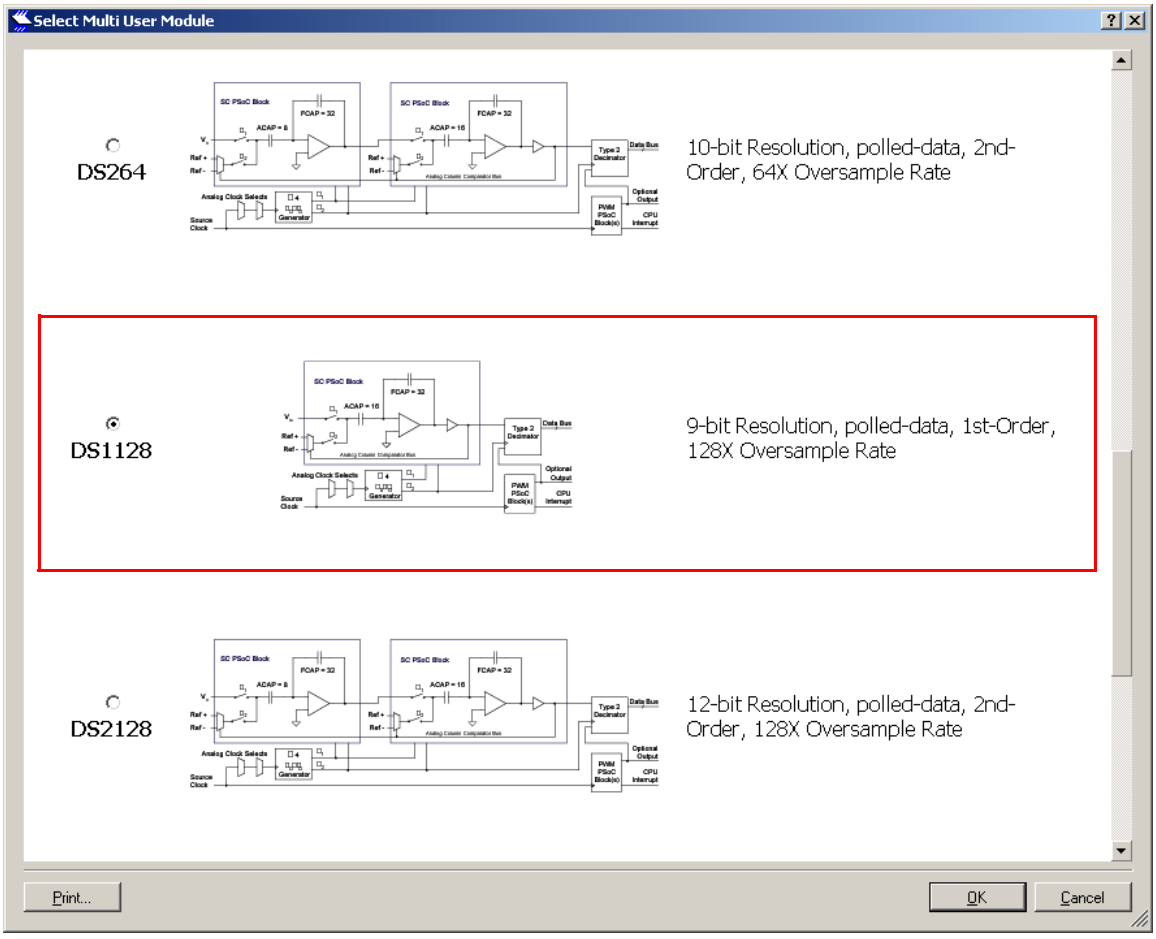
3.1.2 ADC to LCD Project

This project demonstrates a 9-bit Delta Sigma ADC by measuring the voltage of the potentiometer and displaying the result on the LCD. Connect the voltage potentiometer (VR) to the ADC input P0_1. The program reads the 9-bit ADC result and prints it to the LCD.

3.1.2.1 Creating ADC to LCD Project

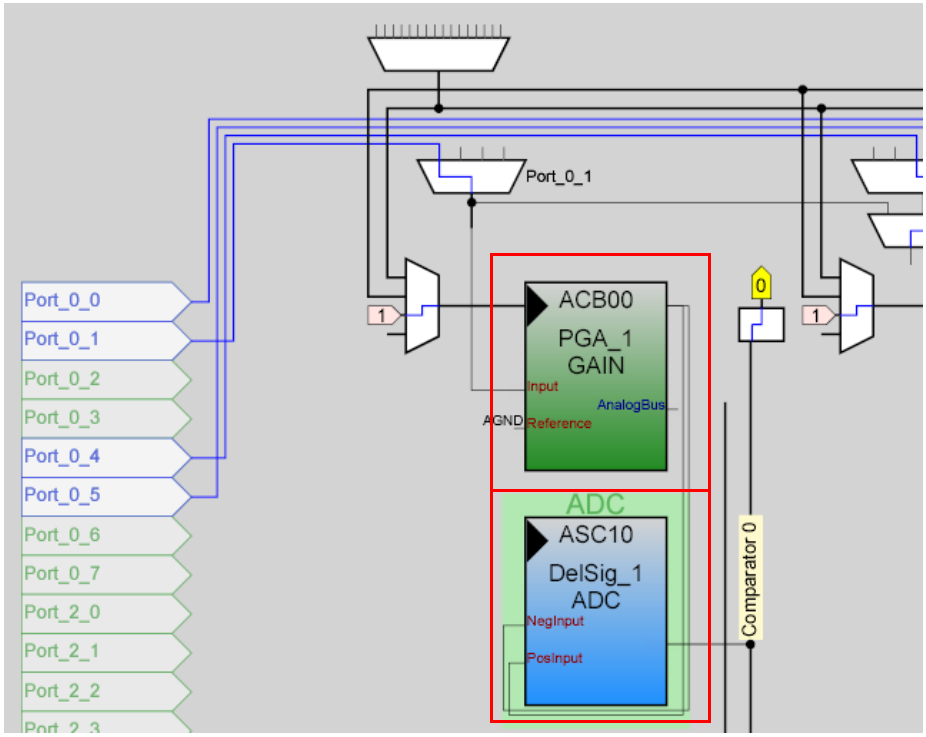
1. Follow steps 1 to 10 in section 3.1.1.1 on page 17, change the **Name** of the project to **Example_ADC_to_LCD**.
2. In the **User Modules** window expand the **ADCs** folder, and then expand the **DelSig** subfolder. Right click **DS1128** and choose **Place**. A window opens with multiple options for the DelSig UM. In this case the **DS1128** configuration is used. Scroll down in the window to verify that this is the case.

Figure 3-12. Select Multi User Module Window

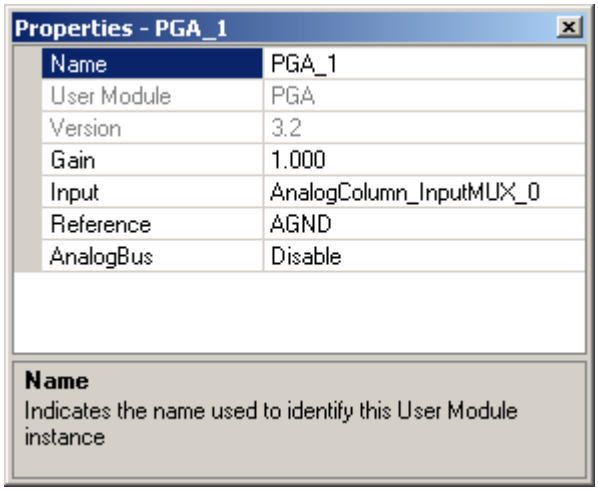


3. Click **OK**.
4. Verify that the **DelSig_1** UM is placed in **ASC10**.

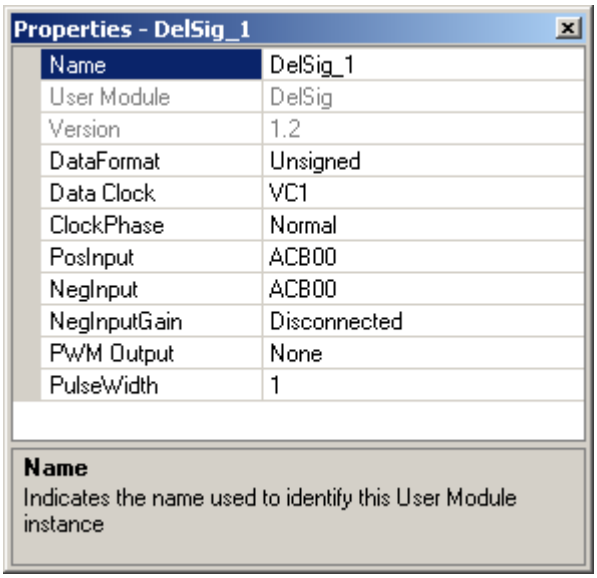
5. In the **User Modules** window expand the **Amplifiers** window. Right click **PGA**, select place. Ensure that the **PGA** is placed in **ACB00**.



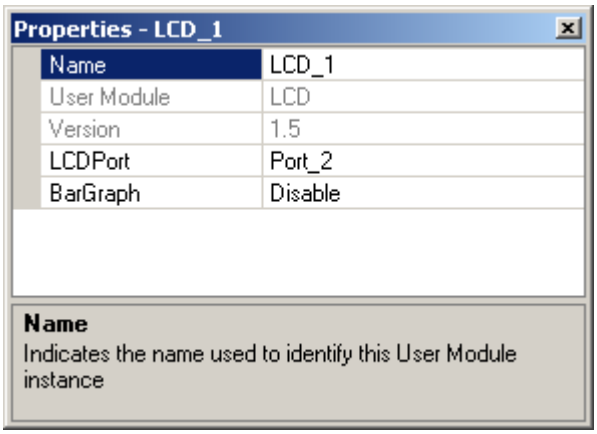
6. In the **User Modules** window expand **Misc Digital**, Right click **LCD** and click place.
7. Double click **PGA_1** and configure the properties to match this figure.



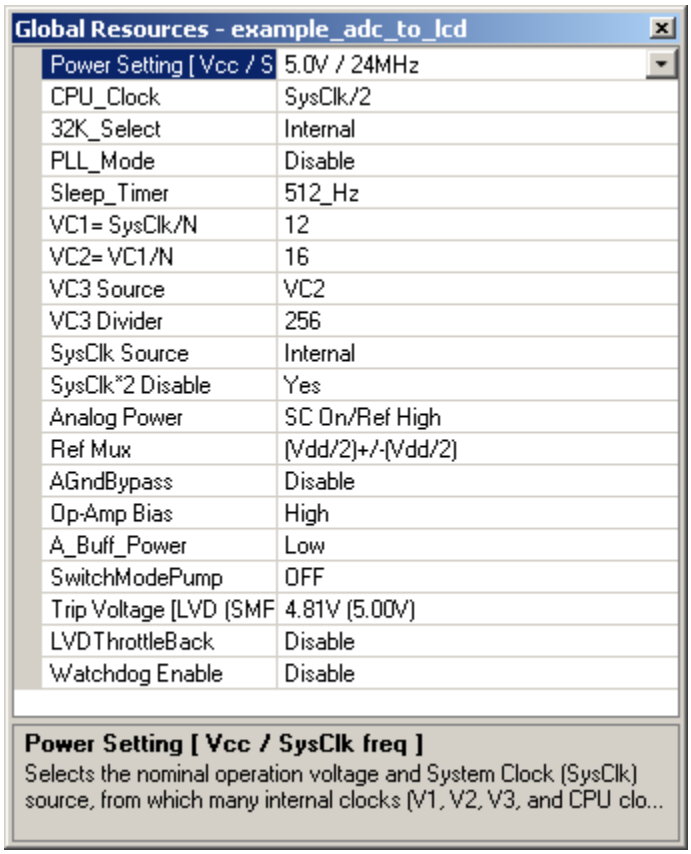
8. Double click **DelSig_1** and configure the properties to match this figure.



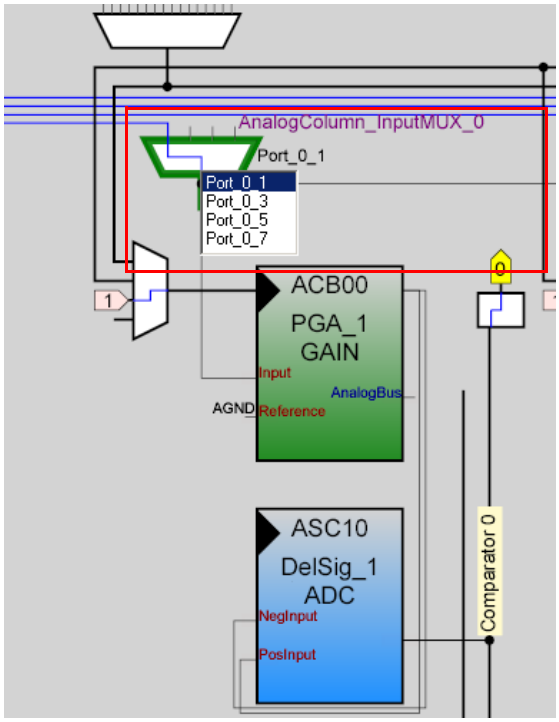
9. Double click **LCD_1** and configure the properties to match this figure.



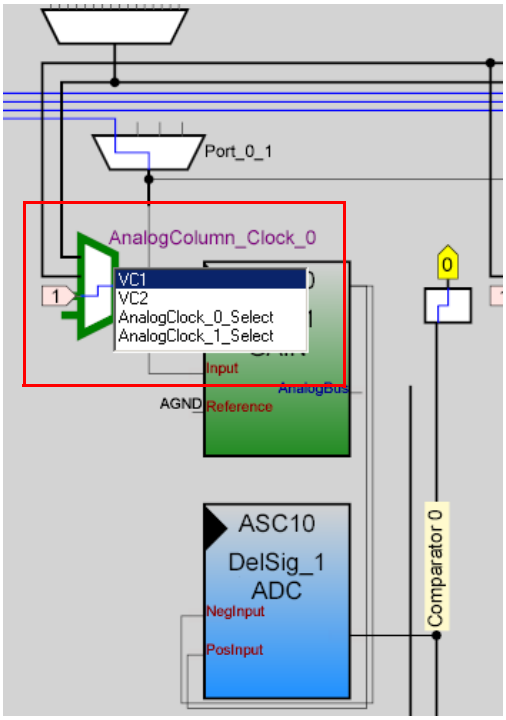
10. Configure the **Global Resources** to match the following figure.



11. Ensure that **AnalogColumn_InputMUX_0** is connected to **Port_0_1**. If it is not configured for this port, double click the **MUX** and choose **Port_0_1**.



12. Ensure that **AnalogColumn_Clock_0**, is connected to **VC1**. If it is not, double click the **MUX** and chose **VC1**.



13. Open your *main.c* file and copy the example code located in section 3.1.2.2 on page 31 to the existing *main.c*.
14. Save the project.
15. Build the project. **Build → Generate/Build 'Example_ADC_to_LCD' Project**.
16. Disconnect power to the board.
17. Configure the DVK bread board SW3 to 5V.
18. Configure the DVK bread board using the included jumper wires:
    - P0_1 to VR
19. Reapply power to the board.
20. Use PSoC Designer as described in Programming My First PSoC Project on page 11 to program the device.
21. After programming the device, press the reset button and vary the pot to see the results on the LCD.
22. Save and close the project.

3.1.2.2 main.c

```
/*****************************************************************************
* File Name: main.c
*
* Description:
*  This file provides source code for the ADC to LCD example project. The
*  firmware takes a voltage output from a potentiometer and displays the raw
*  counts on an LCD.
*
*****************************************************************************

/*****************************************************************************
*   PGA Settings:(The PGA buffers the potentiometer voltage on P0.1 into the *
ADC)
*
*     Gain      = 1
*     Input     = AnalogColumn_InputMUX_0  (P0.1)
*     Reference = AGND
*     AnalogBus = Disable
*****************************************************************************/
/*****************************************************************************
*   LCD Settings:
*     LCDPort   = Port_2
*     BarGraph  = Disable
*****************************************************************************/
/*****************************************************************************
*   DelSig Settings:
*   The ADC can read full range values from 0-5V, if the Ref Mux setting is
*      selected as (Vdd/2)+/- (Vdd/2) and Vdd = 5V. The ADC is configured for a
*   resolution of 9 bits, this is achieved by selecting the appropriate
*   configuration when placing the UM.
*
*     DataFormat   = Unsigned
*     DataClock    = VC1       // VC1 = 24MHz/12 = 2MHz
*     ClockPhase   = Normal
*     PosInput     = ACB00 (PGA_1)
*     NegInput     = ACB00  *Note, this parameter is unused
*     NegInputGain = Disconnected
*     PWM Output   = None
*     PulseWidth   = 1      *Note, this parameter is unused
*****************************************************************************/


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


unsigned int wADCResult; // Holds the integer ADC result

void main()
{
    PGA_1_Start(PGA_1_HIGHPOWER);//Initialize the PGA, PGA used to buffer
                                 //input from the VR on P0.1 to the ADC

   DelSig_1_Start(DelSig_1_HIGHPOWER); //Initialize the ADC
   LCD_1_Start();   //Initialize the LCD
```

```
    LCD_1_Position(0,0);   //Set the LCD to (Row=0,Column=0)

    LCD_1_PrCString("V Count: ");

    DelSig_1_StartAD();    //Start gathering conversions from the ADC

    M8C_EnableGInt;//Enable Global interrupts

    //This loop waits for a valid ADC result, and then displays it on the LCD
    while (1)
    {
while (!(DelSig_1_fIsDataAvailable()));       //Wait for ADC data to be ready
      wADCResult=DelSig_1_wGetDataClearFlag();//Store result from ADC
      LCD_1_Position(0,9);                    //Set LCD to     (Row=0,Column=9)
      LCD_1_PrHexInt(wADCResult);             //Print ADC result on LCD
    }//End of while(1)



}//End of main
```

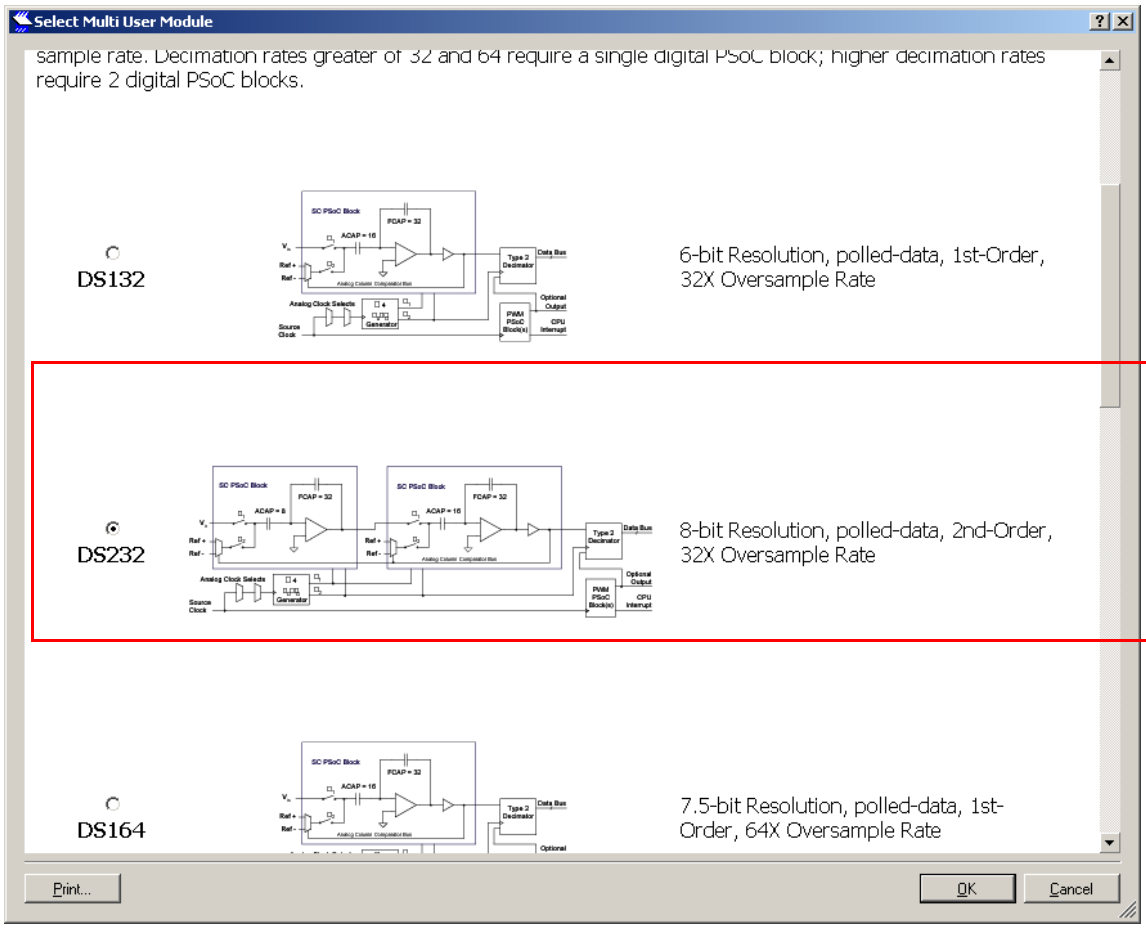
3.1.3 ADC to LCD with DAC and UART

This project demonstrates sine wave generation by using an 6-bit DAC. The sine wave period is based on the current value of the ADC. The firmware reads the voltage output by the DVK board potentiometer and displays the raw counts on the DVK board character LCD display similarly to those shown in the previous project. An 6-bit DAC outputs a table generated sine wave at a frequency proportional to the ADC count. The frequency is in the approximate range of 15 Hz to 350 Hz and outputs to an LED. A 38400 BAUD UART outputs the current ADC count as ASCII formatted into a hexadecimal number.

3.1.3.1 Creating ADC to LCD with DAC and UART Project

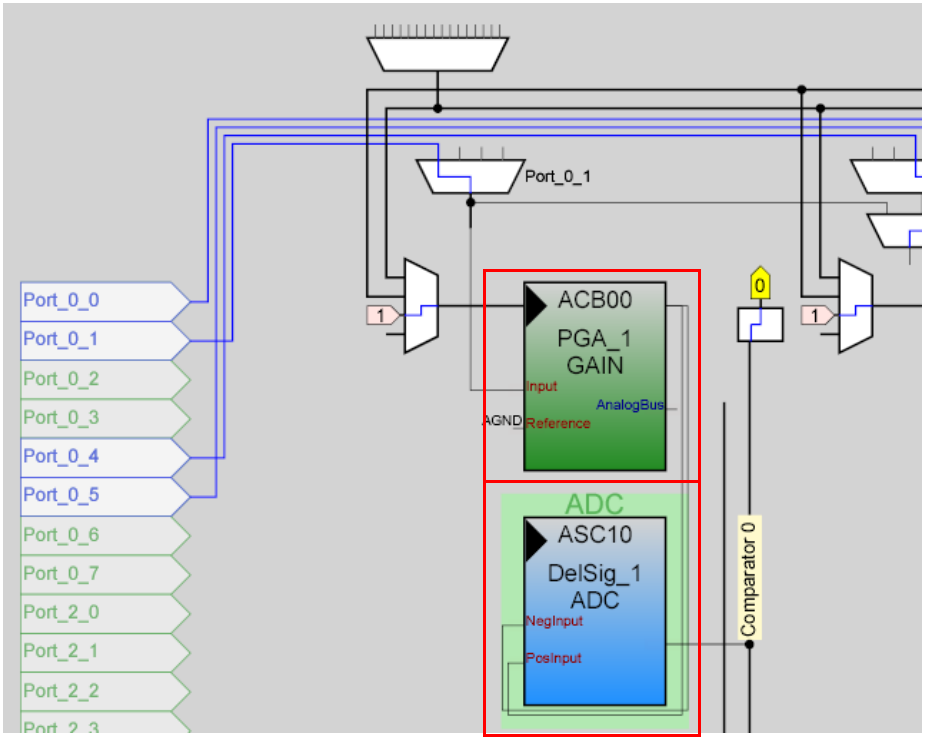
1. Follow steps 1 to 10 in section 3.1.1.1 on page 17, change the **Name** of the project to **Example_ADC_to_LCD_with_DAC_and_UART**.
2. In the **User Modules** window expand the **ADCs** folder, and then expand the **DelSig** subfolder. right click **DS232** and choose place. A window appears with multiple options for the DelSig UM. In this case the **DS232** configuration is used. Scroll down in the window for verification.

Figure 3-13. Select Multi User Module Window



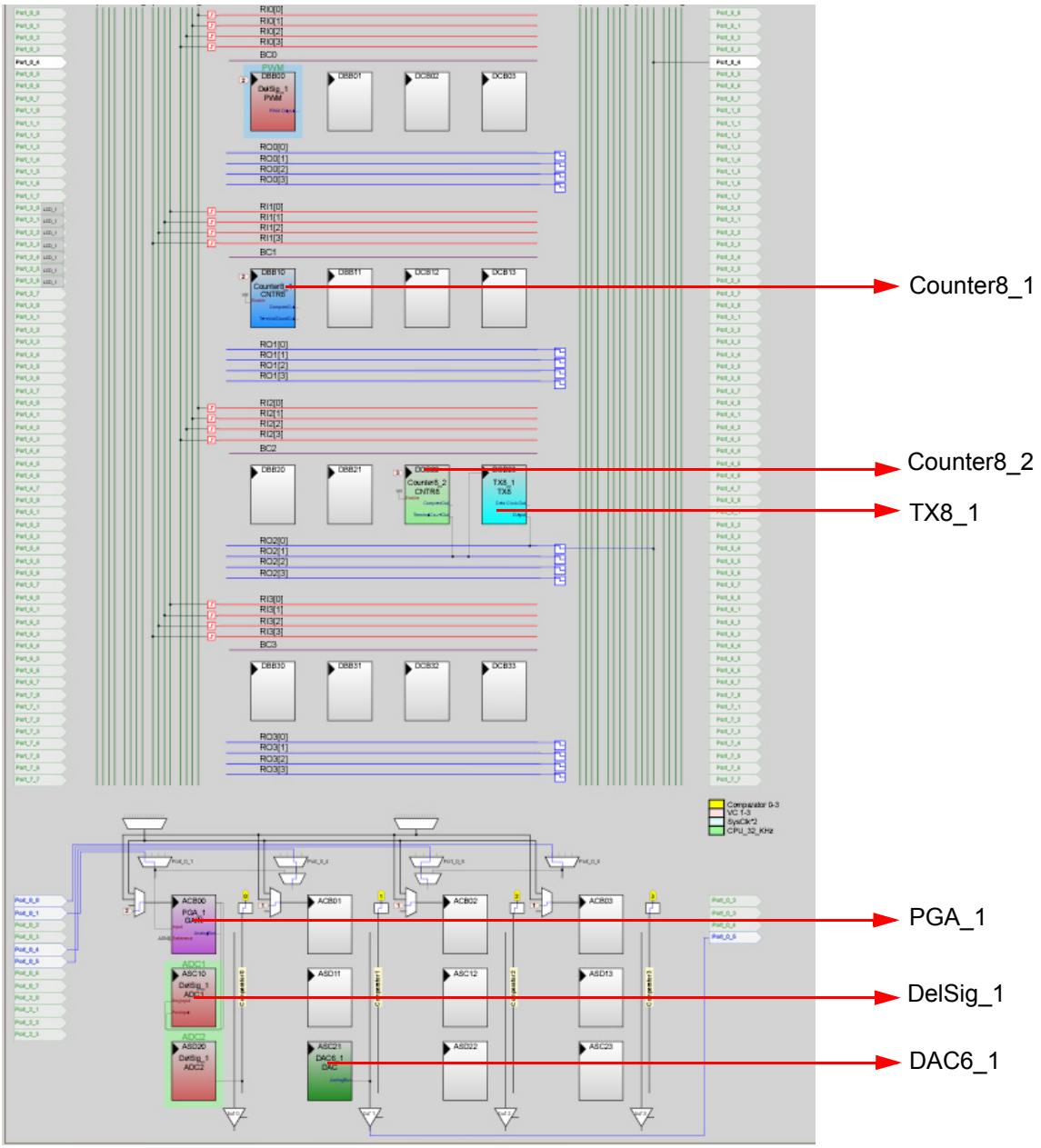
3. Click **OK**.

4. Verify that the UM is placed in **ASC10**.
5. In the **User Modules** window expand the **Amplifiers** window. Right click **PGA**, select place. Ensure that the **PGA** is placed in **ACB00**.

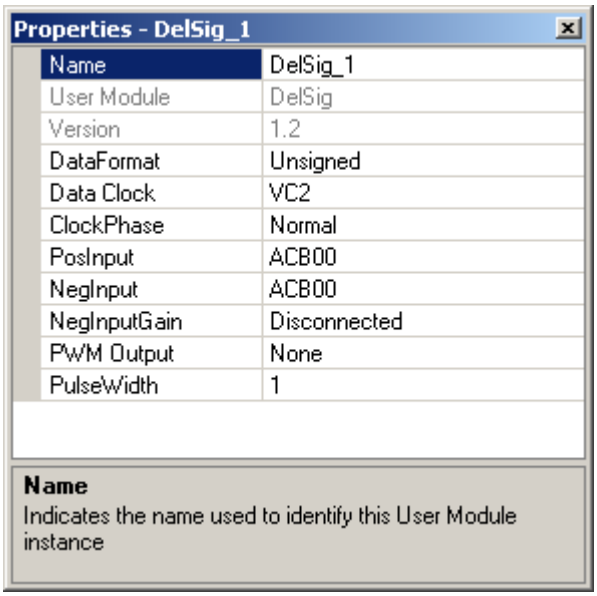


6. In the **User Modules** window expand **Misc Digital**, right click **LCD**, and click place.
7. In the **User Modules** window expand **Counters**, right click **Counter8**, and select place. Complete this step twice to place two **Counter8s**.
8. In the **User Modules** window expand **Digital Comm**, right click **TX8**, and click place.
9. In the **User Modules** window expand **DACs**, right click **DAC6**, and click place.

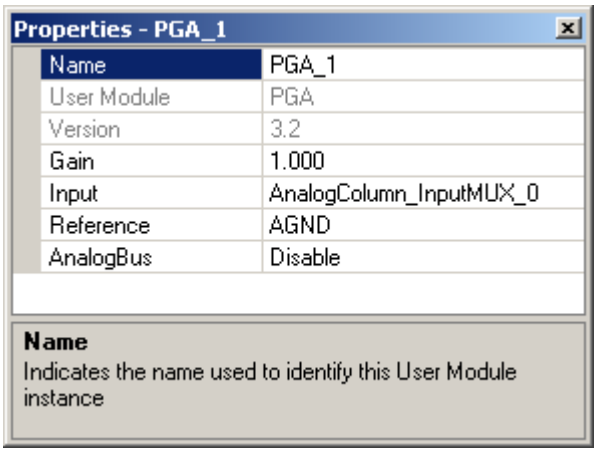
10. Move the UMs so that they match the configuration shown in this figure.



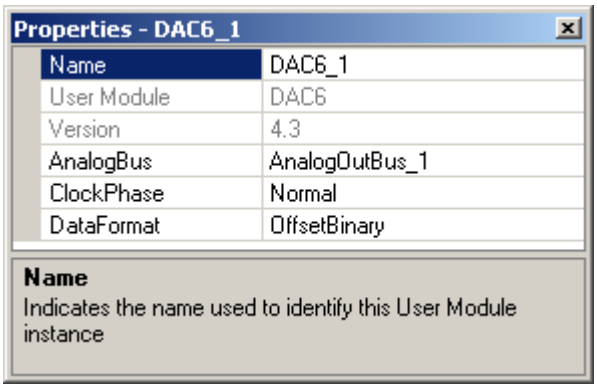
11. Double click on **DelSig_1** and configure it to match this figure.



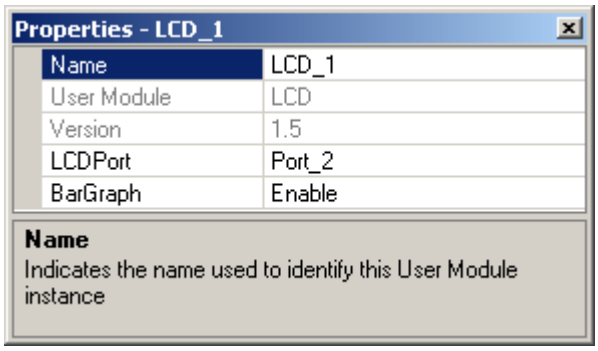
12. Double click **PGA_1** and configure it to match this figure.



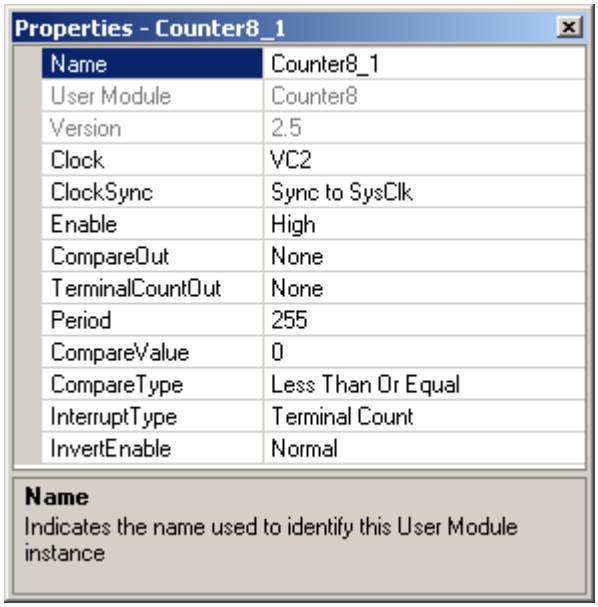
13. Double click **DAC6_1** and configure it to match this figure.



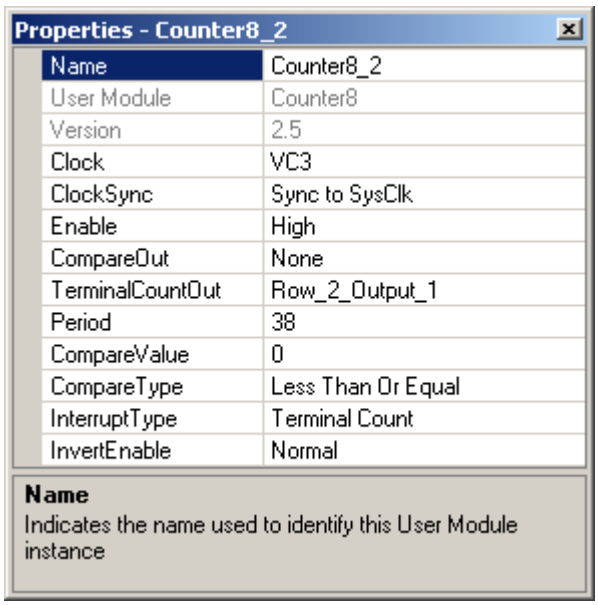
14. Double click **LCD_1** and configure it to match this figure.



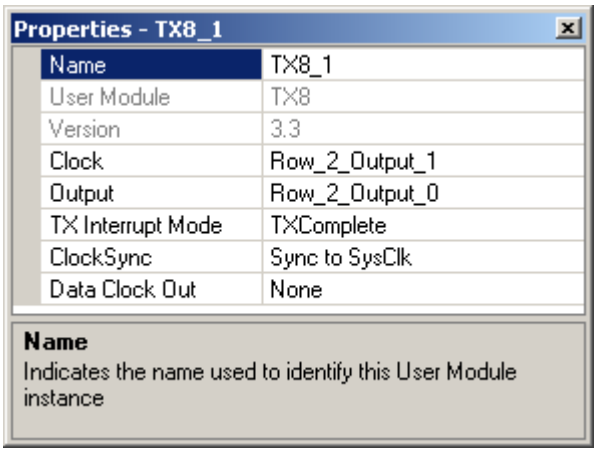
15. Double click on **Counter8_1** and configure it to match this following figure.



16. Double click **Counter8_2** and configure it to match this figure.

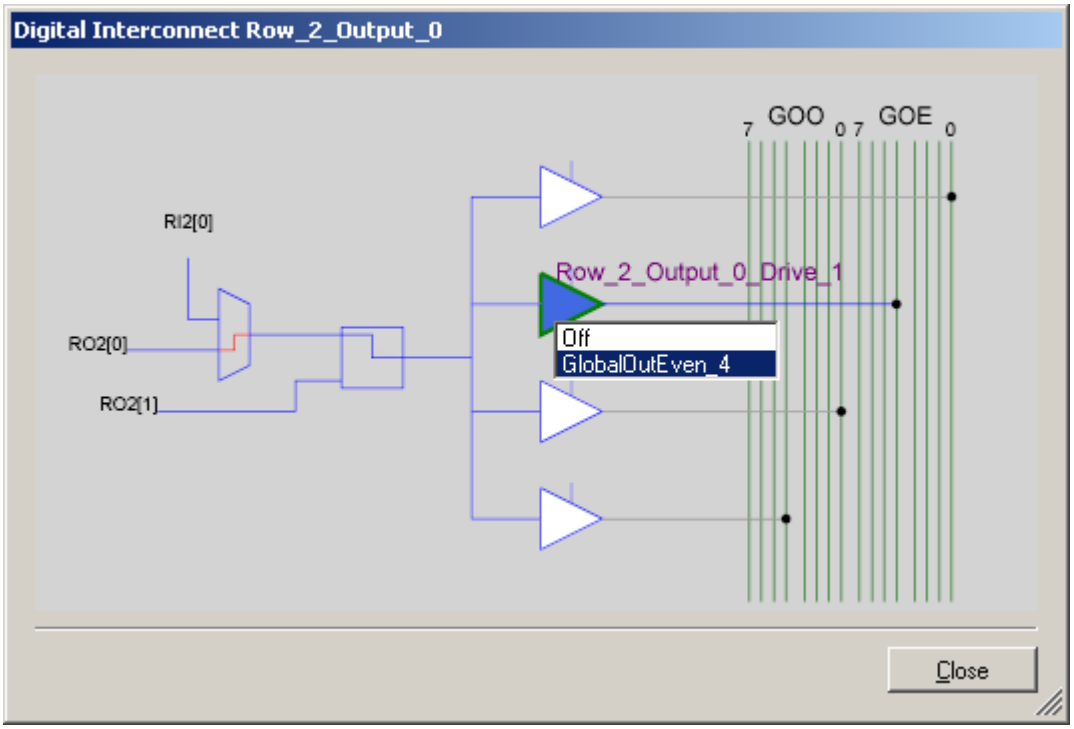


17. Double click **TX8_1** and configure it to match this figure.

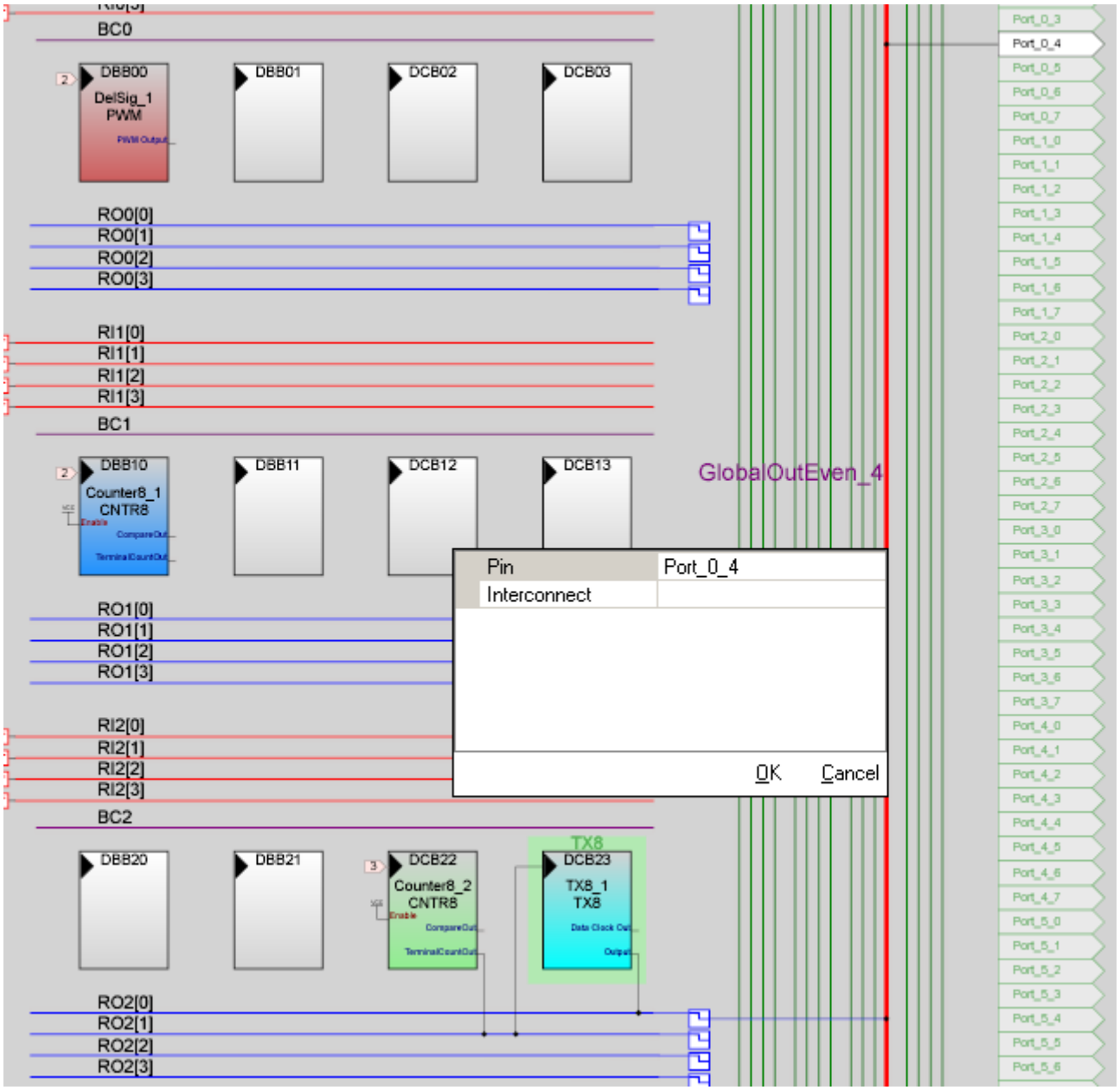


18. Double click **RO2[0] LUT**, enable **Row_2_Output_0_Drive_1** to connect **GlobalOutEven_4**.

Figure 3-14. Digital Interconnect Window

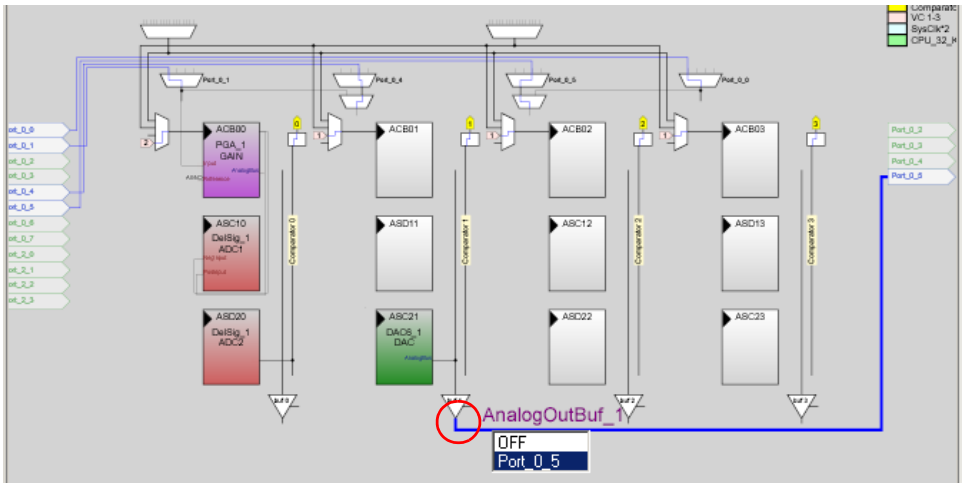


19. Double click **GlobalOutEven_4**. A window appears; in this window configure **PIN** for **Port_0_4**.

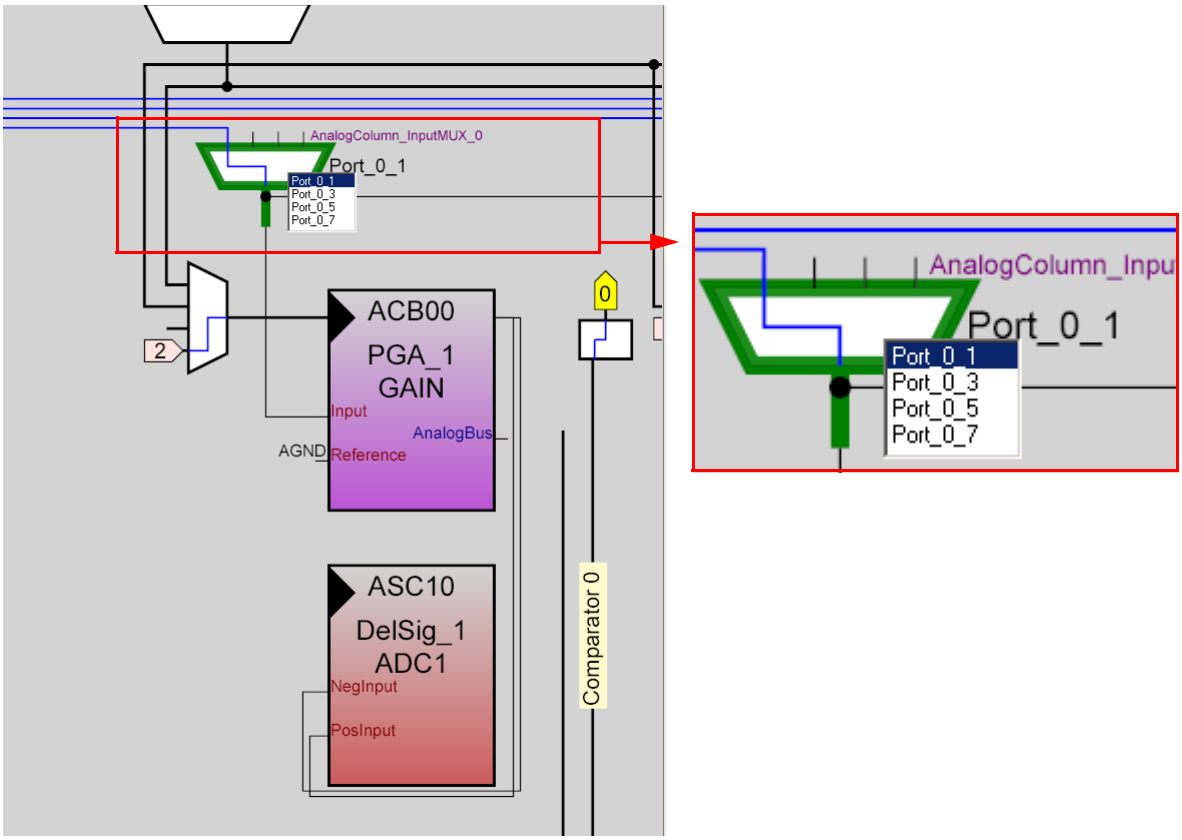


20. Click **OK** to continue.

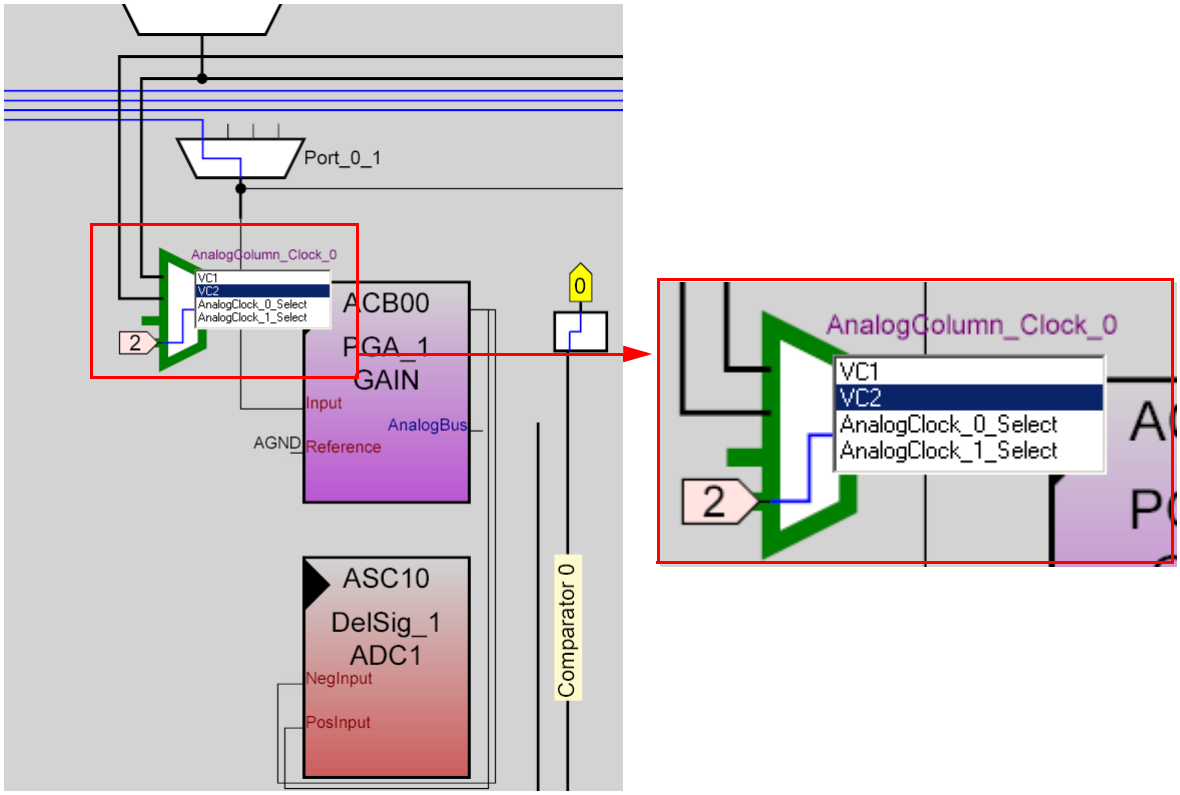
21. Click **AnalogOutBuf_1** and configure it for **Port_0_5**.



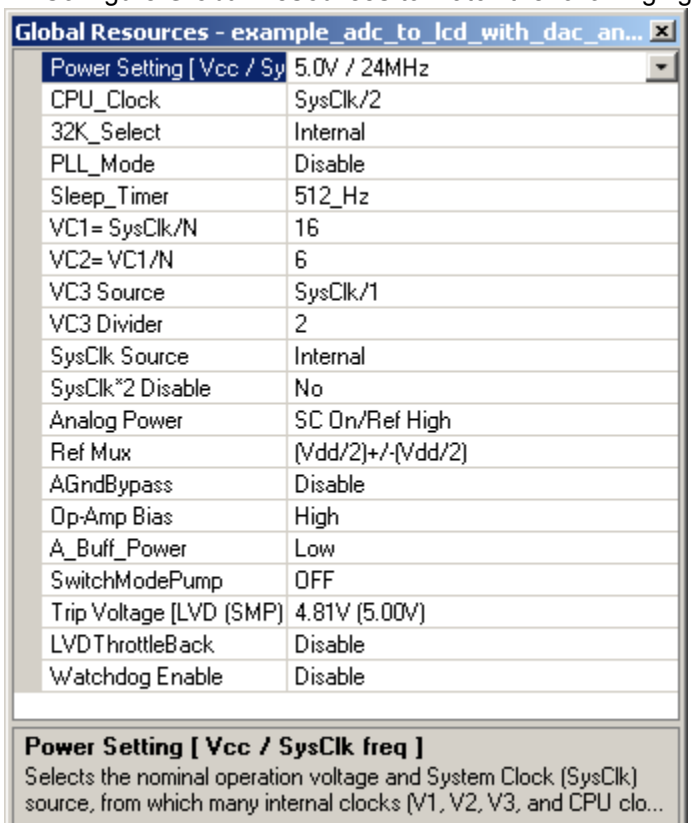
22. Verify that **AnalogColumn_InputMUX_0** is connected to **Port_0_1**. If it is not configured for this port, double click the **MUX** and choose **Port_0_1**.



23. Verify that **AnalogColumn_Clock_0**, is connected to **VC2**. If it is not, double click the **MUX** and chose **VC2**.



24. Configure **Global Resources** to match the following figure.



25. Open your *main.c* file and copy the example code located in section 3.1.3.2 on page 43 to the existing *main.c*.
26. Open your *Counter8_1INT.asm* file in **Files** → **lib** → **Library Source Files**. Copy the example code located in section 3.1.3.3 on page 46 to the existing *Counter8_1INT.asm*.
27. Save the project.
28. To Build the project, click **Build** → **Generate/Build 'Example_ADC_to_LCD_with_DAC_and_UART'** Project.
29. Disconnect power to the board.
30. Configure the DVK bread board SW3 to 5V.
31. Configure the DVK bread board using the included jumper wires as follows:
    - P0_1 to VR
    - P0_4 to Tx
    - P0_5 to Scope
32. Connect a serial cable to the PC and the DVK board.
33. On the DVK board, verify that **RS232_PWR(J10)** is jumpered to **ON**.
34. Reapply power to the board.
35. Use a terminal application such as TeraTerm or HyperTerminal with these setup parameters.
    - Baud Rate: 38400
    - Data: 8-bit
    - Parity: none
    - Stop: 1bit
    - Flow Control: none
36. Use PSoC Designer as described in Programming My First PSoC Project on page 11 to program the device.
37. After programming the device, press Reset and vary the pot to see the result on the LCD as well as in the terminal application. View the DAC output on a scope or with an LED.
38. Save and close the project.

3.1.3.2 main.c

```
/*******************************************************************************
* File Name: main.c
*
* Description:
*  This file provides source code for the ADC to LCD with DAC and UART example
*  project. The firmware takes a voltage output from a potentiometer and
*  displays the ADC raw count on an LCD. The raw count is also transmitted
*  serially. The raw count also determines the clock divider value of the clock
*  driving the DAC update rate.

/*******************************************************************************
*   PGA_1 Settings:(The PGA buffers the potentiometer voltage on P0.1 into the ADC)
*
*     Gain      = 1
*     Input     = AnalogColumn_InputMUX_0  (P0.1)
*     Reference = AGND
*     AnalogBus = Disable
*
*******************************************************************************/
/*******************************************************************************
*   LCD_1 Settings:
*     LCDPort  = Port_2
*     BarGraph = Disable
*
*******************************************************************************/
/*******************************************************************************
*   DelSig_1 Settings:
*   The ADC can read full range values from 0-5V, if the Ref Mux setting is
*   selected as (Vdd/2)+/- (Vdd/2) and Vdd = 5V. The ADC is configured for a
*   resolution of 8 bits, this is achieved by selecting the appropriate
*   configuration when placing the UM.
*
*     DataFormat   = Unsigned
*     DataClock    = VC2 //VC2 = 24MHz/16/16 = 250kHz
*     ClockPhase   = Normal
*     PosInput     = ACB00 (PGA_1)
*     NegInput     = ACB00   *Note this parameter is not used
*     NegInputGain = Disconnected
*     PWM Output   = None
*     PulseWidth   = N/A   *Note this parameter is not used
*
*******************************************************************************/
/*******************************************************************************
*   Counter8_1 Settings:
*   The Counter8_1 controls the update rate of the DAC. The DAC is updated
*   during ever TerminalCount ISR. The frequency of the TerminalCount ISR is
*   determined by the Counter Input Clock divided by the (Period value +1).
*   The Period Value of the counter is changed by the ADC reading. Thus the
*   frequency of the TerminalCount ISR can range from 125kHz (Period Value=1)
*   to 977Hz (Period Value = 255)
*
*     Clock            = VC2 // VC2 = 24MHz/16/16 = 250kHz
*     ClockSync        = Sync to SysClk
*     Enable           = High
*     CompareOut       = None
*     TerminalCountOut = None
```

```
*     Period           = 255   *Note this parameter is updated in the
*                        main loop
*     CompareValue     = 0 *Note this parameter is not used
*     CompareType      = Less Than or Equal
*     InterruptType    = Terminal Count
*     InvertEnable     = Normal
*
********************************************************************************/
/*******************************************************************************
*  Counter8_2 Settings:
*  The Counter8_1 provides a clock to the TX8 UM to achieved a desired baud
*  rate. For this project the desired baud rate is 38400. The TX8 UM
*  derives the baud rate by dividing its input clock by 8. Thus the input
*  clock to the TX8 needs to be around 307.2kHz to achieve a baud rate of
*  38400. The Counter8_1 UM provides this clock by dividing VC3 (12MHz) by
*  39 to get 307.7kHz.
*
*     Clock            = VC3 //VC3 = 24MHz/2 = 12MHz
*     ClockSync        = Sync to SysClk
*     Enable           = High
*     CompareOut       = None
*     TerminalCountOut = Row_2_Output_1
*     Period           = 38
*     CompareValue     = 0*Note this parameter is not used
*     CompareType      = Less Than or Equal
*     InterruptType    = Terminal Count
*     InvertEnable     = Normal
*
********************************************************************************/
/*******************************************************************************
*  TX8_1 Settings:
*  The TX8 UM provides serial communication of the ADC data to another
*  device or PC. The TX8 UM send data out at a baud rate of 38400. This
*  baud rate is derived by dividing the UM's input clock by 8.
*
*     Clock             = Row_2_Output_1 (From Counter8_1)
*     Output            = Row_2_Output_0
*     Tx Interrupt Mode = TXComplete
*     ClockSync         = Sync to SysClk
*     Data Clock Out    = None

********************************************************************************/
/*******************************************************************************
*  DAC6 Settings:
*  The DAC6 outputs a sine wave on P0.5. The shape of the sine wave is
*  determined by a 64 element lookup table found in SINtable.asm. The=
*  update rate of the DAC6 is determined by the Counter8 terminal count ISR.
*  The frequency of the DAC output equals the Counter8 Terminal Count
*  frequency divided by 64 (the number of elements in the table).
*
*     AnalogBus  = AnalogOutBus_1
*     ClockPhase = Normal
*     DataFormat = OffsetBinary
*
********************************************************************************/

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

```
const BYTE SINtable[]=
{
  31, 33, 36, 39, 41, 44, 46, 49, 51, 53, 55, 56, 58, 59, 59,
  60, 60, 60, 59, 59, 58, 56, 55, 53, 51, 49, 47, 44, 42, 39,
  36, 33, 31, 28, 25, 22, 19, 16, 13, 11, 9, 7, 5, 3, 2, 1, 0,
  0, 0, 0, 1, 2, 3, 4, 6, 7, 10, 12, 14, 17, 20, 23, 26, 29
};

BYTE bADCvalue;//Variable for holding ADC result, and updating counter period

void main()
{
    Counter8_1_Start();             //Enable the counter used for DAC update rate
    Counter8_1_EnableInt();         //Enable DAC update interrupt
    Counter8_2_Start();             //Enable counter for TX8 clock rate divider
    TX8_1_Start(TX8_1_PARITY_NONE);   //Start the TX8 UM with no parity (baud rate
                                      // = 38400)
    PGA_1_Start(PGA_1_HIGHPOWER);     //Enable to PGA to buffer signal from VR to
                                      // ADC
   DAC6_1_Start(DAC6_1_HIGHPOWER);    //Start the DAC
   DelSig_1_Start(DelSig_1_HIGHPOWER);//Start the ADC
   DelSig_1_StartAD();                //Start reading values on the ADC
   LCD_1_Start();                     //Start the character LCD

   M8C_EnableGInt;                    // Enable Global Interrupts

   while(1)
   {
   /* Step 1: Get BYTE data from the ADC
   Step 2: Write BYTE data from ADC to the counter in order to change the
      DAC update rate
   Step 3: Move the LCD cursor back to the beginning and display new ADC data
   Step 4: Write ADC data out the TX port, and then send a return
   */
      if (DelSig_1_fIsDataAvailable())//Is new data available from the ADC?
      {


      bADCvalue = DelSig_1_bGetDataClearFlag();  //Get new data from ADC
      Counter8_1_WritePeriod(bADCvalue);//Update DAC update rate counter
      LCD_1_Position(0,0);//Move LCD (row=0,column=0)
      LCD_1_PrHexByte(bADCvalue);      //Print ADC result to LCD
      TX8_1_PutSHexByte(bADCvalue);    //Write LCD result out TX8 to PC
      TX8_1_PutCRLF();      //Send a return character

      }
   } //end of while(1)


}  //End of Main
```

3.1.3.3 Counter8_1INT.asm

```
;;*****************************************************************************
;;*****************************************************************************
;;  FILENAME: Counter8_1INT.asm
;;   Version: 2.5, Updated on 2009/3/31 at 12:2:49
;;  Generated by PSoC Designer 5.0.423.0
;;
;;  DESCRIPTION: Counter8 Interrupt Service Routine
;;-----------------------------------------------------------------------------
;;  Copyright (c) Cypress MicroSystems 2000-2004. All Rights Reserved.
;;*****************************************************************************
;;*****************************************************************************

include "m8c.inc"
include "memory.inc"
include "Counter8_1.inc"


;-----------------------------------------------
;  Global Symbols
;-----------------------------------------------
export  _Counter8_1_ISR


AREA InterruptRAM (RAM,REL,CON)

;@PSoC_UserCode_INIT@ (Do not change this line.)
;---------------------------------------------------
; Insert your custom declarations below this banner
;---------------------------------------------------
export bTablePos// Stores last table position index
export _bTablePos

;------------------------
; Includes
;------------------------


;------------------------
;  Constant Definitions
;------------------------


;------------------------
; Variable Allocation
;------------------------
area bss(RAM)
bTablePos:blk 1
_bTablePos:
;---------------------------------------------------
; Insert your custom declarations above this banner
;---------------------------------------------------
;@PSoC_UserCode_END@ (Do not change this line.)


AREA UserModules (ROM, REL)
```

```
;-------------------------------------------------------------------------
;  FUNCTION NAME: _Counter8_1_ISR
;
;  DESCRIPTION: Unless modified, this implements only a null handler stub.
;
;-------------------------------------------------------------------------
;

_Counter8_1_ISR:

   ;@PSoC_UserCode_BODY@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.

   push A
   push X

   dec [bTablePos]    ;Go to the next element in the table
   mov A, [bTablePos]
   jnz SINlookup      ;If we are at the end go back to the beginning
   mov [bTablePos], 64

SINlookup:
   index   _SINtable;Get the value in the SINtable pointed to by [bTablePos]
   lcall   DAC6_1_WriteBlind;Write value from SINtable (stored in A) to the DAC

   pop X
   pop A

   ;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)

   reti


; end of file Counter8_1INT.asm
```

3.2 CY8C38 Family Processor Module Example Projects

3.2.1 My First PSoC Project

This project demonstrates basic hardware and software functionality with the PSoC3 device. It flashes two LEDs independently, one using hardware, the other with software. The hardware LED uses a hardware enabled digital port and a PWM to generate a duty cycle and flash the LED. The software LED uses a software enabled digital port and a simple delay in the *main.c* to flash the LED at a known rate.

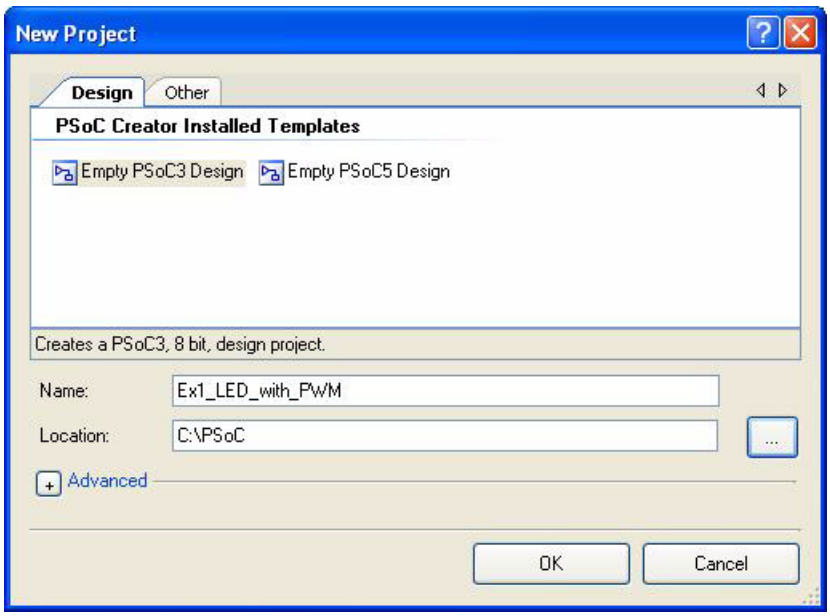
This example project uses these components:

- Digital Port (**Component Catalog** → **System** → **Digital Port**)
- PWM (**Component Catalog** → **Digital Functions** → **PWM**)
- Clock (**Component Catalog** → **System** → **Clock**)
- Logic Low
- Logic High

3.2.1.1 Creating My First PSoC Project

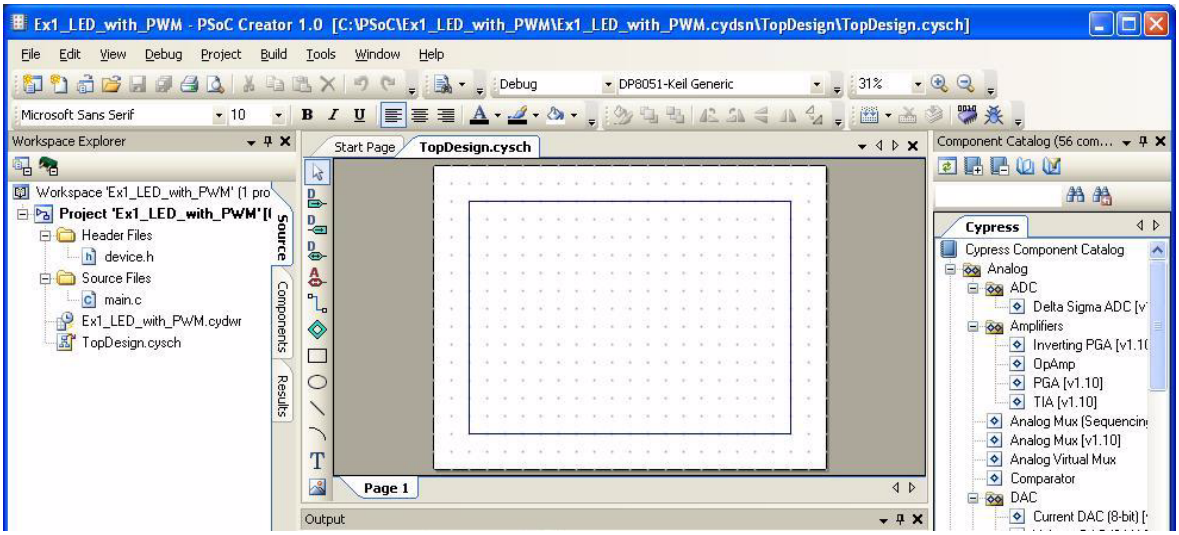
1. Open PSoC Creator.
2. Create a new project by clicking **Create New Project…** in the **Start Page** of PSoC Creator.
3. In the **New Project** window, select the **Empty PSoC3 Design** template and name the project **Ex1_LED_with_PWM**.
4. In **Location**, type the path where you want to save the project, or click the ... button and navigate to the appropriate directory.

Figure 3-15. New Project Window



5. By default, the design window opens *TopDesign.cysch*. This is the project's schematic entry file within PSoC Creator.

Figure 3-16. Ex1_LED_with_PWM



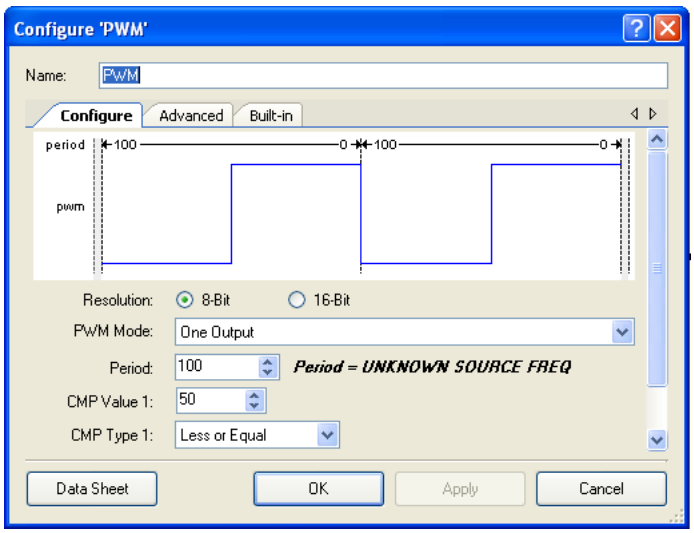
3.2.1.2 *Placing and Configuring the PWM*

1. Drag-and-drop the **PWM** component (**Component Catalog** → **Digital** → **Functions** → **PWM**) to-workspace.
2. Double click the **PWM_1** component in the schematic to open the configuration window.
3. Configure the PWM in this manner:

   Configure Tab

   - **Name**: PWM
   - **Resolution**: 8-Bit
   - **PWM Mode**: One Output
   - **Period**: 100
   - **CMP Value 1**: 50
   - **CMP Value Type 1**: Less or Equal

Figure 3-17. PWM Component Configuration



Advanced Tab

- **Enable Mode**: Hardware Only
- **Interrupt On Terminal Count Event**: Select

For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

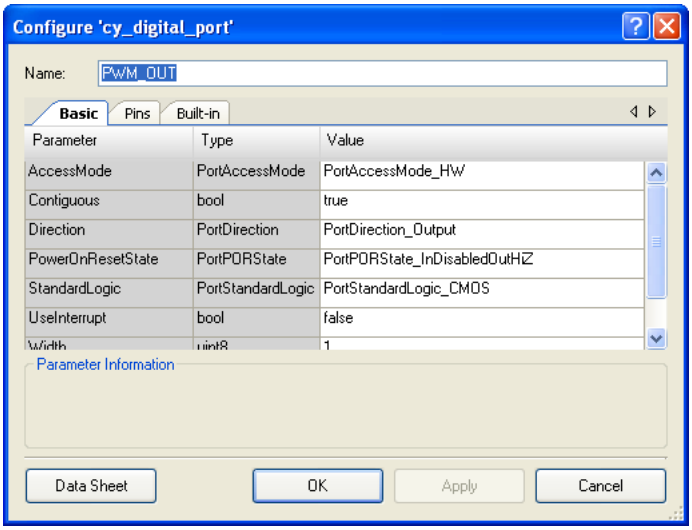
3.2.1.3 *Placing and Configuring the Digital Port Hardware*

1. Drag-and-drop the **Digital Port** component (**Component Catalog** → **System** → **Digital Port**).
2. Double click the **dPort_1** component in the schematic to open the configuration window.
3. Configure the digital port:

Basic Tab

- **Name**: PWM_OUT
- **AccessMode**: PortAccessMode_HW
- **Direction**: PortDirection_Output
- **Width**: 1

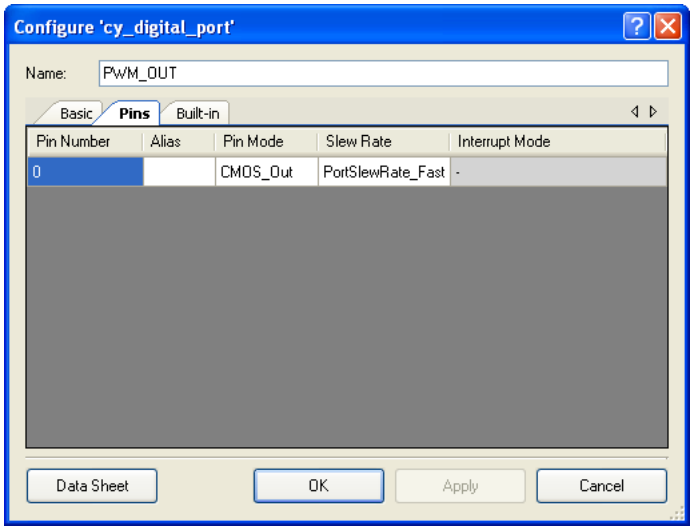
Figure 3-18. PWM_OUT Component Configuration



Pins Tab

- **Pin Mode**: CMOS_Out
- Leave remaining parameters set to their default values

Figure 3-19. Pins - PWM_OUT Component Configuration



For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

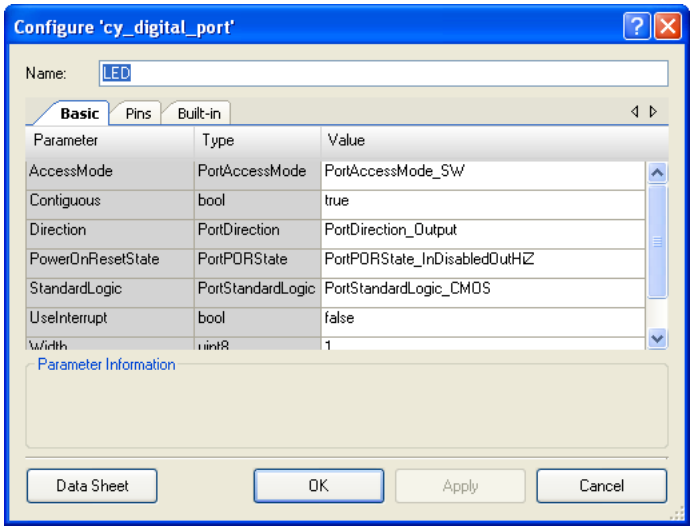
3.2.1.4 *Placing and Configuring the Software Digital Port*

1. Drag-and-drop the **Digital Port** component (**Component Catalog** → **System** → **Digital Port**).
2. Double click the **dPort_1** component in the schematic to open the configuration window.
3. Configure the digital port:

   Basic Tab

   - **Name**: LED
   - **AccessMode**: PortAccessMode_SW
   - **Direction**: PortDirection_Output
   - **Width**: 1

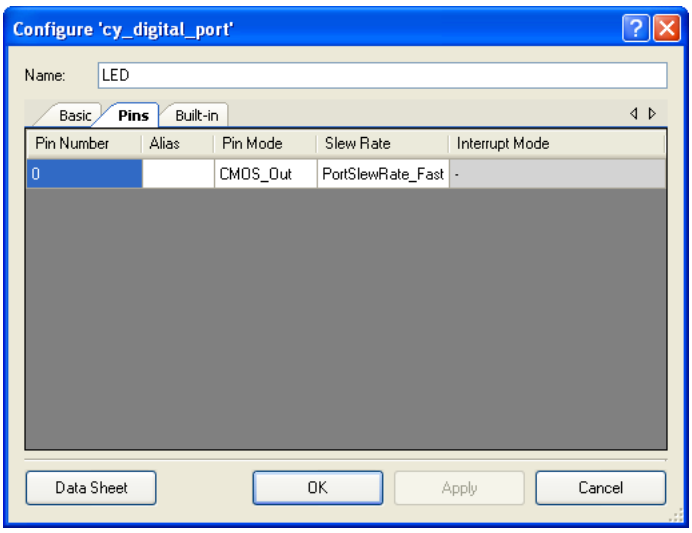
Figure 3-20. LED Component Configuration



Pins Tab

- **Pin Mode**: CMOS_Out
- Leave remaining parameters to default

Figure 3-21. Pins - LED Component Configuration



For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

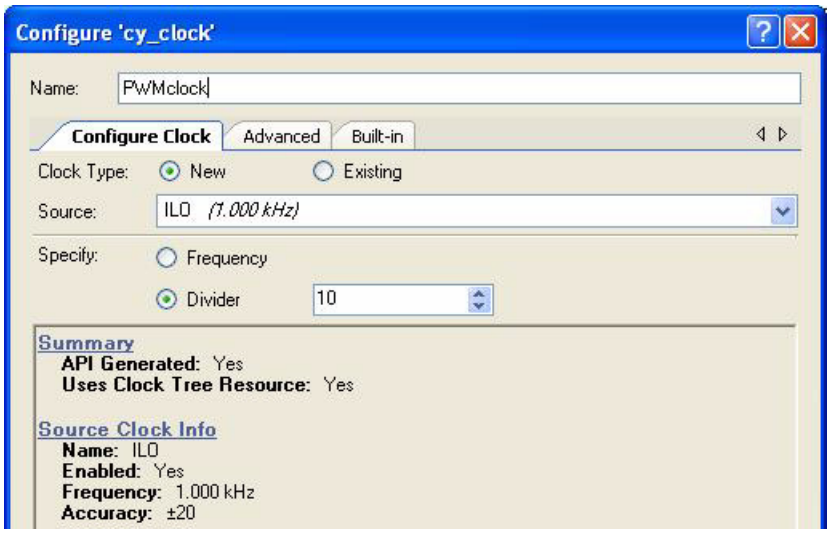
3.2.1.5 *Connecting the Components Together*

1. Using the Wire Tool , connect **pwm** (in the PWM component) to **o** on the digital port (PWM_OUT).
2. Connect a Logic High component (**Component Catalog** → **Digital Logic** → **Logic High**) to the **enable** on the PWM
3. Connect a Logic Low component (**Component Catalog** → **Digital Logic** → **Logic Low**) to the **reset** on the PWM
4. Connect a Logic High component (**Component Catalog** → **Digital Logic** → **Logic High**) to the **oe** on the digital port (PWM_OUT)
5. Connect a Clock component (**Component Catalog** → **System** → **Clock**) to the **clock** on the PWM.
6. Double click the **Clock** component to configure.
7. Configure the clock:

   Configure Clock Tab

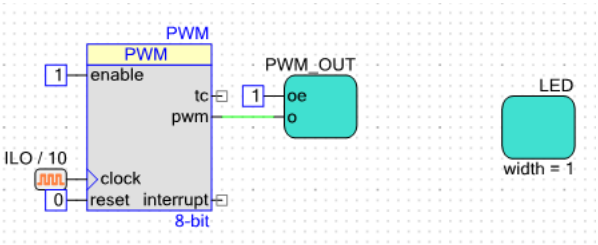
   - **Name**: PWMclock
   - **Source**: ILO (1.000 kHz)
   - Select **Divider** and set the value as 10
   - Leave remaining parameters to default

Figure 3-22. Clock Component Configuration



8. When complete the schematic looks like Figure 3-23.

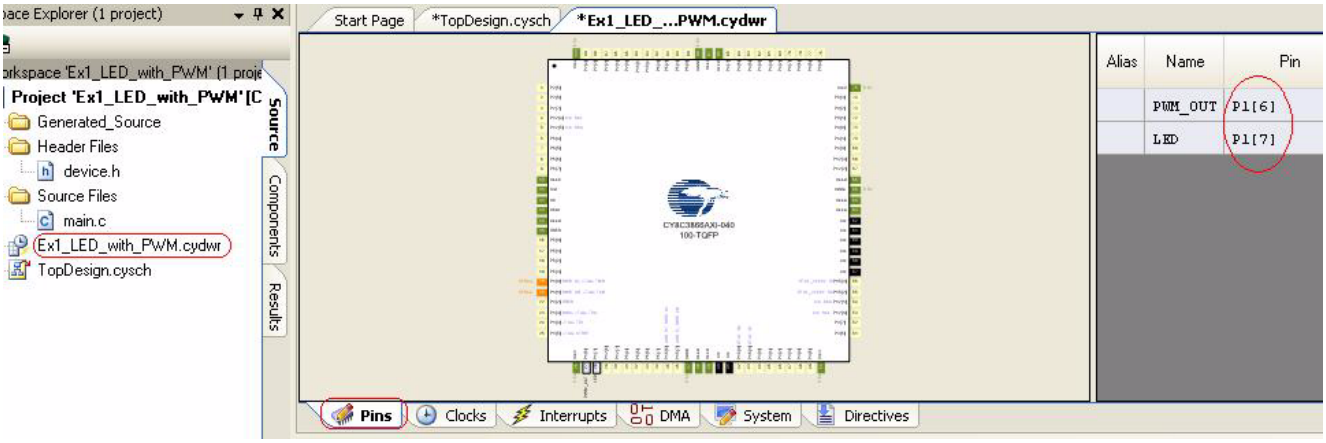
Figure 3-23. Connected Components



3.2.1.6 *Configuring the Pins*

1. From the **Workspace Explorer**, double click the *Ex1_LED_with_PWM.cydwr* file (see Figure 3-24 on page 54).
2. Click the **Pins** tab.
3. Select pin P1[6] for PWM_OUT.
4. Select pin P1[7] for LED.

Figure 3-24. Set P1[6] to PWM_OUT



3.2.1.7 *Creating the main.c File*

1. From the **Workspace Explorer** double click the *main.c* file
2. Use this code to replace the contents of the *main.c* file. (A soft copy of the *main.c* file is embedded in this PDF under “Attachments.”)

main.c

```
#include <device.h>

#define MS_DELAY 167u /* For delay, about 167ms */


/*******************************************************************************
* Function Name: main
********************************************************************************
*
* Summary:
*  The main function initializes the PWM and starts the PWM clock which
*  will blink LED1 at about once a second. Then the main loop is entered
*  which delays enough for LED2 to blink at a quicker rate than LED1.
*
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
void main(void)
{
    uint8 ledState = 0x00; /* Initially set LED to off */

    PWMclock_Enable(); /* Start the clock */
    PWM_Start();       /* Enable PWM      */

    /* Following loop does software blinking of LED connected to P0.1 */
    while (1)
    {
        CyDelay(MS_DELAY);   /* Have software loop blink control   */
```

```
            ledState ^= 0x01u;   /* Toggle LED setting between low and high */
            LED_Write(ledState); /* Set the LED */
        }
    }


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

3. From the **Build** menu, select **Build Ex1_LED_with_PWM**.
4. PSoC Creator builds the project and displays the comments in the **Output** dialog box. When you see the message “Build Succeeded” the build is complete.

3.2.1.8 *Configuring and Programming the PSoC Development Board*

1. Disconnect the power to the board.
2. Configure the DVK bread board SW3 to 3.3V.
3. Configure the following on the PSoC Development Board's prototyping area using the included jumper wires
   - P1[6] to LED1
   - P1[7] to LED2
4. Apply power to the board.
5. Use PSoC Creator as described in Programming a Device on page 15 to program the device.
6. After programming the device, press the **Reset** button on the PSoC Development Board. The PWM causes the LED1 to blink at approximately 1 Hz due to PSoC Creator's PWM component and LED2 blinks at a slower rate using a software timing loop to toggle the LED.
7. Save and close the project.

3.2.2 ADC to LCD Project

This project demonstrates the Delta Sigma ADC by measuring the voltage of the potentiometer on the board and displays the result on the Character LCD of the PSoC Development Board.

The ADC is clocked by the internal clock of 3 MHz and the sampling rate is set to 10,000 sps. Connect the voltage potentiometer (labeled “VR” on the PSoC Development Board) to the ADC input (programmed to P0[7] for this example). The program reads the ADC result and prints it to the LCD.

The instructions that follow assume that the user has completed My First PSoC Project and therefore has a basic understanding of the PSoC Creator software environment.

This example project uses these components:

- Delta Sigma ADC (**Component Catalog** → **Analog** → **ADC** → **Delta Sigma ADC**)
- Character LCD (**Component Catalog** → **Display** → **Character LCD**)
- Analog Port (**Component Catalog** → **System** → **Analog Port**)

3.2.2.1 *Creating ADC to LCD Project*

1. Open PSoC Creator.
2. Create a new project by clicking **Create New Project…** in the **Start Page** of PSoC Creator.
3. In the **New Project** window, select the **Empty PSoC3 Design** template and **Name** the project **Ex2_ADC_to_LCD**.
4. In **Location**, type the path where you want to save the project, or click  and navigate to the appropriate directory.

5. By default, the design window opens *TopDesign.cysch*. This is the project's schematic entry file within PSoC Creator.

3.2.2.2 *Placing and Configuring Delta Sigma ADC*

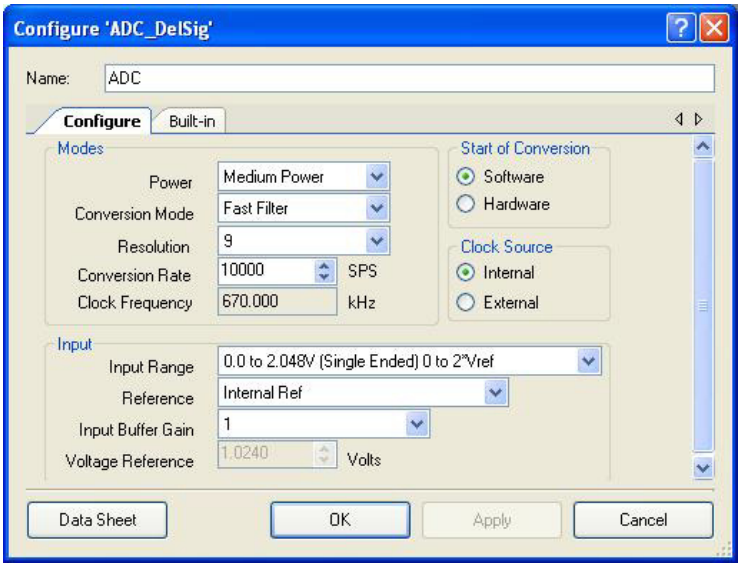
1. Drag and drop the Delta Sigma ADC component (**Component Catalog** → **Analog** → **ADC** → **Delta Sigma ADC**)
2. Double click the **ADC_DelSig_1** component in the schematic to open the configuration window.
3. Configure the digital port in this manner:

   Configure Tab

   - **Name**: ADC
   - **Power**: Medium Power
   - **Conversion Mode**: Fast Filter
   - **Resolution**: 9
   - **Conversion Rate**: 10000
   - **Input Range**: 0.0 to 2.048V (Single Ended) 0 to 2*Vref
   - **Input Buffer Gain**: 1
   - **Reference**: Internal Ref
   - **Clock Source**: Internal
   - **Start of Conversion**: Software

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-25. ADC Component Configuration



3.2.2.3 *Placing and Configuring the Analog Port*

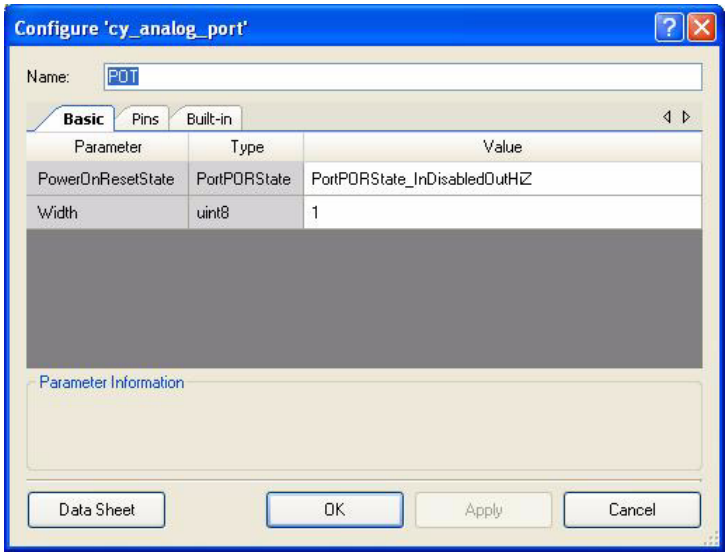
1. Drag-and-drop the Analog Port component (**Component Catalog** → **System** → **Analog Port**).
2. Double click on the **aPort_1** component in the schematic to open the configuration window.
3. Configure the digital port in this manner:

   Basic Tab

   - **Name**: POT
   - **Width**: 1

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-26. Analog Port Component Configuration

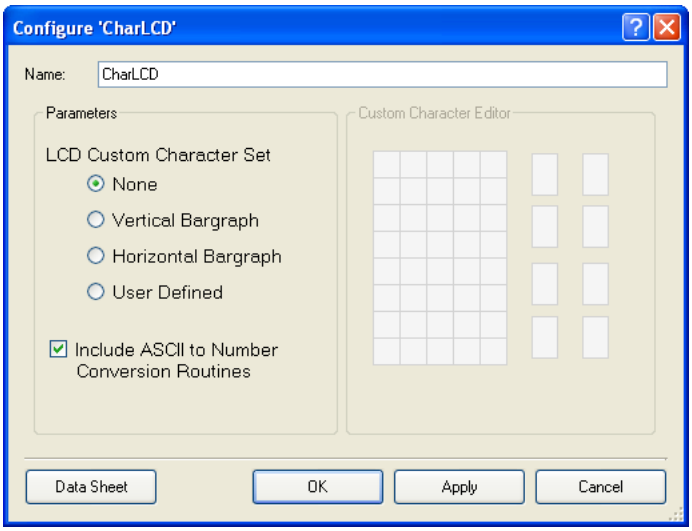


3.2.2.4 *Placing and Configuring Character LCD*

1. Drag-and-drop the Character LCD component (**Component Catalog** → **Display** → **Character LCD**)
2. Double click the **LCD_Char_1** component in the schematic to open the configuration window.
3. Configure the digital port:
   - **Name**: CharLCD
   - **LCD Custom Character Set**: None
   - **Include ASCII to Number Conversion Routines**: Check box

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

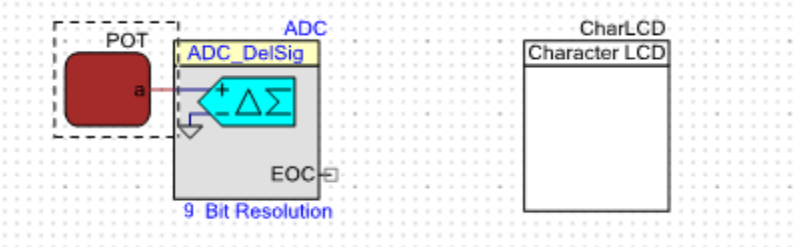
Figure 3-27. Configure CharLCD



3.2.2.5 *Connecting the Components Together*

1. When complete the schematic looks like Figure 3-28.

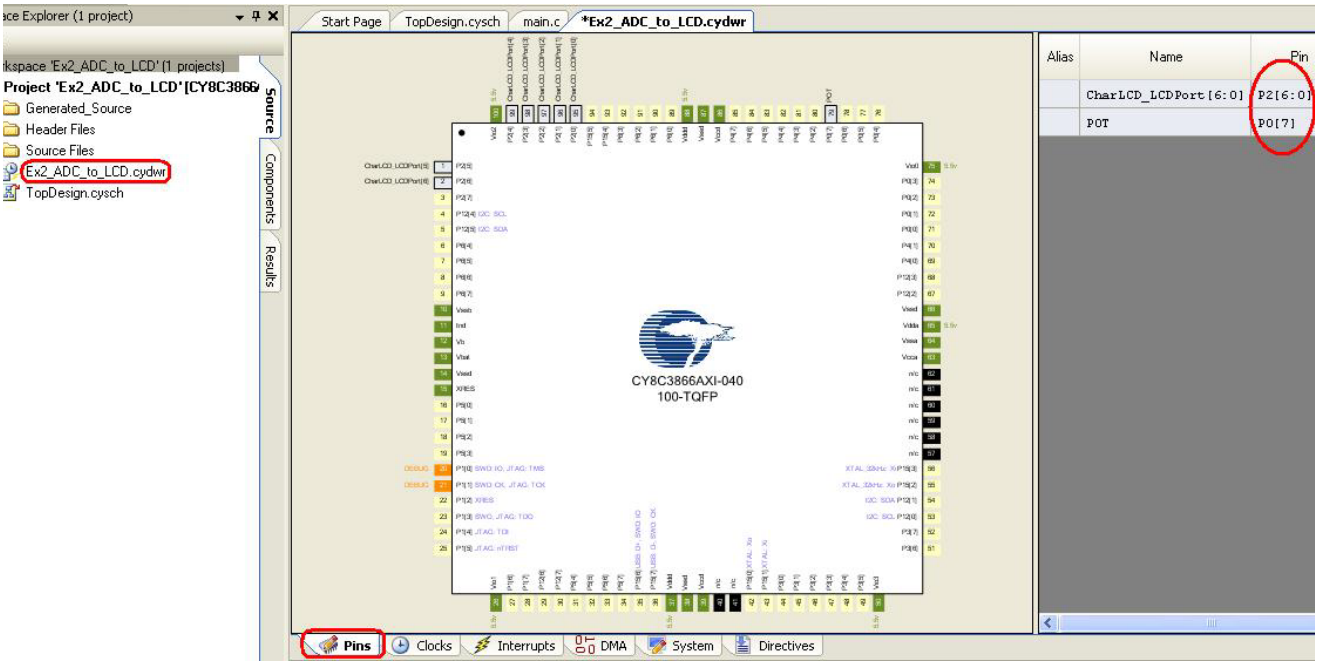
Figure 3-28. Connected Components



3.2.2.6 *Configuring the Pins*

1. From the **Workspace Explorer**, double click the *Ex2_ADC_to_LCD.cydwr* file.
2. Click the **Pins** tab.
3. Select pins P2[6:0] for CharLCD.
4. Select pin P0[7] for POT.

Figure 3-29. Assign Pins



3.2.2.7 *Creating the main.c File*

1. From the **Workspace Explorer** double click the *main.c* file
2. Use this code to replace the contents of the *main.c* file. (A soft copy of the *main.c* file is embedded in this PDF under “Attachments.”)

main.c

```
#include <device.h>

void UpdateDisplay(uint16 * voltageRawCount);
void TxHex (uint8 voltageRawCount);

/* Table of voltage values for DMA to send to the DAC. These values range
   between 0x3D and 0x9F because these are the two points where the LED
   is not visible and where the LED is saturated */
const uint8 voltageWave[] =
{
    0x3D, 0x3E, 0x3F,
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
0x4D, 0x4E, 0x4F,
    0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C,
0x5D, 0x5E, 0x5F,
    0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C,
0x6D, 0x6E, 0x6F,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x7B, 0x7C,
0x7D, 0x7E, 0x7F,
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8A, 0x8B, 0x8C,
0x8D, 0x8E, 0x8F,
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9A, 0x9B, 0x9C,
0x9D, 0x9E, 0x9F,
```

```
    0x9F, 0x9E, 0x9D, 0x9C, 0x9B, 0x9A, 0x99, 0x98, 0x97, 0x96, 0x95, 0x94, 0x93,
0x92, 0x91, 0x90,
    0x8F, 0x8E, 0x8D, 0x8C, 0x8B, 0x8A, 0x89, 0x88, 0x87, 0x86, 0x85, 0x84, 0x83,
0x82, 0x81, 0x80,
    0x7F, 0x7E, 0x7D, 0x7C, 0x7B, 0x7A, 0x79, 0x78, 0x77, 0x76, 0x75, 0x74, 0x73,
0x72, 0x71, 0x70,
    0x6F, 0x6E, 0x6D, 0x6C, 0x6B, 0x6A, 0x69, 0x68, 0x67, 0x66, 0x65, 0x64, 0x63,
0x62, 0x61, 0x60,
    0x5F, 0x5E,0x5D, 0x5C,0x5B, 0x5A, 0x59, 0x58, 0x57, 0x56, 0x55, 0x54, 0x53,
0x52, 0x51, 0x50,
    0x4F, 0x4E, 0x4D, 0x4C, 0x4B, 0x4A, 0x49, 0x48, 0x47, 0x46, 0x45, 0x44, 0x43,
0x42, 0x41, 0x40,
    0x3F, 0x3E, 0x3D
};

/*******************************************************************************
* Function Name: main
********************************************************************************
*
* Summary:
*   The main function initializes the ADC, LCD, VDAC, Analog Buffer, and UART.
*   It also initializes DMA by allocating/configuring a DMA channel and
*   Transaction Descriptor and also copies the voltage table address to the DAC
*   address. In the main loop, it starts and waits for an ADC conversion, then
*   it displays the ADC raw count to the LCD, transmits the raw count serially,
*   and sets the DMA clock divider proportional to the raw count.
*
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
void main()
{
    uint16 voltageRawCount;
    uint8 myChannel, myTd;

    ADC_Start();       /* Configure and power up ADC */
    CharLCD_Start();   /* Initialize and clear the LCD */
    DAC_Start();       /* Initializes VDAC8 with default values */
    OpAmp_Start();      /* Enable Analog Buffer and sets power level */
    UART_Start();      /* Enable UART */
    CyDmacConfigure(); /* Set DMA configuration register */

    /* Allocate and initialize a DMA channel to be used by the caller */
    myChannel = DMA_DmaInitialize(1,  /* Bursts are one byte each */
                                  1,  /* One request per burst */
                                  0,  /* Upper 16 bits of the source address are
zero */
                                  0); /* Upper 16 bits of the destination address
are zero */

    /* Allocate a Transaction Descriptor (TD) from the free list */
    myTd = CyDmaTdAllocate();

    CharLCD_Position(0,0); /* Move the LCD cursor to Row 0, Column 0 */
```

```
    /* Print Label for the pot voltage raw count */
    CharLCD_PrintString("V Count: ");

    CyDmaTdSetConfiguration(myTd, sizeof(voltageWave),
                            myTd, TD_INC_SRC_ADR ); /* Configure the TD */

    /* Copy address of voltageWave to address of DAC. Set the lower 16 bits of */
    /* the source and destination addresses for this TD */
    CyDmaTdSetAddress(myTd, (uint16) voltageWave, DAC_viDAC8__D);

    /* Associate TD with channel */
    CyDmaChSetInitialTd(myChannel, myTd);

    /* Enable DMA channel */
    CyDmaChEnable(myChannel, 1);

    /* Clock will make burst requests to the DMAC */
    DMAclock_Enable();

   ADC_StartConvert(); /* Force ADC to initiate a conversion */

    while(1)
    {
        ADC_IsEndConversion(ADC_WAIT_FOR_RESULT); /* Wait for end of conversion */
        voltageRawCount = ADC_GetResult16(); /* Get conversion result */
      /* ADC count errata workaround */
      if (voltageRawCount == -1)
      {
      voltageRawCount = 0;
      }
      else
      {
      voltageRawCount /= 2;
         }

      UpdateDisplay(&voltageRawCount); /* Print the result to LCD */

      TxHex(voltageRawCount); /* Transmit result to UART */

      /* The LED blinking frequency is dependant on the Voltage raw count. With
      a 3MHz clock the lowest divider (for raw count of 0) should be 1010
      to blink at a very slow pace. The highest value is about 44,390
      (193*230) to blink at a really fast pace */
      DMAclock_SetDivider(((voltageRawCount) * 193) + 1010);
    }
}

/*******************************************************************************
* Function Name: UpdateDisplay
********************************************************************************
*
* Summary:
*   Print voltage raw count result to the LCD.  Clears some characters if
*   necessary.  The voltageRawCount parameter is also updated for use in other
*   functions.
*
* Parameters:
```

```
*   voltageRawCount: Voltage raw count from ADC
*
* Return:
*   void
*
*******************************************************************************/
void UpdateDisplay (uint16 * voltageRawCount)
{
    CharLCD_Position(0,9); /* Move the cursor to Row 0, Column 9 */
    CharLCD_PrintNumber(voltageRawCount[0]); /* Print the result */

    if (voltageRawCount[0] < 10)
    {
        CharLCD_Position(0,10); /* Move the cursor to Row 0, Column 10 */
        CharLCD_PrintString("    "); /* Clear last characters */
    }
    else if (voltageRawCount[0] < 100)
    {
        CharLCD_Position(0,11); /* Move the cursor to Row 0, Column 11 */
        CharLCD_PrintString("   "); /* Clear last characters */
    }
}

/*******************************************************************************
* Function Name: TxHex
********************************************************************************
*
* Summary:
*   Convert voltage raw count to hex value and TX via UART.
*
* Parameters:
*   voltageRawCount: The voltage raw counts being received from the ADC
*
* Return:
*   void
*
*******************************************************************************/
void TxHex (uint8 voltageRawCount)
{
    static char8 const hex[16] = "0123456789ABCDEF";

    UART_PutChar(hex[voltageRawCount>>4]);  /* TX converted first nibble */
    UART_PutChar(hex[voltageRawCount&0x0F]); /* TX converted second nibble */
    UART_PutStringConst("h\r");  /* h for hexadecimal and carriage return */
}


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

3. From the **Build** menu, select **Build Ex2_ADC_to_LCD**. PSoC Creator builds the project and displays the comments in the **Output** dialog box. When you see the message “Build Succeeded” the build is complete.

3.2.2.8 Configuring and Programming the PSoC Development Board

1. Disconnect power to the board.
2. Configure the DVK bread board SW3 to 3.3V.

3. Using the jumper wires included, configure the PSoC Development Board's prototyping.
   - P0[7] to VR
4. Verify that VR_PWR (J11) is jumpered to ON.
5. Apply power to the board.
6. Use PSoC Creator as described in Programming a Device on page 15 to program the device.
7. After programming the device, press the **Reset** button on the PSoC Development Board to see the output of the ADC displayed on the LCD. Turning the potentiometer results in the LCD value changing.
8. Save and close the project.

3.2.3 ADC to UART with DAC

This project demonstrates sine wave generation by using an 8-bit DAC and DMA. The sine wave period is based on the current value of the ADC value of the potentiometer.

The firmware reads the voltage output by the DVK board potentiometer and displays the raw counts on the DVK board character LCD display similarly to that shown in the previous project. An 8-bit DAC outputs a table generated sine wave to an LED using DMA at a frequency proportional to the ADC count. A 9600 BAUD 8N1 UART outputs the current ADC count as ASCII formatted into a hexadecimal number.

The instructions below assume that the user has completed My First PSoC Project and ADC to LCD Project and therefore has a basic understanding of the PSoC Creator software environment.

This example project uses the following components:

- Delta Sigma ADC (**Component Catalog** → **Analog** → **ADC** → **Delta Sigma ADC**)
- Voltage DAC (**Component Catalog** → **Analog** → **DAC** → **Voltage DAC**)
- Analog Buffer (**Component Catalog** → **Analog** → **Amplifiers** → **Analog Buffer**)
- DMA (**Component Catalog** → **System** → **DMA**)
- Character LCD (**Component Catalog** → **Display** → **Character LCD**)
- UART (**Component Catalog** → **Communications** → **UART**)
- Analog Port (**Component Catalog** → **System** → **Analog Port**)
- Digital Port (**Component Catalog** → **System** → **Digital Port**)
- Clock
- Logic Low
- Logic High

3.2.3.1 Creating ADC to UART with DAC Project

1. Open PSoC Creator.
2. Create a new project by clicking **Create New Project…** in the **Start Page** of PSoC Creator.
3. In the **New Project** window, select the **Empty PSoC3 Design** template and **Name** the project **Ex3_ADC_to_UART_with_DAC**.
4. In **Location**, type the path where you want to save the project, or click 

and navigate to the appropriate directory.
5. By default, the design window opens *TopDesign.cysch*. This is the project's schematic entry file within PSoC Creator.

3.2.3.2 *Placing and Configuring Delta Sigma ADC*

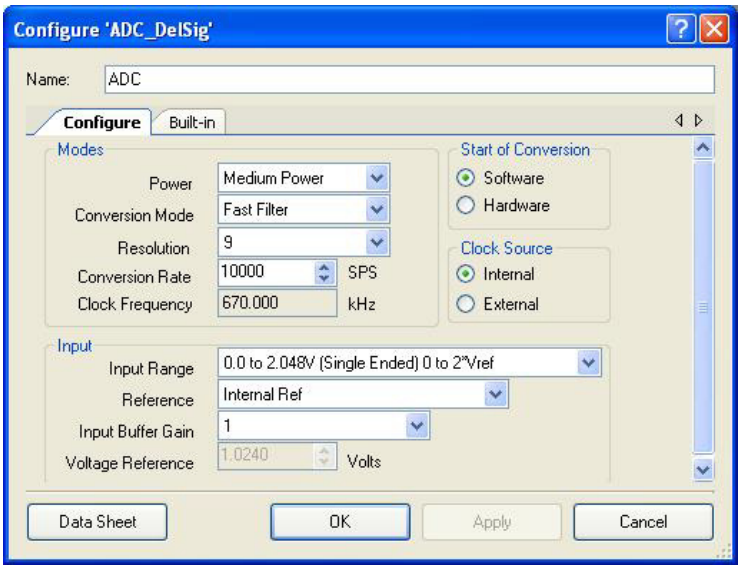
1. Drag-and-drop the Delta Sigma ADC component (**Component Catalog** → **Analog** → **ADC** → **Delta Sigma ADC**)
2. Double click the **ADC_DelSig_1** component in the schematic to open the configuration window.
3. Configure the digital port in this manner:

   Configure Tab

   - **Name**: ADC
   - **Power**: Medium Power
   - **Conversion Mode**: Fast Filter
   - **Resolution**: 9
   - **Conversion Rate**: 10000
   - **Input Range**: 0 to 2.048V (Single Ended) 0 to 2*Vref
   - **Input Buffer Gain**: 1
   - **Reference**: Internal Ref
   - **Clock Source**: Internal
   - **Start of Conversion**: Software

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-30. Delta Sigma ADC Component Configuration



3.2.3.3 *Placing and Configuring Analog Port*

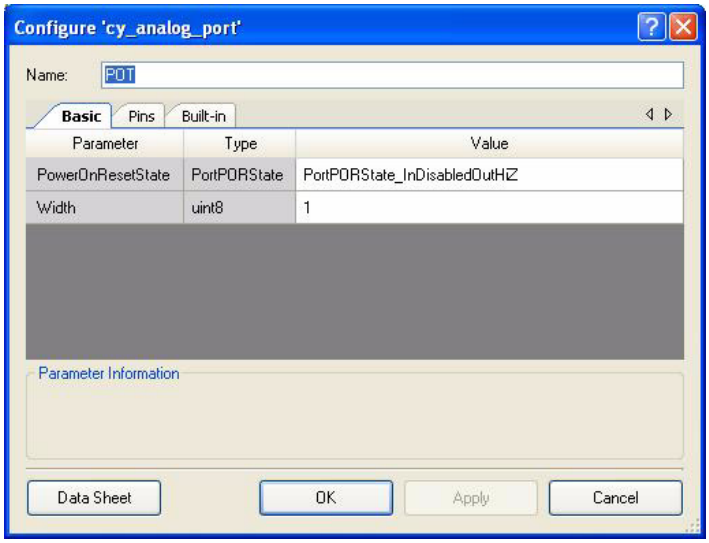
1. Drag-and-drop the Analog Port component (**Component Catalog** → **System** → **Analog Port**)
2. Double click the **aPort_1** component in the schematic to open the configuration window.
3. Configure the digital port:

   Basic Tab

   - **Name**: POT
   - **Width**: 1

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-31. POT Component Configuration

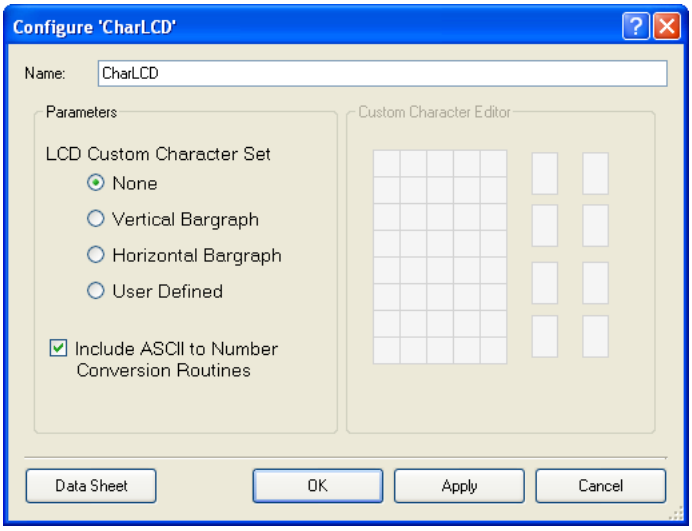


3.2.3.4 *Placing and Configuring Character LCD*

1. Drag-and-drop the Character LCD component (**Component Catalog** → **Display** → **Character LCD**)
2. Double click the **LCD_Char_1** component in the schematic to open the configuration window.
3. Configure the digital port:
   - **Name**: CharLCD
   - **LCD Custom Character Set**: None
   - **Include ASCII to Number Conversion Routines**: Check box

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-32. Character LCD Component Configuration



3.2.3.5 *Placing and Configuring Voltage DAC*

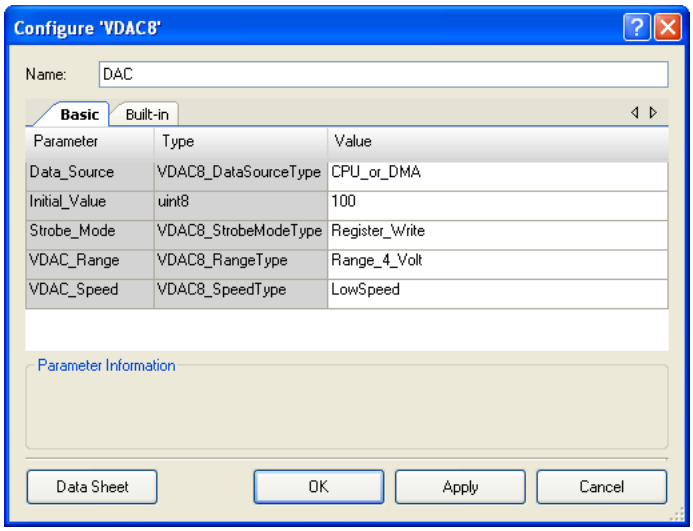
1. Drag-and-drop the Voltage DAC component (**Component Catalog** → **Analog** → **DAC** → **Voltage DAC**)
2. Double click the **VDAC8_1** component in the schematic to open the configuration window.
3. Configure the digital port:

   Basic Tab

   - **Name**: DAC
   - **Data_Source**: CPU_or_DMA
   - **Initial_Value**: 100
   - **Strobe_Mode**: Register_Write
   - **VDAC_Range**: Range_4_Volt
   - **VDAC_Speed**: LowSpeed

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-33. Voltage DAC Component Configuration



3.2.3.6 *Placing and Configuring Opamp*

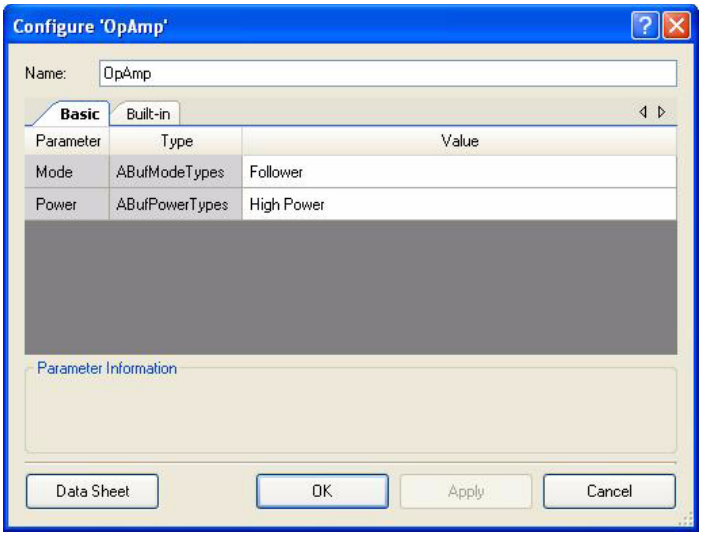
1. Drag-and-drop the Opamp component (**Component Catalog** → **Analog** → **Amplifiers** → **Opamp**)
2. Double click the **Opamp_1** component in the schematic to open the configuration window.
3. Configure the digital port:

   Basic Tab

   - **Name**: Opamp
   - **Mode**: Follower
   - **Power**: High Power

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-34. Opamp Component Configuration



3.2.3.7 *Placing and Configuring Analog Port*

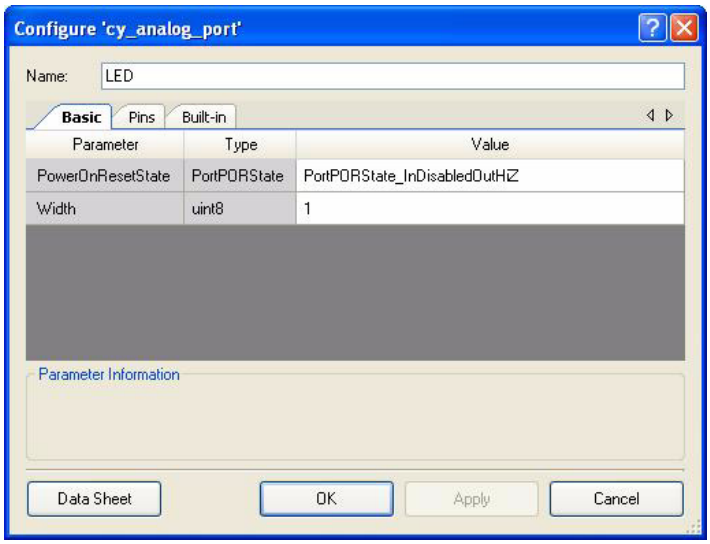
1. Drag-and-drop the Analog Port component (**Component Catalog** → **System** → **Analog Port**)
2. Double click the **aPort_1** component in the schematic to open the configuration window.
3. Configure the digital port:

   Basic Tab

   - **Name**: LED
   - **Width**: 1

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-35. LED Component Configuration



3.2.3.8 *Placing and Configuring UART*

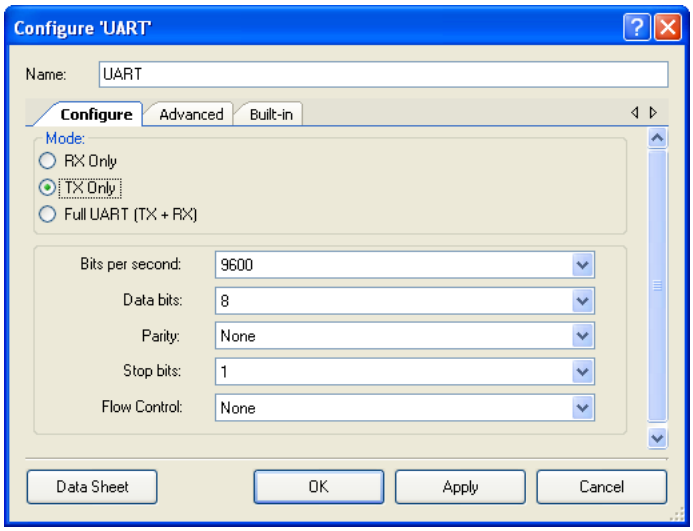
1. Drag-and-drop the UART component (**Component Catalog** → **Communications** → **UART**)
2. Double click the **UART_1** component in the schematic to open the configuration window.
3. Configure the digital port:

   Configure Tab

   - **Name**: UART
   - **Mode**: TxOnly
   - **Bits per second**: 9600
   - Leave the remaining parameters to default

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-36. UART Component Configuration



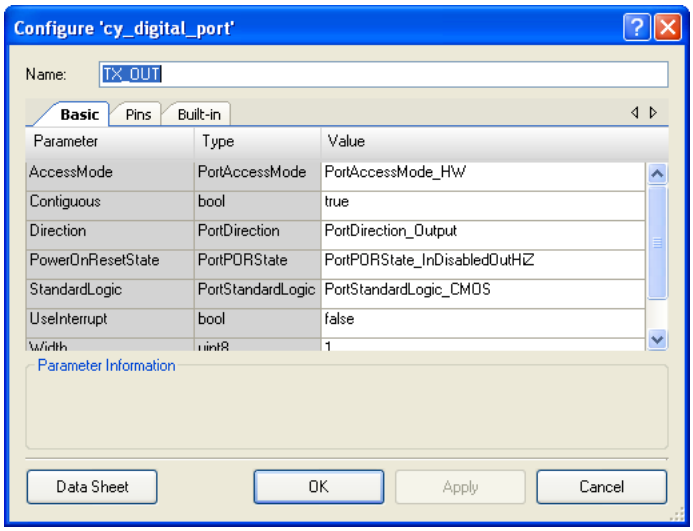
3.2.3.9 *Placing and Configuring Digital Port*

1. Drag-and-drop the Digital Port component (**Component Catalog** → **System** → **Digital Port**)
2. Double click the **dPort_1** component in the schematic to open the configuration window.
3. Configure the digital port:

   Basic Tab

   - **Name**: TX_OUT
   - **AccessMode**: PortAccessMode_HW
   - **Direction**: PortDirection:_Output
   - **Width**: 1

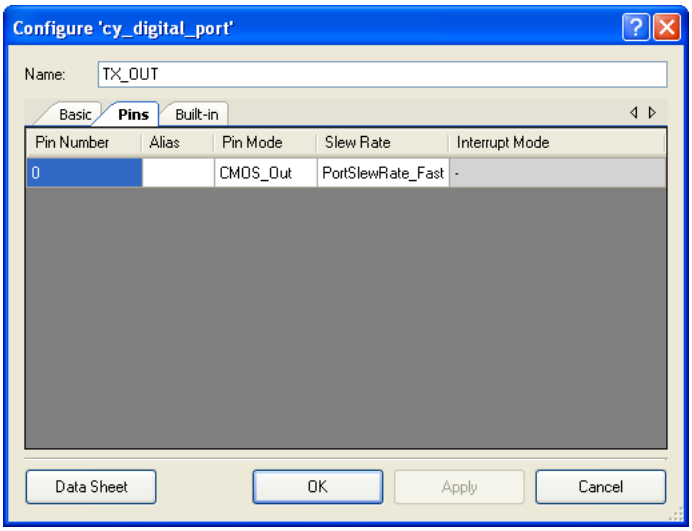
Figure 3-37. TX_OUT Component Configuration



Pins Tab

- **Pin Mode**: CMOS_Out

Figure 3-38. Pins - TX_OUT Component Configuration



For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

3.2.3.10 *Placing and Configuring DMA*

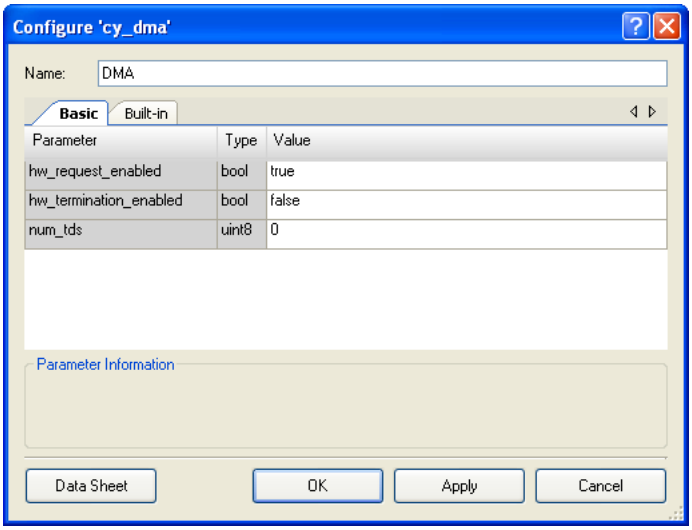
1. Drag-and-drop the DMA component (**Component Catalog** → **System** → **DMA**)
2. Double click the **DMA_1** component in the schematic to open the configuration window.
3. Configure the digital port:

   Basic Tab

   - **Name**: DMA
   - **Hw_request_enabled**: true
   - Leave the remaining parameters to default

   For more information about what the parameters mean, click the **Data Sheet** button in the configuration window.

Figure 3-39. DMA Component Configuration



3.2.3.11 *Connecting the Components Together*

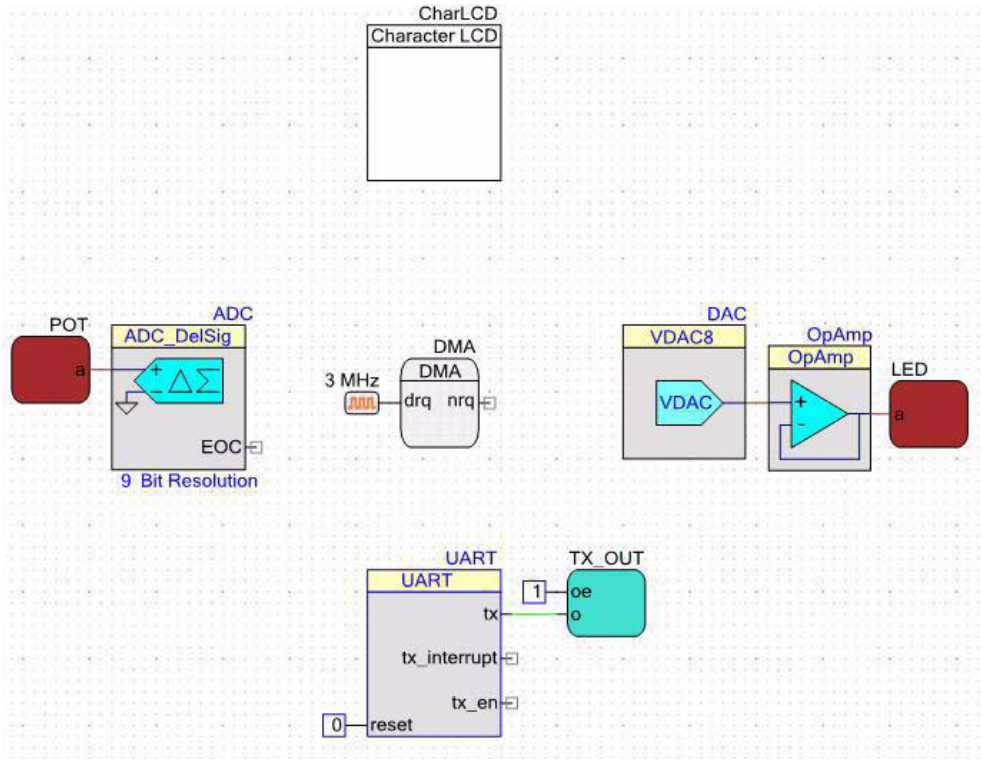
1. Connect a Logic High component (**Component Catalog** → **Digital Logic** → **Logic High**) to the **oe** on TX_OUT
2. Connect a Logic Low component (**Component Catalog** → **Digital Logic** → **Logic Low**) to the **reset** on the UART
3. Connect a Clock component (**Component Catalog** → **System** → **Clock**) to the **clock** on the DMA.
4. Double click the **Clock** component to configure.
5. Configure the clock:

   Configure Clock Tab
   - **Name**: DMAclock
   - **Source**: IMO (3.000 MHz)
   - **Desired Frequency**: 3 MHz
   - Leave remaining parameters set to their default values

Figure 3-40. Clock Component Configuration

6. Using the Wire Tool, connect **tx** (in the UART component) to **o** on the digital port (TX_OUT).
   - **Pin Mode**: CMOS_Out
7. Right click the LED analog port, select the **Shape** menu option and then **Flip Horizontal**. This allows the LED pin to line up with the analog buffer.
8. When complete the schematic looks like Figure 3-41.

Figure 3-41. Connected Components



3.2.3.12 *Configuring the Pins*

1. From the **Workspace Explorer**, open the *Ex3_ADC_to_UART_with_DAC.cydwr* file.
2. Click the **Pins** tab.
3. Select pins P2[6:0] for CharLCD
4. Select pin P0[7] for POT
5. Select pin P1[6] for LED
6. Select pin P1[2] for TX_OUT

Figure 3-42. Pin Assignments

Alias	Name	Pin
	POT	P0[7]
	LED	P1[6]
	TX_OUT	P1[2]
	CharLCD_LCDPort[6:0]	P2[6:0]

3.2.3.13 Creating the main.c File

1. From the Workspace Explorer double click the *main.c* file.
2. Use this code to replace the contents of the *main.c* file. (A soft copy of the *main.c* file is embedded in this PDF under “Attachments.”)

main.c

```
#include <device.h>

void UpdateDisplay(uint16 * voltageRawCount);
void TxHex (uint8 voltageRawCount);

/* Table of voltage values for DMA to send to the DAC. These values range
   between 0x3D and 0x9F because these are the two points where the LED
   is not visible and where the LED is saturated */
const uint8 voltageWave[] =
{
    0x3D, 0x3E, 0x3F,
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C,
0x4D, 0x4E, 0x4F,
    0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C,
0x5D, 0x5E, 0x5F,
    0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C,
0x6D, 0x6E, 0x6F,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x7B, 0x7C,
0x7D, 0x7E, 0x7F,
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8A, 0x8B, 0x8C,
0x8D, 0x8E, 0x8F,
    0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9A, 0x9B, 0x9C,
0x9D, 0x9E, 0x9F,

    0x9F, 0x9E, 0x9D, 0x9C, 0x9B, 0x9A, 0x99, 0x98, 0x97, 0x96, 0x95, 0x94, 0x93,
0x92, 0x91, 0x90,
    0x8F, 0x8E, 0x8D, 0x8C, 0x8B, 0x8A, 0x89, 0x88, 0x87, 0x86, 0x85, 0x84, 0x83,
0x82, 0x81, 0x80,
    0x7F, 0x7E, 0x7D, 0x7C, 0x7B, 0x7A, 0x79, 0x78, 0x77, 0x76, 0x75, 0x74, 0x73,
0x72, 0x71, 0x70,
    0x6F, 0x6E, 0x6D, 0x6C, 0x6B, 0x6A, 0x69, 0x68, 0x67, 0x66, 0x65, 0x64, 0x63,
0x62, 0x61, 0x60,
    0x5F, 0x5E,0x5D, 0x5C,0x5B, 0x5A, 0x59, 0x58, 0x57, 0x56, 0x55, 0x54, 0x53,
0x52, 0x51, 0x50,
    0x4F, 0x4E, 0x4D, 0x4C, 0x4B, 0x4A, 0x49, 0x48, 0x47, 0x46, 0x45, 0x44, 0x43,
0x42, 0x41, 0x40,
    0x3F, 0x3E, 0x3D
};

/*******************************************************************************
* Function Name: main
********************************************************************************
*
* Summary:
*   The main function initializes the ADC, LCD, VDAC, Analog Buffer, and UART.
*   It also initializes DMA by allocating/configuring a DMA channel and
*   Transaction Descriptor and also copies the voltage table address to the DAC
*   address. In the main loop, it starts and waits for an ADC conversion, then
*   it displays the ADC raw count to the LCD, transmits the raw count serially,
*   and sets the DMA clock divider proportional to the raw count.
*
```

```
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
void main()
{
    uint16 voltageRawCount;
    uint8 myChannel, myTd;

    ADC_Start();       /* Configure and power up ADC */
    CharLCD_Start();   /* Initialize and clear the LCD */
    DAC_Start();       /* Initializes VDAC8 with default values */
    OpAmp_Start();      /* Enable Analog Buffer and sets power level */
    UART_Start();      /* Enable UART */
    CyDmacConfigure(); /* Set DMA configuration register */

    /* Allocate and initialize a DMA channel to be used by the caller */
    myChannel = DMA_DmaInitialize(1,  /* Bursts are one byte each */
                                  1,  /* One request per burst */
                                  0,  /* Upper 16 bits of the source address are
zero */
                                  0); /* Upper 16 bits of the destination address
are zero */

    /* Allocate a Transaction Descriptor (TD) from the free list */
    myTd = CyDmaTdAllocate();

    CharLCD_Position(0,0); /* Move the LCD cursor to Row 0, Column 0 */

    /* Print Label for the pot voltage raw count */
    CharLCD_PrintString("V Count: ");

    CyDmaTdSetConfiguration(myTd, sizeof(voltageWave),
                            myTd, TD_INC_SRC_ADR ); /* Configure the TD */

    /* Copy address of voltageWave to address of DAC. Set the lower 16 bits of */
    /* the source and destination addresses for this TD */
    CyDmaTdSetAddress(myTd, (uint16) voltageWave, DAC_viDAC8__D);

    /* Associate TD with channel */
    CyDmaChSetInitialTd(myChannel, myTd);

    /* Enable DMA channel */
    CyDmaChEnable(myChannel, 1);

    /* Clock will make burst requests to the DMAC */
    DMAclock_Enable();

   ADC_StartConvert(); /* Force ADC to initiate a conversion */

    while(1)
    {
        ADC_IsEndConversion(ADC_WAIT_FOR_RESULT); /* Wait for end of conversion */
        voltageRawCount = ADC_GetResult16(); /* Get conversion result */
      /* ADC count errata workaround */
```

```
        if (voltageRawCount == -1)
        {
           voltageRawCount = 0;
        }
        else
        {
           voltageRawCount /= 2;
          }

        UpdateDisplay(&voltageRawCount); /* Print the result to LCD */

        TxHex(voltageRawCount); /* Transmit result to UART */

        /* The LED blinking frequency is dependant on the Voltage raw count. With
           a 3MHz clock the lowest divider (for raw count of 0) should be 1010
           to blink at a very slow pace. The highest value is about 44,390
           (193*230) to blink at a really fast pace */
        DMAclock_SetDivider(((voltageRawCount) * 193) + 1010);
    }
}

/*******************************************************************************
* Function Name: UpdateDisplay
********************************************************************************
*
* Summary:
*   Print voltage raw count result to the LCD.  Clears some characters if
*   necessary.  The voltageRawCount parameter is also updated for use in other
*   functions.
*
* Parameters:
*   voltageRawCount: Voltage raw count from ADC
*
* Return:
*   void
*
*******************************************************************************/
void UpdateDisplay (uint16 * voltageRawCount)
{
CharLCD_Position(0,9); /* Move the cursor to Row 0, Column 9 */
    CharLCD_PrintNumber(voltageRawCount[0]); /* Print the result */

    if (voltageRawCount[0] < 10)
    {
        CharLCD_Position(0,10); /* Move the cursor to Row 0, Column 10 */
        CharLCD_PrintString("    "); /* Clear last characters */
    }
    else if (voltageRawCount[0] < 100)
    {
        CharLCD_Position(0,11); /* Move the cursor to Row 0, Column 11 */
        CharLCD_PrintString("   "); /* Clear last characters */
    }
}

/*******************************************************************************
* Function Name: TxHex
********************************************************************************
*
```

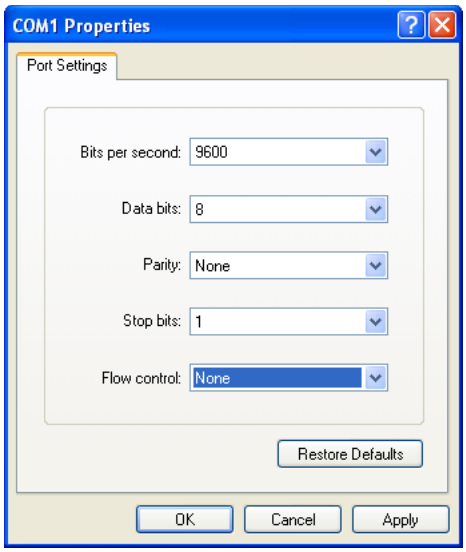
```
* Summary:
*   Convert voltage raw count to hex value and TX via UART.
*
* Parameters:
*   voltageRawCount: The voltage raw counts being received from the ADC
*
* Return:
*   void
*
*******************************************************************************/
void TxHex (uint8 voltageRawCount)
{
    static char8 const hex[16] = "0123456789ABCDEF";

    UART_PutChar(hex[voltageRawCount>>4]);  /* TX converted first nibble */
    UART_PutChar(hex[voltageRawCount&0x0F]); /* TX converted second nibble */
    UART_PutStringConst("h\r");  /* h for hexadecimal and carriage return */
}


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

3. From the **Build** menu, select **Build Ex3_ADC_to_UART_with_DAC**. PSoC Creator builds the project and displays the comments in the **Output** dialog box. When you see the message “Build Succeeded” the build is complete.

3.2.3.14 Configuring and Programming the PSoC Development Board

1. Disconnect power to the board.
2. Configure the DVK bread board SW3 to 3.3V.
3. Using the jumper wires included, configure the PSoC Development Board's prototyping area to:
   - P0[7] to VR
   - P1[2] to TX
   - P1[6] to LED1
4. Verify that VR_PWR (J11) is jumpered to ON.
5. Verify that RS232_PWR (J10) is jumpered to ON.
6. Connect a serial cable from the PSoC Development Board to a PC.
7. Apply power to the board.
8. Install a terminal application such as TeraTerm or HyperTerminal with these setup parameters:
   - **Baud Rate**: 9600
   - **Data**: 8-bit
   - **Parity**: none
   - **Stop**: 1-bit
   - **Flow Control**: none

Figure 3-43. HyperTerminal Settings



9. Use PSoC Creator as described in Programming a Device on page 15 to program the device.
10. After programming the device, press the **Reset** button on the PSoC Development Board to see the output of the ADC displayed on the LCD and in the terminal application. LED1 is a sine wave output whose period is based on the ADC. Turning the potentiometer results in the LCD and observed terminal value change.
11. Save and close the project.

3.2.4 USB HID

This project demonstrates a simple HID keyboard. The firmware begins enabling global interrupts, setting up the button (SW), and initializing USB for 3V operation. The firmware, after allowing the HID device to enumerate, continuously checks for a button press to see if it needs to send the keyboard key sequences for the Cypress website.

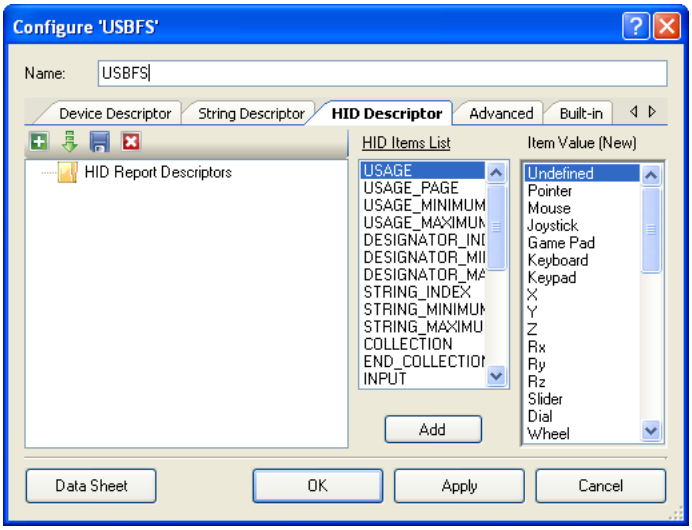
3.2.4.1 Creating USB HID Project

1. Open PSoC Creator.
2. Create a new project by clicking on **Create New Project…** in the **Start Page** of PSoC Creator.
3. In the **New Project** window, select the **Empty PSoC3 Design** template and **Name** the project **Ex4_USB_HID**.
4. In **Location**, type the path where you want to save the project, or click ... and navigate to the appropriate directory.
5. By default, the design window opens *TopDesign.cysch*. This is the project's schematic entry file within PSoC Creator.

3.2.4.2 Placing and Configuring USBFS

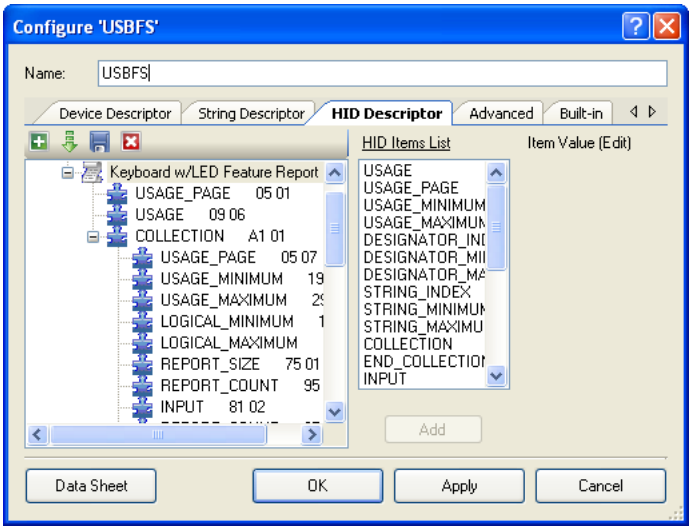
1. Drag-and-drop a USBFS component from the **Components Catalog** to the workspace.
2. Double click the **USBFS_1** component.
3. Name the component **USBFS**.
4. Select the **HID Descriptor** tab.

Figure 3-44. USBFS Component Configuration



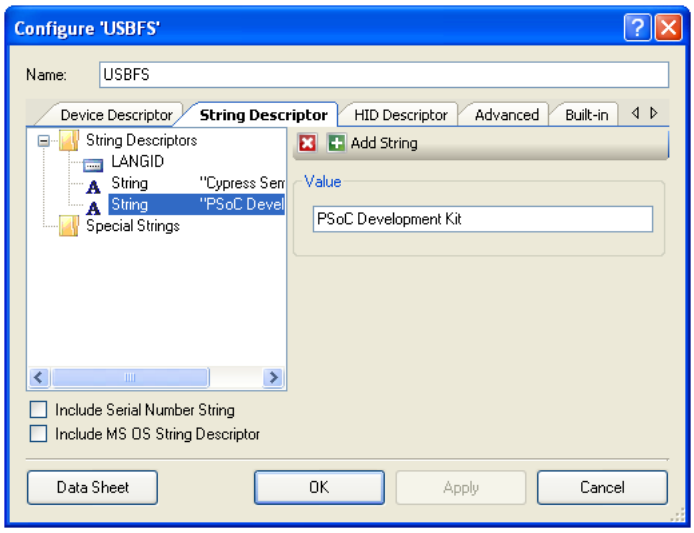
5. Click to import a report.
6. Browse the **Kit CD** and open the XML descriptor file, *USB_HID_Example.xml*.

Figure 3-45. HID Descriptor Configuration



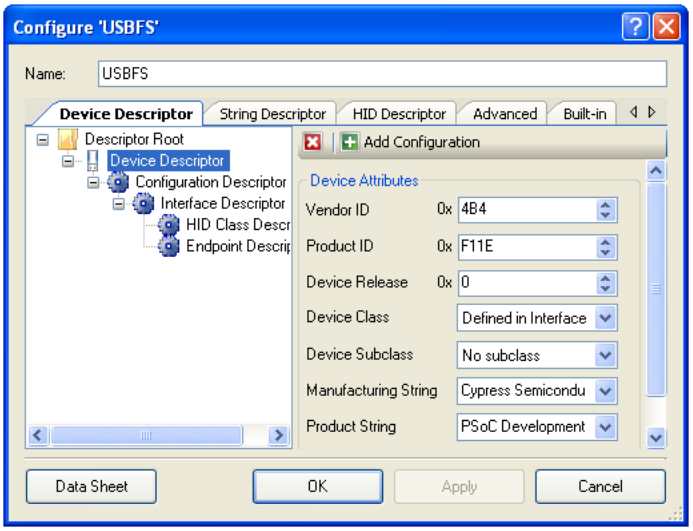
7. Select the **String Descriptor** tab.

Figure 3-46. String Descriptor Configuration



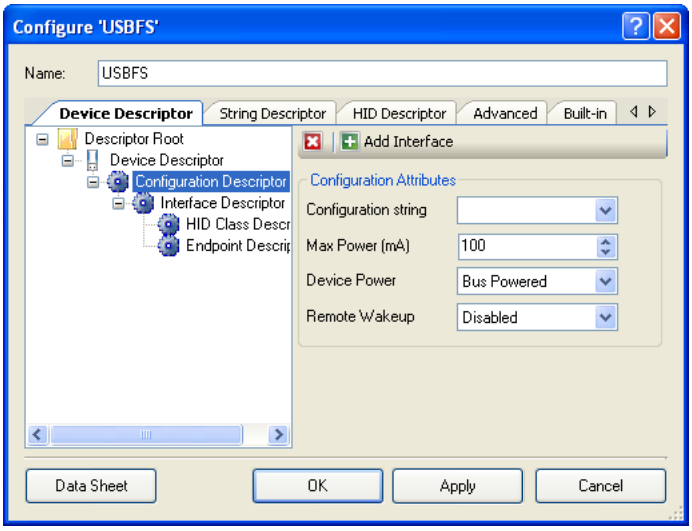
8. Select **String Descriptors** in the left window.
9. Click **Add String**.
10. Click **Add String** a second time to add a total of two strings.
11. Click the **String** that shows up at the top in the left window.
12. Type **Cypress Semiconductor** in the **Value** field.
13. Click the **String** that shows up at the bottom in the left window.
14. Type **PSoC Development Kit** in the **Value** field.
15. Select the **Device Descriptor** tab.
16. Select **Device Descriptor**
17. Set the **Product ID** to **F11E**.
18. Set the **Manufacturing String** to **Cypress Semiconductor**.
19. Set the **Product String** to **PSoC Development Kit**.

Figure 3-47. Device Descriptor Configuration



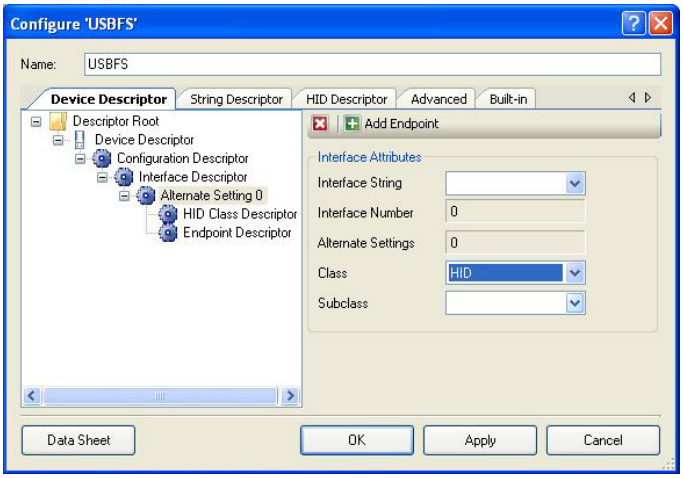
20. Select **Configuration Descriptor**.
21. Set **Max Power** to **100 mA**.

Figure 3-48. Configuration Descriptor Configuration



22. Select **Alternate Setting 0**
23. Set **Class** to **HID**.

Figure 3-49. Interface Descriptor Configuration



24. Select **HID Class Descriptor**.
25. Set **HID Report** to **Keyboard w/LED Feature Report**.
26. Select **Endpoint Descriptor**.
27. Set **Direction** to **IN** and **Transfer Type** to **INT**.

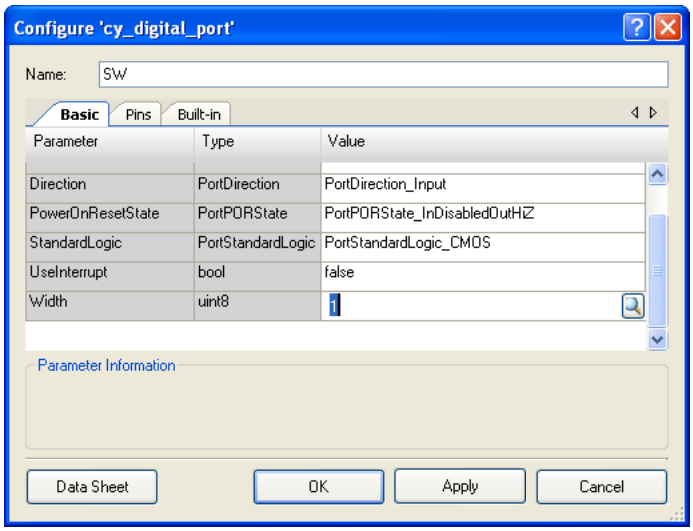
3.2.4.3 *Placing and Configuring Software Digital Port*

1. Drag-and-drop a Digital Port component
2. Configure in this manner.

   Basic Tab

   - **Name**: SW
   - **Width**: 1

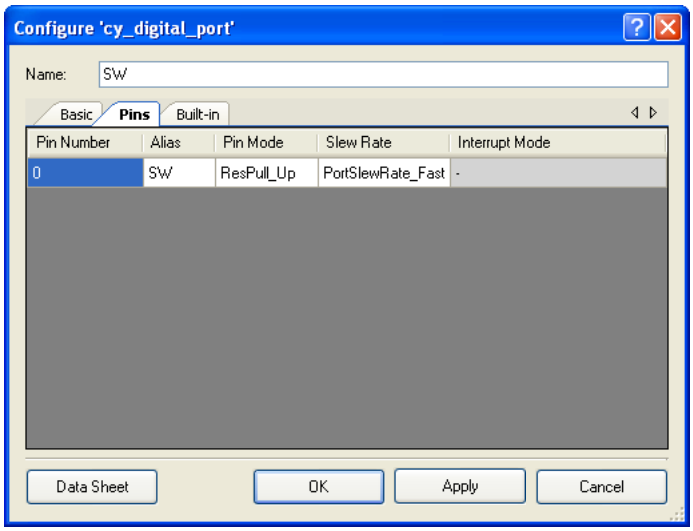
Figure 3-50. SW Digital Port Configuration



Pins Tab

- **Pin Mode**: ResPull_Up.

Figure 3-51. Pins - SW Digital Port Configuration



3. Click **OK**.

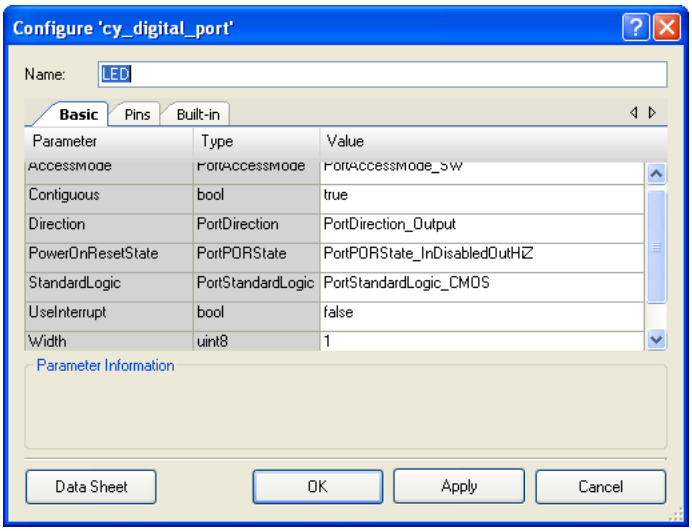
3.2.4.4 *Placing and Configuring LED*

1. Drag-and-drop a **Digital Port**
2. Configure in this manner:

   Basic Tab

   - **Name**: LED
   - **Direction**: PortDirection_Output
   - **Width**: 1

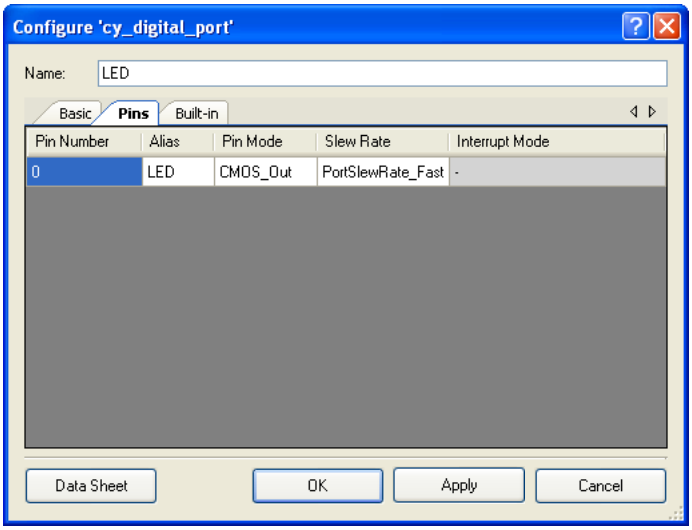
Figure 3-52. LED Component Configuration



Pins Tab

- **Pin Mode**: CMOS_Out

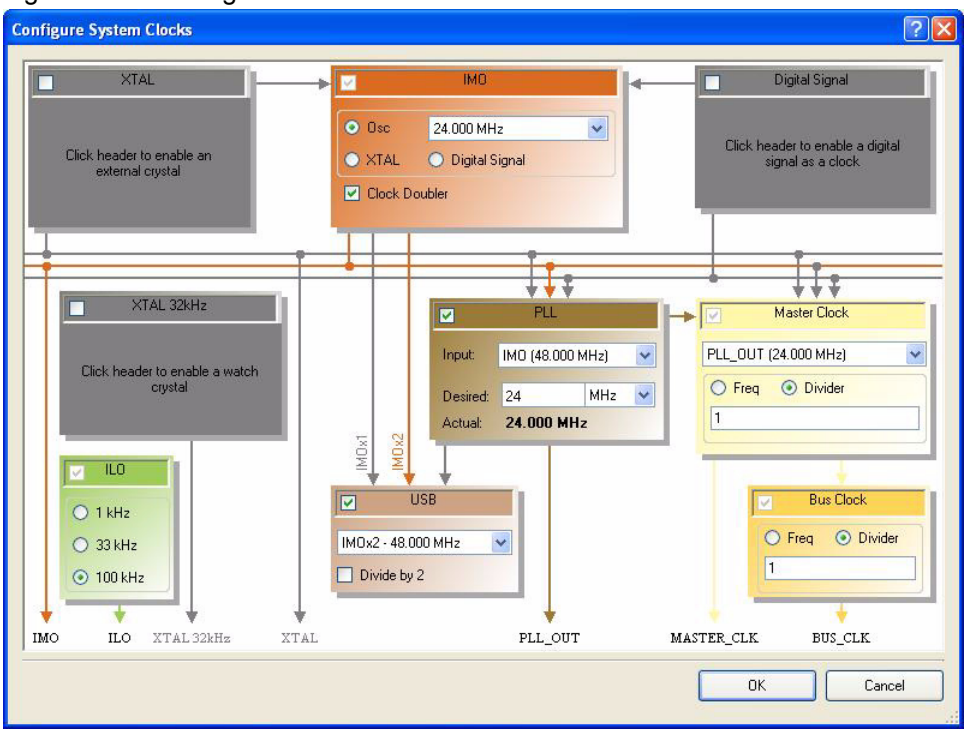
Figure 3-53. Pins - LED Component Configuration



3. From the **Workspace Explorer**, open the *Ex4_USB_HID.cydwr* window and select the **Clocks** tab.

4. Click the **IMO** block and set:
   - Osc: 24.000 MHz
5. Select **Clock Doubler**.
6. Click the **Master Clock** block and select **IMO (48.000 MHz)** for the master clock.
7. Enable the USB clock.
8. Set the ILO clock to 100 kHz

Figure 3-54. Configure the Builtin Clocks



9. Click **OK**.

3.2.4.5 *Configuring the Pins*

1. Click the **Pins** tab.
   - Assign USBFS_dp to P15[6]
   - Assign USBFS_dm to P15[7]
   - Assign LED to P1[6]
   - Assign SW to P1[5]

Figure 3-55. Pin Assignments

dp	USBFS_dp	P15[6]
dm	USBFS_dm	P15[7]
	LED	P1[6]
	SW	P1[5]

3.2.4.6 Creating the main.c File

1. From the **Workspace Explorer** double click the *main.c* file.
2. Use this code to replace the contents of the *main.c* file. (A soft copy of the *main.c* file is embedded in this PDF under “Attachments.”)

main.c

```
#include “device.h”

/* Keyboard scan codes */
#define WINDOWS_LEFT_MODIFIER 0x80u /* This is the left windows key */
#define LETTER_R              0x15u
#define CARRIAGE_RETURN       0x28u
#define KEY_RELEASE           0x00u

/* For button setup */
#define SET_SW      0x01u

/* USB related */
#define KEYBOARD_ENDPOINT       0x01u
#define KEYBOARD_DEVICE         0x00u
#define KEYBOARD_DATA_SIZE      0x08u
#define KEY_DATA_INDEX          0x02u
#define MODIFIER_KEY_DATA_INDEX 0x00u

static uint8 ButtonPressed(void);
static void  SendKey(uint8 key);
static void  StartWindowsRun(void); /* Open Windows Run App */
static void  GetAckLoadEp(uint8 * keyboardData);

/*******************************************************************************
* Function Name: main
********************************************************************************
*
* Summary:
*  The main function starts out enabling global interrupts, setting up the
*  button (SW1), then initializing USB for 3V operation. Then allows the HID
*  device to enumerate, and loads the keyboard endpoint to allow an ack to be
*  sent before sending the first keyboard data. Then it continuously checks for
*  USB plug-and-play and a button press to see if the keyboard data needs sent.
*
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
void main()
{
    uint8 ledState = 0; /* Set initial LED state to off */
uint8 i;

    /* Data array for the keyboard device endpoint */
    uint8 keyboardData[] = {0,0,0,0,0,0,0,0};

/* Keyboard scan codes for the cypress website (“www.cypress.com<cr>”) */
    uint8 cypressWebsiteCharSequence[] = { 0x1A, 0x1A, 0x1A, 0x37, 0x06, 0x1C,
```

```
                                                0x13, 0x15, 0x08, 0x16, 0x16, 0x37,
                                                0x06, 0x12, 0x10, 0x28};

    CYGlobalIntEnable; /* Enable global interrupts */

    SW_Write(SET_SW);  /* Set port pin for button (SW1) */

       /* Start USBFS operation using keyboard device (0) and with 3V operation */
    USBFS_Start(KEYBOARD_DEVICE, USBFS_3V_OPERATION);

    while(!USBFS_bGetConfiguration()); /* Wait for Device to enumerate */

    /* Enumeration is completed load keyboard endpoint to set up ACK for first
       key */
    USBFS_LoadInEP(KEYBOARD_ENDPOINT, keyboardData, KEYBOARD_DATA_SIZE);

    while (1)
    {
        if (ButtonPressed())
        {
            ledState ^= 0x01u;    /* Set/clear (toggle) the LED port pin */
            LED_Write(ledState);  /* Toggle LED */

                StartWindowsRun(); /* Open Windows Run App */

                /* Send the cypress website key sequence */
                for (i = 0; i < sizeof(cypressWebsiteCharSequence); i++)
                {
             SendKey(cypressWebsiteCharSequence[i]);
                }
        }
    }
}

/*******************************************************************************
* Function Name: ButtonPressed
********************************************************************************
*
* Summary:
*  Check to see if the button is pressed.
*
* Parameters:
*  void
*
* Return:
*  TRUE: button pressed
*  FALSE: button not pressed
*
*******************************************************************************/
static uint8 ButtonPressed(void)
{
    if(!SW_Read())
    {
        /* wait for the button to be released */
        while(!SW_Read());
        return TRUE;
    }
    return FALSE;
```

```
}

/*******************************************************************************
* Function Name: SendKey
********************************************************************************
*
* Summary:
*  Sends keyboard key.
*
* Parameters:
*  key: Keyboard scan code.
*
* Return:
*  void
*
*******************************************************************************/
static void SendKey(uint8 key)
{
    /* Data array for the keyboard device endpoint */
    uint8 keyboardData[] = {0,0,0,0,0,0,0,0};

    keyboardData[KEY_DATA_INDEX] = key; /* Send key-down data, make */
       GetAckLoadEp(keyboardData); /* Send USB keyboard data */

       keyboardData[KEY_DATA_INDEX] = KEY_RELEASE; /* Send key-up data, break */
    GetAckLoadEp(keyboardData); /* Send USB keyboard data */
}

/*******************************************************************************
* Function Name: StartWindowsRun
********************************************************************************
*
* Summary:
*  Sends the Windows Run command by sending Windows Modifier and 'r' (while
*  the Windows modifier key is in a down state) and then releasing both keys.
*
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
static void StartWindowsRun(void)
{
   /* Data array for the keyboard device endpoint */
   uint8 keyboardData[] = {0,0,0,0,0,0,0,0};

   /* Send Windows modifier key-down data, modifier make */
   keyboardData[MODIFIER_KEY_DATA_INDEX] = WINDOWS_LEFT_MODIFIER;
        GetAckLoadEp(keyboardData); /* Send USB keyboard data */

        /* While Windows modifier key is down send r key, r make */
   keyboardData[KEY_DATA_INDEX] = LETTER_R;
        GetAckLoadEp(keyboardData); /* Send USB keyboard data */

        /* Send up keys for both Windows modifier key and r key */
   keyboardData[KEY_DATA_INDEX] = KEY_RELEASE; /* r break */
```

```
   keyboardData[MODIFIER_KEY_DATA_INDEX] = KEY_RELEASE; /* Windows modifier break
*/
   GetAckLoadEp(keyboardData); /* Send USB keyboard data */
        CyDelay (500); /* Delay about 0.5 seconds to allow Run window to pop-up */
}

/*******************************************************************************
* Function Name: GetAckLoadEp
********************************************************************************
*
* Summary:
*  It first confirms that an Acknowledge transaction occurred on the keyboard
*  endpoint. Once the ACK is confirmed the endpoint and enabled and loaded.
*
* Parameters:
*  keyboardData: Data array for the keyboard device endpoint
*
* Return:
*  void
*
*******************************************************************************/
static void GetAckLoadEp(uint8 * keyboardData)
{
/* Wait for ACK before loading data */
    while(!USBFS_bGetEPAckState(KEYBOARD_ENDPOINT));
    /* ACK has occurred, load the endpoint */
    USBFS_LoadInEP(KEYBOARD_ENDPOINT, keyboardData, KEYBOARD_DATA_SIZE);
}
/* [] END OF FILE */
```

3. From the **Build** menu, select **Build Ex4_USB_HID**. PSoC Creator builds the project and displays the comments in the **Output** dialog box. When you see the message “Build Succeeded” the build is complete.

3.2.4.7 *Configuring and Programming the PSoC Development Board*

Note The PSoC Development Board is able to draw up to 1 amp of current. As a result, be careful when powering the development board from the USB 'VBUS' power rail. Connect the development board to the host PC's root hub rather than an external hub. To reduce the chances of disrupting other USB peripherals make certain that the hub is self powered.

1. Disconnect the power to the board.
2. Configure the DVK bread board SW3 to 3.3V.
3. Configure the DVK bread board using the jumper wires.
   - P1[5] to SW1
   - P1[6] to LED1
4. Connect the USB cable to the PC and to the USB port (J9)
5. Reapply power to the board.
6. Use PSoC Creator as described in section Programming a Device on page 15 to program the device.
7. After programing the device, press **Reset**.
8. When button SW1 is pressed, the Windows Run window opens and the keyboard key sequence for the Cypress website is sent to open the Cypress website.
9. Save and close the project.

3.2.5 CapSense

This project demonstrates CapSense. The firmware begins by initializing the LCD and CapSense components. In the main loop it scans the two buttons for activity. If there is a signal from either or both buttons, the corresponding LED lights up.

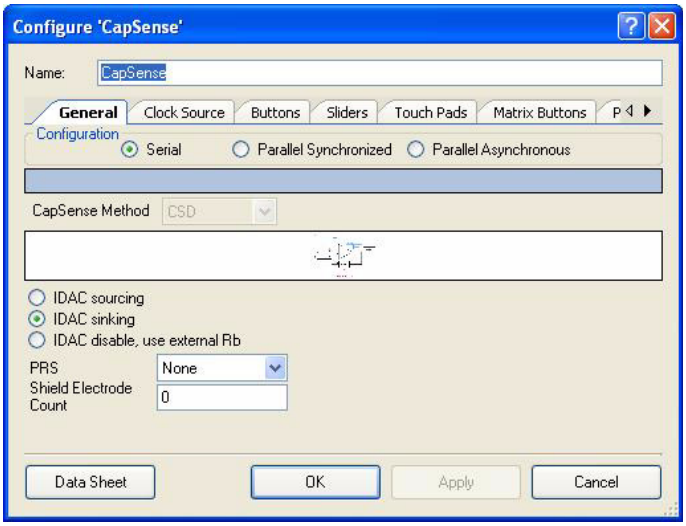
3.2.5.1 Creating CapSense Project

1. Open PSoC Creator.
2. Create a new project by clicking **Create New Project…** in the **Start Page** of PSoC Creator.
3. In the **New Project** window, select the **Empty PSoC3 Design** template and name the project **Ex5_CapSense**.
4. In Location, type the path where you want to save the project, or click ... and navigate to the appropriate directory.
5. By default, the design window opens *TopDesign.cysch*. This is the project's schematic entry file within PSoC Creator.

3.2.5.2 Placing and Configuring CapSense

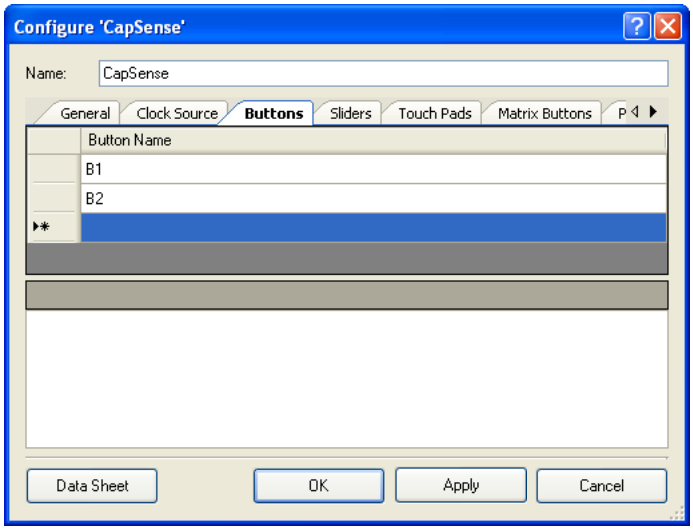
1. Drag-and-drop a CapSense component from the Component Catalog to the workspace.
2. Double click the **CapSense_1** component
3. Configure it in this manner.

Figure 3-56. CapSense Component Configuration



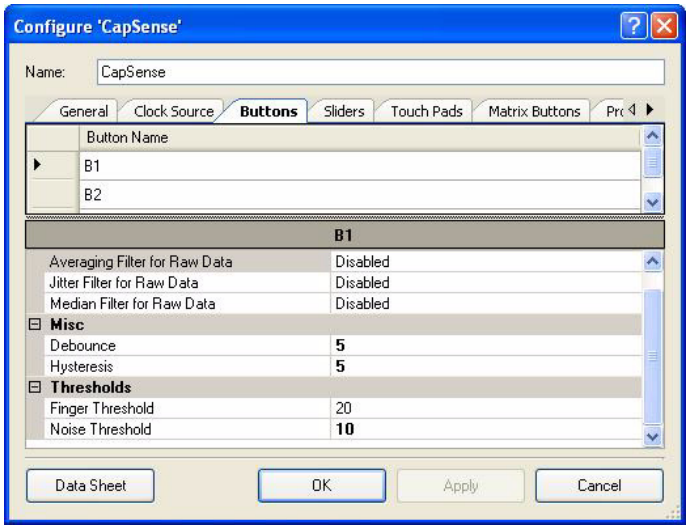
4. Name the component **CapSense**.
5. Select **IDAC sinking**.
6. Select the **Buttons** tab.

Figure 3-57. Buttons - CapSense Component Configuration



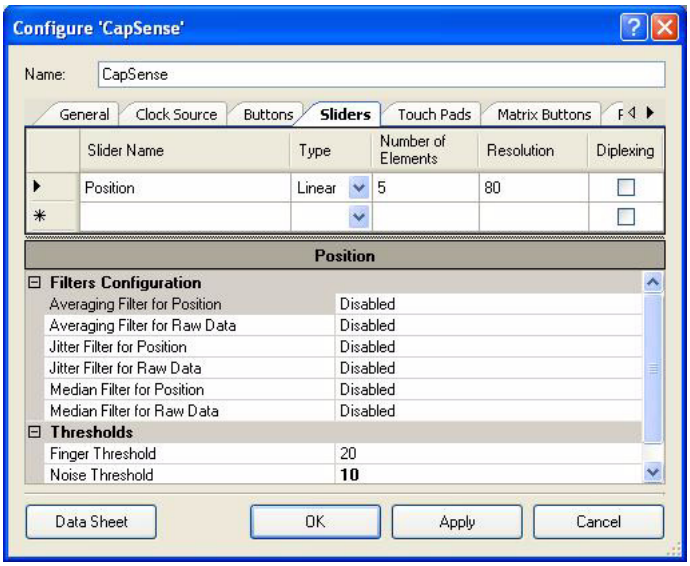
7. Type these names for button one, **B1** and button two, **B2** in the **Button Name** field.

Figure 3-58. B1 Button Configuration



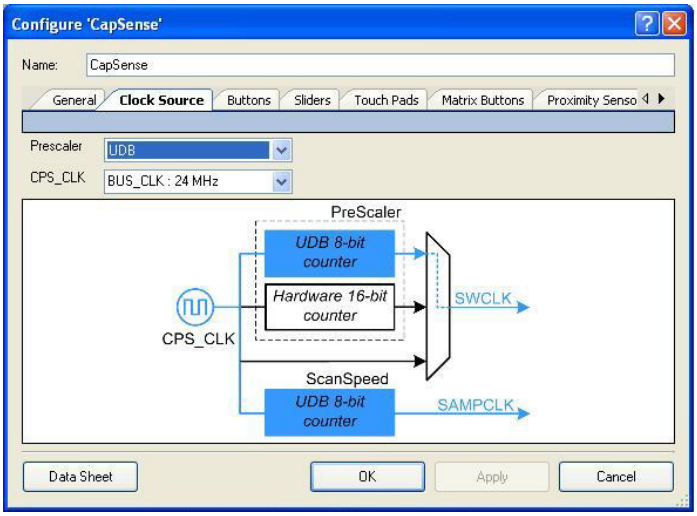
8. Select the **Sliders** tab.
9. Enter **Position** for the **Slider Name**.
10. Select **Linear** for **Type**.
11. Enter **5** for **Number of Elements**.
12. Enter **80** for **Resolution**.

Figure 3-59. Slider Configuration



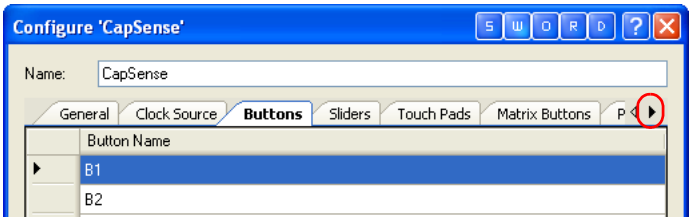
13. Select the **Clock Source** tab.
14. Select **UDB** from the **Prescaler** drop down list. This enables the **Prescaler Period** in **Scan Slots** tab.

Figure 3-60. Clock Source Tab



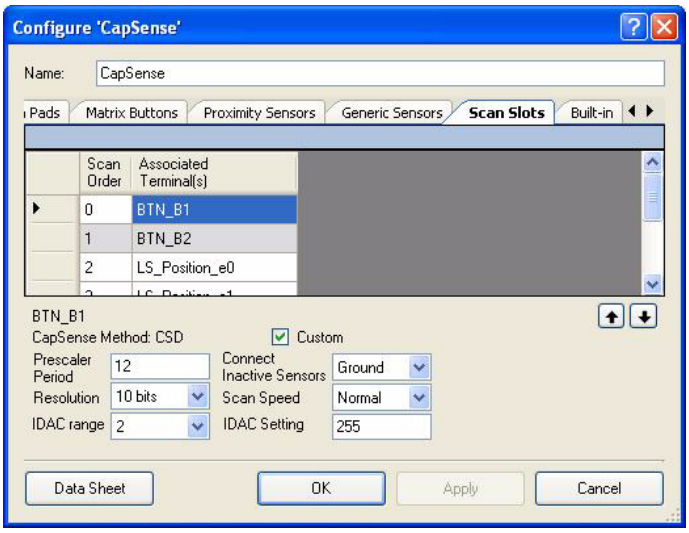
15. Select the **Scan Slots** tab using the right arrow.

Figure 3-61. Click Right Arrow



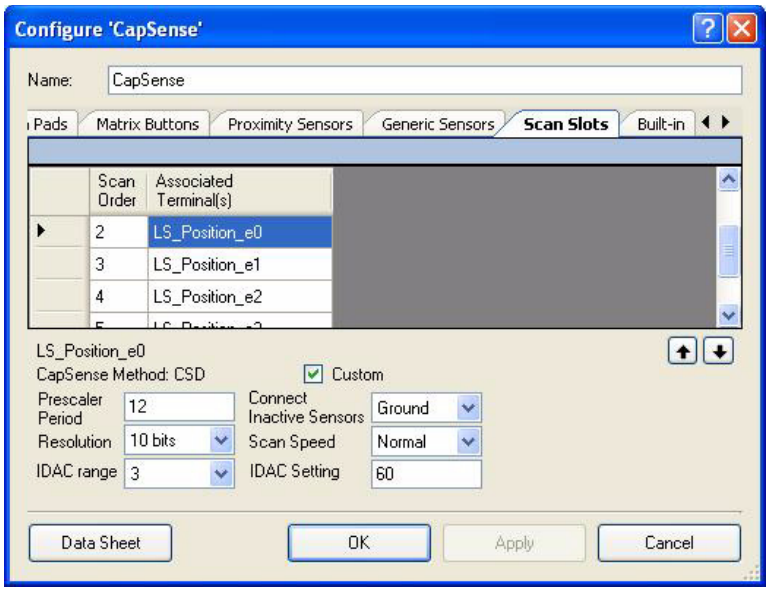
16. Configure button **BTN_B1** in the **Scan Slots** tab in this manner.
    - Select the **Custom** Check Box
    - **Prescaler Period**: 12
    - **Resolution**: 10 bits
    - **IDAC Range**: 2
    - **Connect Inactive Sensors**: Ground
    - **Scan Speed**: Normal
    - **IDAC Setting**: 255

Figure 3-62. Scan Slots Configuration



17. Configure **BTN_B2** similar to **BTN_B1**.
18. Click **OK**.
19. Configure slider terminals **LS_Position_e0** through **LS_Position_e4** in the **Scan Slots** tab:
    - Select **Custom**
    - **Prescaler Period**: 12
    - **Resolution**: 10 bits
    - **IDAC Range**: 3
    - **Connect Inactive Sensors**: Ground
    - **Scan Speed**: Normal
    - **IDAC Setting**: 60

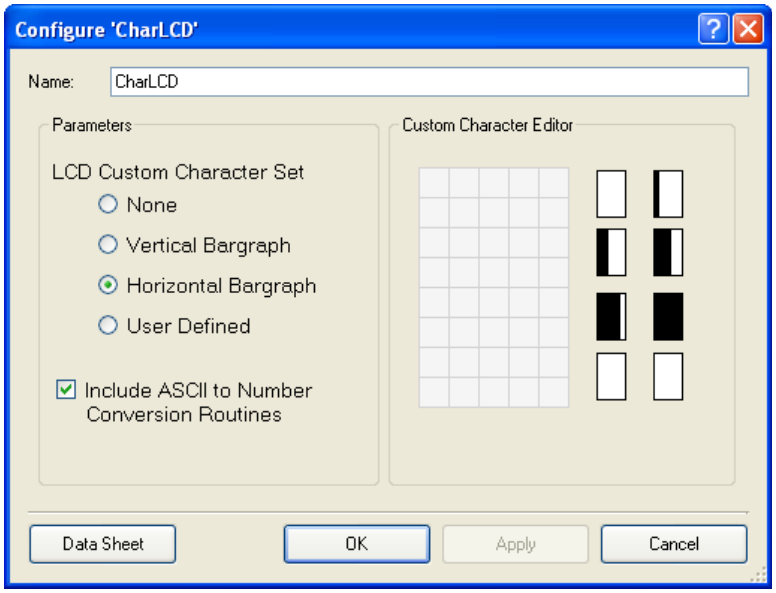
Figure 3-63. Scan Slots Slider Terminals Configuration



3.2.5.3 *Placing and Configuring Character LCD*

1. Drag-and-drop a Character LCD component from the Component Catalog to the workspace.
2. Double click the **LCD_Char_1** component.
3. Set the parameter **LCD Custom Character Set** to **Horizontal Bargraph**.

Figure 3-64. Horizontal Bargraph Configuration

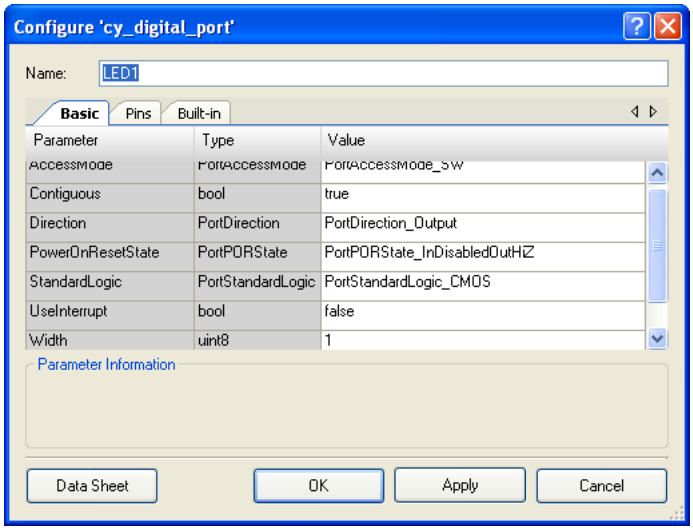


3.2.5.4 *Placing and Configuring Digital Port*

1. Drag-and-drop a Digital Port component from the Component Catalog to the workspace.
2. Configure **LCD_Char_1**:
   - **Name**: LCD
   - Select the **Include ASCII to Number Conversion Routines**.
   - Select **Horizontal Bargraph**
3. Click **OK**.
4. Configure the two **Digital Port** components for LED1 and LED2.

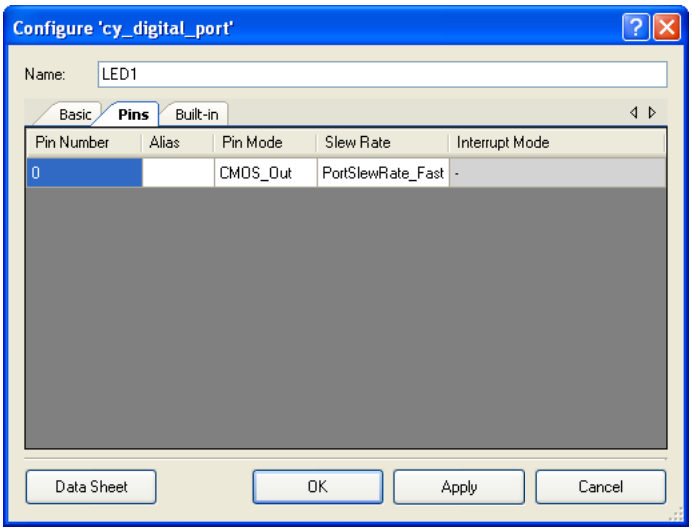
   Basic Tab
   - **Name**: **LED1**
   - **AccessMode**: PortAccessMode_SW
   - **Contiguous**: true
   - **Direction**: PortDirection_Output
   - **PowerOnResetState**: PortPORState_InDisabledOutHiZ
   - **StandardLogic**: PortStandardLogic_CMOS
   - **UseInterrupt**: false
   - **Width**: 1

Figure 3-65. LED Configuration



   - **Pins** Tab
   - **Pin Mode**: CMOS_Out

Figure 3-66. Pins - LED Configuration



5. Click **OK**.
6. Configure **LED2** similar to **LED1**.

3.2.5.5 *Configuring the Pins*

1. Assign the pins in this manner:
   - Cmod to P2[7]
   - B1 to P0[5]
   - B2 to P0[6]
   - Position_e0 to P0[0]
   - Position_e1 to P0[1]
   - Position_e2 to P0[2]
   - Position_e3 to P0[3]
   - Position_e4 to P0[4]
   - LED1 to P1[6]
   - LED2 to P1[7]
   - CharLCD to P2[0] to P2[6] (Drag it to P2[0] and PSoC Creator assigns the pin correctly.)

Figure 3-67. Pin Assignment

Alias	Name	Pin
sCmod	CapSense_sbCSD_cCmod	P2[7]
LS_Position_e4	CapSense_sbCSD_cPort[6]	P0[4]
LS_Position_e3	CapSense_sbCSD_cPort[5]	P0[3]
LS_Position_e2	CapSense_sbCSD_cPort[4]	P0[2]
LS_Position_e1	CapSense_sbCSD_cPort[3]	P0[1]
LS_Position_e0	CapSense_sbCSD_cPort[2]	P0[0]
BTN_B2	CapSense_sbCSD_cPort[1]	P0[6]
BTN_B1	CapSense_sbCSD_cPort[0]	P0[5]
	CharLCD_LCDPort[6:0]	P2[6:0]
	LED2	P1[7]
	LED1	P1[6]

3.2.5.6 Creating the main.c File

1. From the **Workspace Explorer** double click the *main.c* file.
2. Use this code to replace the contents of the *main.c* file. (A soft copy of the *main.c* file is embedded in this PDF under “Attachments.”)

main.c

```
#include <device.h>

/* LCD specific */
#define ROW_0     0  /* LCD row 0     */
#define ROW_1     1  /* LCD row 1     */
#define COLUMN_0  0  /* LCD column 0  */
#define NUM_CHARACTERS 16 /* Number of characters on LCD */

/* For clearing a row of the LCD*/
#define CLEAR_ROW_STR     "                "
/* Button 1 only string for row 0 of the LCD */
#define BUTTON_1_STR      "Button1         "
/* Button 2 only string for row 0 of the LCD */
#define BUTTON_2_STR      "         Button2"
/* Button 1 and 2 string for row 0 of the LCD */
#define BUTTON_1_2_STR    "Button1  Button2"
/* Default string for button row of the LCD */
#define DEFAULT_ROW_0_STR "Touch Buttons   "
/* Default string for slider row of the LCD */
#define DEFAULT_ROW_1_STR "Touch The Slider"

/* LED specific */
#define LED_ON  1 /* For setting LED pin high */
#define LED_OFF 0 /* For setting LED pin low */

/* CapSense specific */
#define SLIDER_RESOLUTION 80

extern const uint8 CharLCD_customFonts[];

void UpdateButtonState(uint8 slot_1, uint8 slot_2);
void UpdateSliderPostion(uint8 value);

/*******************************************************************************
* Function Name: main
********************************************************************************
*
* Summary:
*  The main function initializes CapSense and the LCD.  Then it continuously
*  scans all CapSense slots (slider slots and buttons), gets the state of the
*  buttons and slider and updates the LCD with the current state.
*
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
void main()
{
```

```
    uint8 pos, stateB_1, stateB_2;

     CYGlobalIntEnable; /* Enable global interrupts */

    /* LCD and CapSense Initialization */
    CharLCD_Start();
     CharLCD_LoadCustomFonts(CharLCD_customFonts);

    CapSense_Start();
    CapSense_CSHL_InitializeAllBaselines();

    while(1)
    {
       CapSense_CSD_ScanAllSlots();
       CapSense_CSHL_UpdateAllBaselines();

       stateB_1 = CapSense_CSHL_CheckIsSlotActive(CapSense_SCANSLOT_BTN_B1);
       stateB_2 = CapSense_CSHL_CheckIsSlotActive(CapSense_SCANSLOT_BTN_B2);

       /* Find Slider Position */
       pos = CapSense_CSHL_GetCentroidPos(CapSense_CSHL_LS_POSITION);

         UpdateButtonState(stateB_1, stateB_2);
       UpdateSliderPostion(pos);
}
}

/*******************************************************************************
* Function Name: UpdateButtonState
********************************************************************************
*
* Summary:
*  Updates the LCD screen with the current button state by displaying which
*  button is being touched on row 0. LED's are also updated according to button
*  state.
*
* Parameters:
*  slot_1: Button state for B1
*  slot_2: Button state for B2
*
* Return:
*  void
*
*******************************************************************************/
void UpdateButtonState(uint8 slot_1, uint8 slot_2)
{
    CharLCD_Position(ROW_0,COLUMN_0);

    /* Check the state of the buttons and update the LCD and LEDs*/
    if (slot_1 && slot_2)
    {
       CharLCD_PrintString(BUTTON_1_2_STR);
       LED1_Write(LED_ON); /* Set the LED */
       LED2_Write(LED_ON); /* Set the LED */
    }
    else if (slot_1 || slot_2)
    {
       if (slot_1)
```

```
        {
           CharLCD_PrintString(BUTTON_1_STR);
           LED1_Write(LED_ON); /* Set the LED */
        }
        if (slot_2)
        {
           CharLCD_PrintString(BUTTON_2_STR);
           LED2_Write(LED_ON); /* Set the LED */
        }
    }
    else
    {
         CharLCD_PrintString(DEFAULT_ROW_0_STR);
       LED1_Write(LED_OFF); /* Set the LED */
       LED2_Write(LED_OFF); /* Set the LED */
    }
}

/*******************************************************************************
* Function Name: UpdateSliderPostion
********************************************************************************
*
* Summary:
*  Updates the LCD screen with the current slider position by displaying the
*  horizontal bar graph.
*
* Parameters:
*  value: Centroid position from CapSense slider.
*
* Return:
*  void
*
*******************************************************************************/
void UpdateSliderPostion(uint8 value)
{
    if (value > SLIDER_RESOLUTION)
    {
       /* Clear old slider position (2nd row of LCD) */
       CharLCD_Position(ROW_1, COLUMN_0);
        CharLCD_PrintString(DEFAULT_ROW_1_STR);
    }
    else
    {
       CharLCD_DrawHorizontalBG(ROW_1, COLUMN_0, NUM_CHARACTERS, value);
    }
}

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

3. From the **Build** menu, select **Build CapSense**. PSoC Creator builds the project and displays the comments in the **Output** dialog box. When you see the message “Build Succeeded” the build is complete.

3.2.5.7 *Configuring and Programming the PSoC Development Board*

1. Disconnect power to the board.
2. Configure the DVK bread board SW3 to 3.3V.
3. Using the jumper wires, configure the PSoC Development Board's prototyping area:
   - P1_6 to LED1
   - P1_7 to LED2
4. Use PSoC Creator as described in Programming a Device on page 15 to program the device.
5. After programing the device, press **Reset**.
6. When running the project, an LED lights up when either CapSense button is pushed. If B1 (P0[5]) is pushed it also displays Button1 in the top row of the LCD display. Likewise, If B2 (P0[6]) is pushed it displays Button2 in the top row of the LCD display. The bottom row of the LCD displays the Slider position with a Horizontal Bar graph.
7. Save and close the project.

Appendix A. Board Specifications and Layout

This appendix gives detailed specifications of the PSoC DVK board components

A.1 PSoC Development Board

A.1.1 Factory Default Configuration

A.1.1.1 Power Supply

The board has several power nets. Below are the definitions of the different power nets.

VIN (9V or 12V) - This is the input power before it is fed to any of the regulators. A 9-12V wall wart supply or a 9V battery is used as the source.

VREG (5V) - This is fed by VIN and is the output of the onboard 5V regulator. VREG can be selected as the main 5V source by using the J8 header.

VBUS (5V) - This is power derived from the USB interface via a USB host. VBUS can be selected as the main 5V source by using the J8 header.

VDD (3.3V or 5V) - This is fed by VREG, VBUS, or the onboard 3.3V regulator. VDD can be chosen either to be 3.3V or 5V by the simple positioning of the VDD select switch.

VADJ (1.25V to 3.9V) - This is fed by VDD and is the output of the onboard adjustable regulator. It is mainly used when the PSoC core must be powered at lower voltages. An adjustable resistor R11 is used for adjusting the voltage.

VDD DIG - This is power derived from either VDD or VADJ. It is used to power the PSoC core. The source for VDD DIG can be chosen as VDD or VADJ using the J7 header.

VDD ANLG - This is power derived from either VDD or VADJ. It is mainly used when user wants to separate the analog power from the digital power. The source for VDD ANLG can be chosen as VDD or VADJ using the J7 header.

VDDIO - This is power derived from either VDD or VADJ. It is used to power digital I/O on the PSoC device. There are four sections of GPIO, which can be powered to 5V, 3.3V, or VADJ using four headers. It enables the user to power the PSoC GPIOs at different voltages.

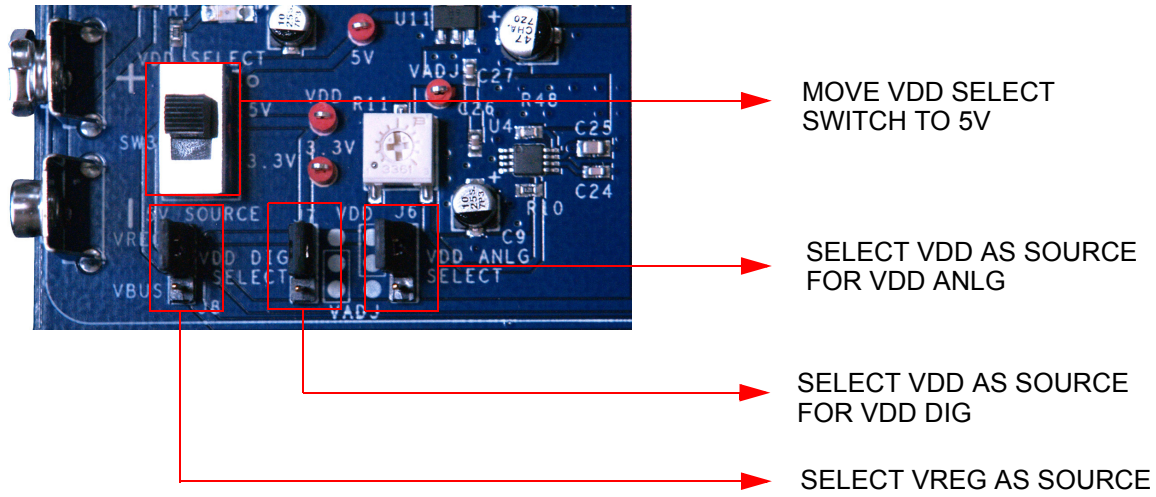
A.1.2 Power Supply Configuration Examples

A.1.2.1 Setting a 5V Supply from VREG

1. Place the jumper on J8 header to select VREG as the source.
2. Move the VDD select switch to select the 5V.
3. Place the jumper on J6 header to select VDD as source for VDD ANLG.
4. Place the jumper on J7 header to select VDD as source for VDD DIG.

Note 5V operation is not recommended with ES1 Silicon. See the silicon errata for details.

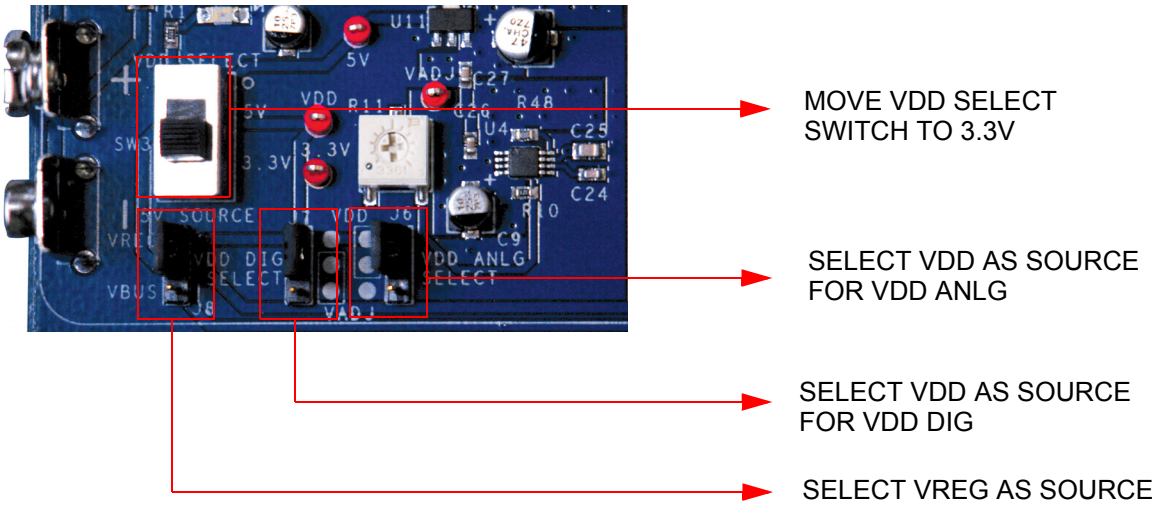
Figure A-1. Setting a 5V Supply from VREG



A.1.2.2 Setting a 3.3V Supply from VREG

1. Place the jumper on J8 header to select VREG as the source.
2. Move the VDD select switch to select 3.3V.
3. Place the jumper on J6 header to select VDD as source for VDD ANLG.
4. Place the jumper on J7 header to select VDD as source for VDD DIG.

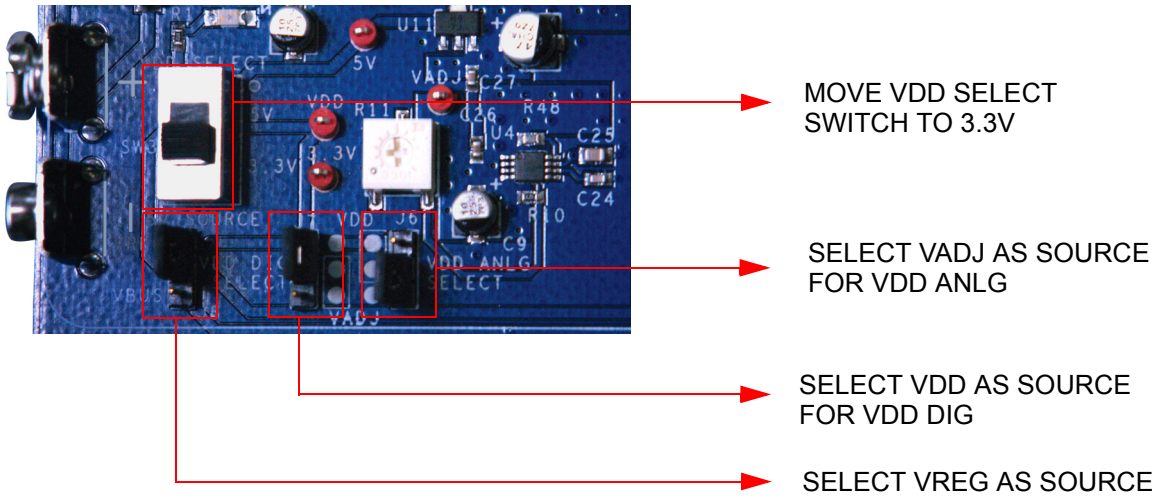
Figure A-2. Setting a 3.3V Supply from VREG



A.1.2.3 *Setting VDD ANLG as VADJ and VDD DIG as VDD for VDD = 3.3V*

1. Place the jumper on J8 header to select VREG as the source.
2. Move the VDD select switch to select 3.3V.
3. Place the jumper on J6 header to select VADJ as source for VDD ANLG.
4. Place the jumper on J7 header to select VDD as source for VDD DIG.

Figure A-3. Setting VDD ANLG as VADJ and VDD DIG as VDD for VDD = 3.3V

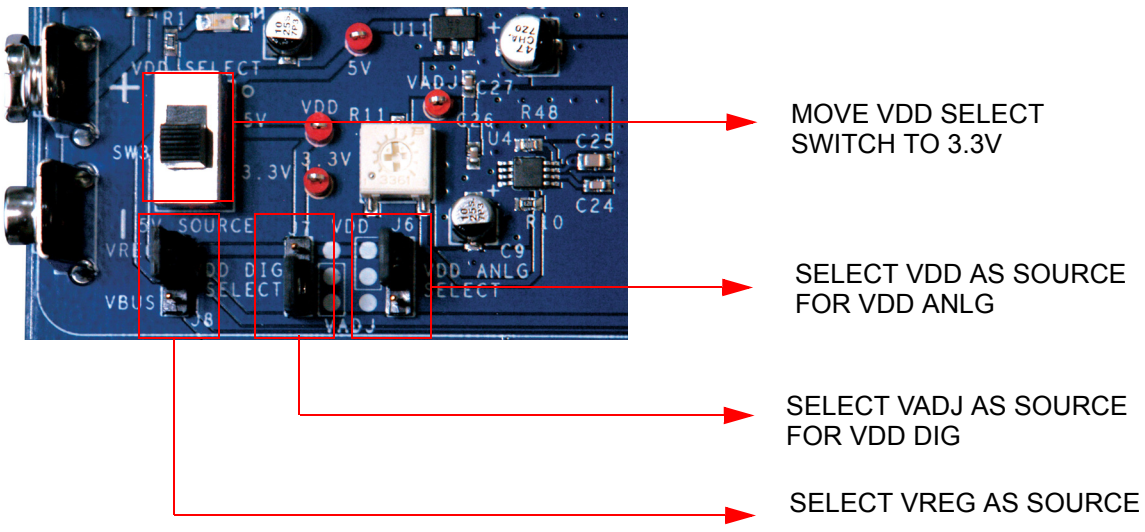


This helps to separate the analog supply from the digital supply and VDD.

A.1.2.4 *Setting VDD DIG as VADJ and VDD ANLG as VDD for VDD = 3.3V*

1. Place the jumper on J8 header to select VREG as the source.
2. Move the VDD select switch to select 3.3V.
3. Place the jumper on J6 header to select VDD as source for VDD ANLG.
4. Place the jumper on J7 header to select VADJ as source for VDD DIG.

Figure A-4. Setting VDD DIG as VADJ and VDD ANLG as VDD for VDD = 3.3V



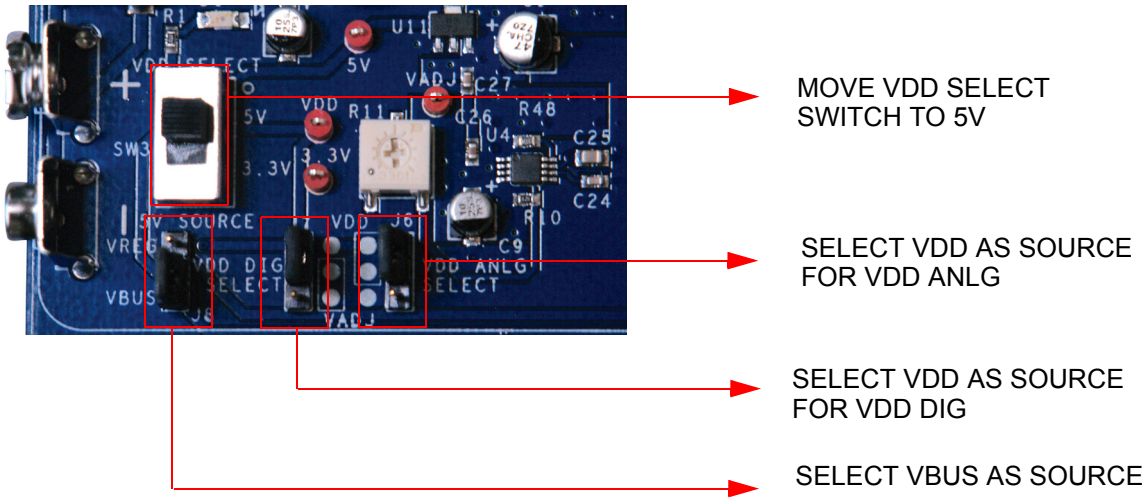
This helps to separate the digital supply from the analog supply and VDD.

A.1.2.5 *Setting a 5V Supply from VBUS*

1. Place the jumper on J8 header to select VBUS as the source.
2. Move the VDD select switch to select the 5V.
3. Place the jumper on J6 header to select VDD as source for VDD ANLG.
4. Place the jumper on J7 header to select VDD as source for VDD DIG.

Note 5V operation is not recommended with ES1 Silicon. See the silicon errata for details.

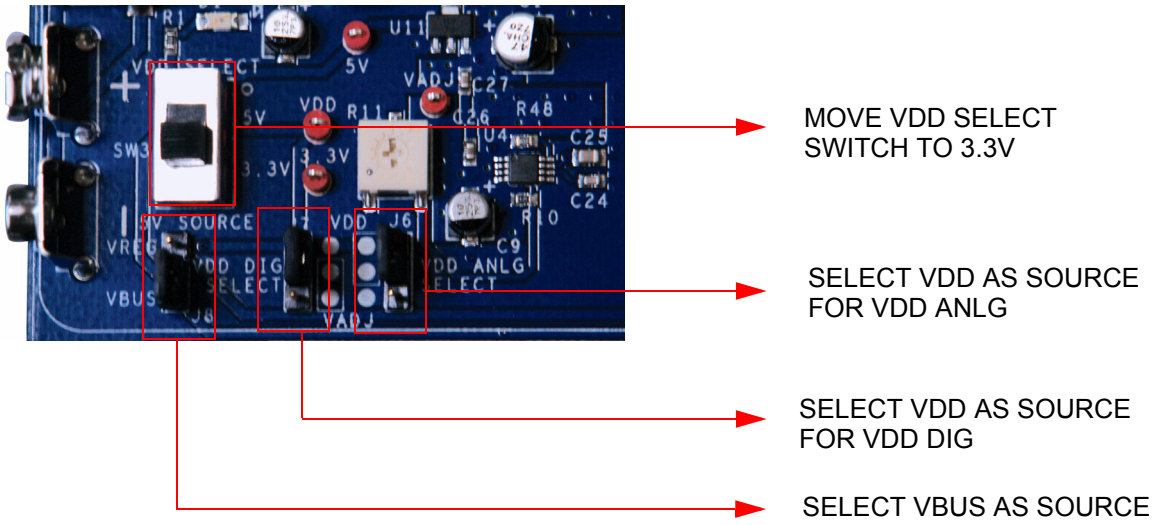
Figure A-5. Setting a 5V Supply from VBUS



A.1.2.6 *Setting a 3.3V Supply from VBUS*

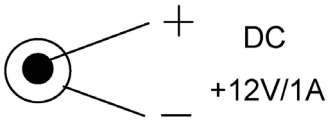
1. Place the jumper on J8 header to select VREG as the source.
2. Move the VDD select switch to select 3.3V.
3. Place the jumper on J6 header to select VDD as source for VDD ANLG.
4. Place the jumper on J7 header to select VDD as source for VDD DIG.

Figure A-6. Setting a 3.3V supply from VBUS



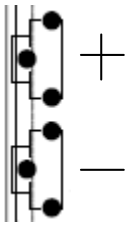
You can measure current from VREG, VBUS, VDD ANLG, VDD DIG and VDDIOs by removing the jumpers and connecting the meter across the respective header.

A.1.2.7 J1 - DC Power Jack



Use a 12V/1A wall wart power supply when powering from the barrel power jack. This input power is VIN.

A.1.2.8 9V Battery Terminals



Use a 9V alkaline battery to connect to the 9V battery terminals. This input power is VIN.

A.1.2.9 J8 - 5V Source

This header allows the user to select the 5V source from either the onboard 5V regulator (VREG) or from the USB 5V rail (VBUS).

A.1.2.10 VDD Select Switch

This switch allows for selecting either 5V or 3.3V. VDD feeds VDD DIG, VDD ANLG and VDDIO.

A.1.2.11 J7 - VDD DIG Select

This header allows the user to select the PSoC core source power. If you need to power the PSoC core at either 5V or 3.3V (based on the position of the VDD select switch), place the jumper on the upper two pins. If you need to power the PSoC core at lower voltages (1.25V to 3.9V), then place the jumper on the lower two pins. When the jumper is on the lower two pins, you must adjust R13 to tune the adjustable regulator to output the desired voltage.

A.1.2.12 J6 - VDD ANLG Select

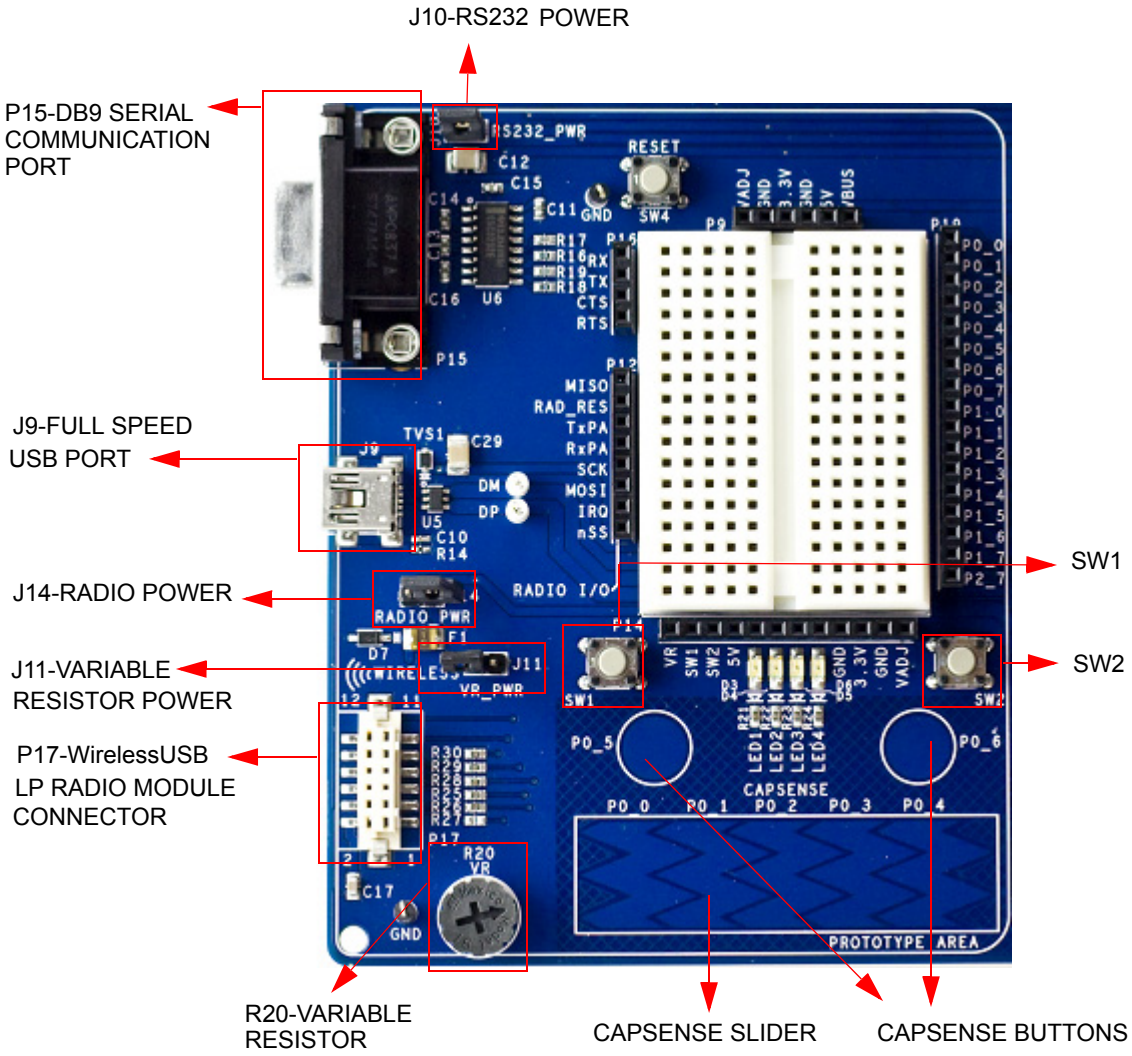
If you want to separate the analog power from the digital power, you can position the jumper on the upper two pins to source analog power at 5V or 3.3V (based on the position of the VDD select switch), or on the lower two pins to source analog power at lower voltages (1.25V to 3.9V).

A.1.2.13 R11 - Adjustable Regulator Variable Resistor

This adjustable resistor is used to tune the VADJ voltage. Turning this variable resistor swings the VADJ voltage between 1.25V and 2.3V when the VDD select switch is in the 3.3V position. When the VDD select switch is in the 5V position, turning this variable resistor swings the VADJ voltage between 1.25V and 3.9V.

A.1.3 Prototyping Components

A.1.3.1 Prototyping Area



Note Port1 pins are used for SWD/JTAG. The P1_6 and P1_7 pins are used for CapSense.

A.1.3.2 P15 - DB9 Serial Communications Port

This is a standard female DB9 serial communications connector. Four signals are brought from the RS232 transceiver to receptacle P16. These signals are Rx, Tx, Clear To Send, and Request To Send. To connect these signals to the PSoC I/O pins, use wires to jumper from P16 to P19, where sockets for ports zero and one are available.

A.1.3.3 J10 - Serial Port Power

Header J10 must be jumpered in order to use the serial communications port. Placing a jumper on J10 provides VDD power to the RS232 transceiver. This power can be either 3.3V or 5V, depending on which the position of the VDD select switch.

A.1.3.4 J9 - Full Speed USB Port

The board has a mini-B full speed USB connector. There are also two test points for the differential pair signals D- and D+. These signals are routed to the processor module socket P1, pins 6 and 8 respectively. The power net VBUS is brought into the board through this interface.

A.1.3.5 P17 - Artaflex WirelessUSB LP Radio Module Receptacle

Receptacle P17 is used specifically for the Artaflex AWP24S WirelessUSB module. Eight signals are routed from this receptacle to P12 receptacle. These signals are 4 SPI (serial peripheral interface) signals MISO (master-in-slave-out), MOSI (master-out-slave-in), nSS (slave select), SCK (serial clock), an IRQ (interrupt request) and RD_RESET (radio reset). The other two signals are radio transmit and receive signals. **Note**: These I/O signals must not be greater than 3.3V.

A.1.3.6 J14 - Wireless Radio Module Power

Header J14 must be jumpered in order to use the Artaflex radio module. Placing a jumper on J14 provides 3.3V power to the P17 module socket. This power is drawn directly from the 3.3V regulator.

A.1.3.7 R20 - Multipurpose Variable Resistor

The board is equipped with a 10 kΩ thumbwheel variable resistor referenced to ground. The high side of the resistor is tied to jumper J11. The wiper is tied to a receptacle pin on P14.

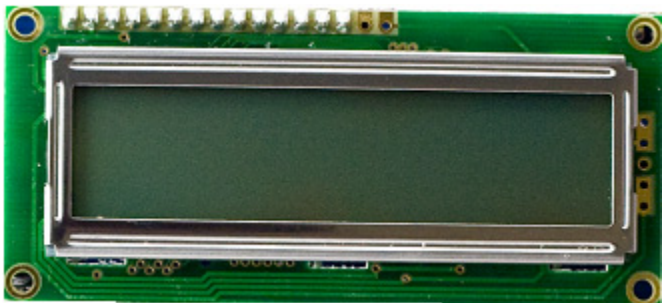
A.1.3.8 J11 - Variable Resistor Power

Header J11 must be jumpered in order to use the variable resistor. Placing a jumper on J11 provides VDD ANLG power to the high side of the resistor.

A.1.3.9 SW1 and SW2 - Multipurpose Push Button Switches

The board has two multipurpose mechanical push buttons, SW1 and SW2, that are referenced to ground. The other sides of the switches are tied to receptacle pins on P14. The switches follow an inverted logic as they connect ground to receptacle pins on P14 when pressed.

A.1.4 LCD Module



The board has a 2x16 alpha-numeric LCD. I/Os of the module are connected to port two of the PSoC device and are routed to the processor module socket P2. This LCD is rated for 5V. However, the I/Os have a level translator inline so that signaling may be as low as 1.8V and still be recognized by the LCD. The header J12 must be jumpered for the LCD Module to be powered. If J12 is not jumpered, it removes power from level translator. If the LCD module is removed, the receptacle pins of P18 can be used as port 2.

A.1.4.1 R31 - LCD Contrast Adjustment

The board is equipped with an LCD contrast adjustment resistor R31. Turning the wiper counter-clockwise increases the contrast, while turning the wiper clockwise decreases the contrast.

A.1.4.2 J12 - LCD Module Power

Power for the LCD module is provided through header J12. Placing a jumper on the upper two pins shorts the VCC pin of the module to ground. Placing the jumper on the lower two pins provides 5V to the VCC pin of the module. This 5V power is taken directly from the onboard 5V regulator.

A.1.5 CapSense Elements

The prototyping area has three capacitive sensing elements. There are two CapSense buttons connected directly to port zero pins. In addition, there is a five segment CapSense slider also connected directly to port zero. Series resistors are placed on these port zero I/Os and should be loaded with appropriate values. A value of 0Ω is used for general purpose CapSense applications, but a value of 560Ω should be used for achieving best performance. The board is loaded with 0Ω series resistors by default. The presence of CapSense elements does not affect the general purpose use of port zero pins.

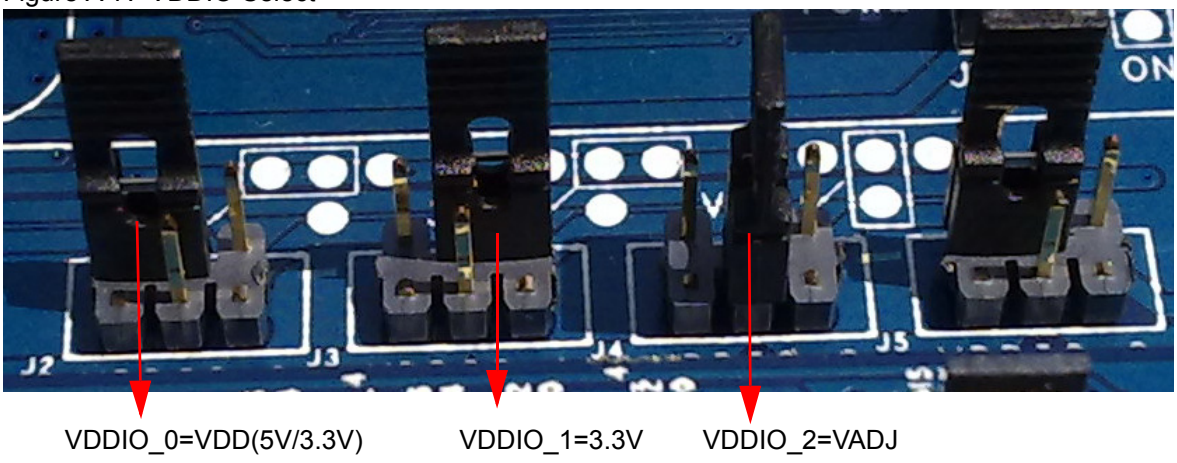
A.1.6 Processor Module

A.1.6.1 J2, J3, J4 and J5 - VDDIO Select

These four headers allow the user to power the PSoC GPIOs at different voltages. For instance, some of the I/O may be powered at 5V, some at 3.3V and some at 1.8V. There are four blocks of GPIO, each having its own source power. Each VDDIO header provides power to specific GPIO's and is selectable from VDD, 3.3V or VADJ. For details on which GPIOs are powered by which VDDIO header, refer to the data sheet for the PSoC device used with this board.

For example, VDDIO_0 is configured to VDD, VDDIO_1 is configured to 3.3V and VDDIO_2 is configured to VADJ by placing the jumpers in the respective positions as shown in Figure A-7.

Figure A-7. VDDIO Select



A.1.6.2 SW4 - Processor Reset Button

The board has a push button switch that resets the PSoC device attached to the processor module. One side of the switch is tied to the XRES pin of the processor module socket. The other end of the switch is tied to the HW_RESET pin of the processor module socket. Doing this allows the module

designer to tie the HW_RESET line either high or low, depending on which direction the processor reset is active.

Note PSoC1 devices are active high reset. Therefore, a light pull-down resistor may be necessary on the XRES pin of designs with these devices to avoid unintentional device resets. PSoC3 and PSoC5 devices are active low reset. Therefore, a light pull-up resistor may be necessary on the XRES pin of designs with these devices to avoid unintentional device resets.

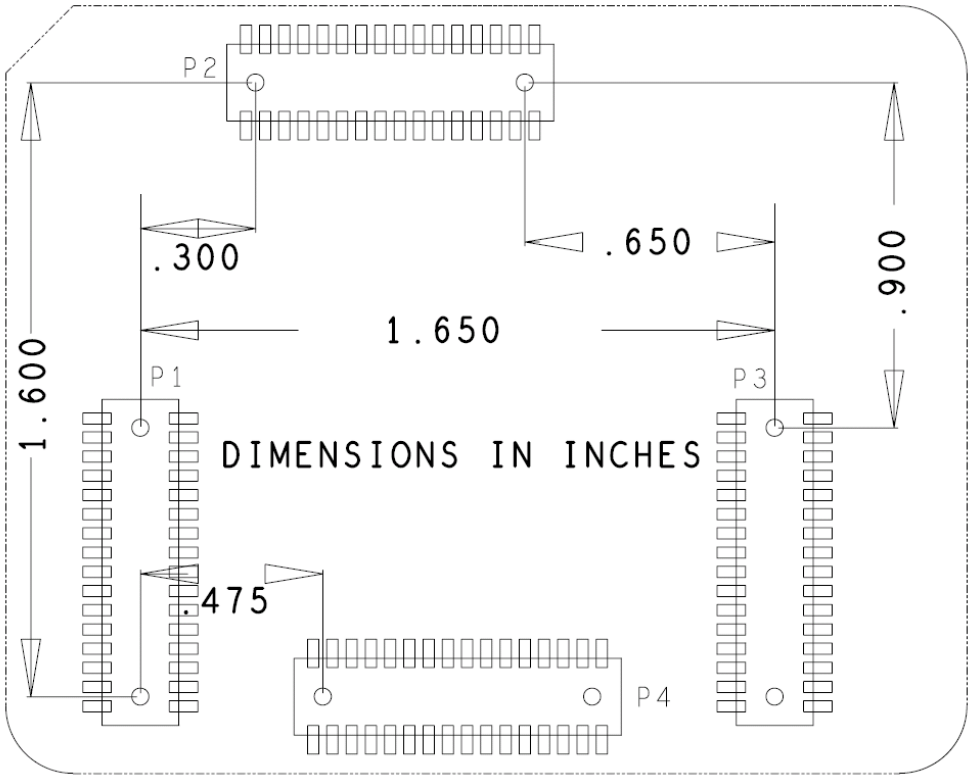
A.1.6.3 U8 - External MHz Oscillator

The board supports the use of an external high frequency 8-pin PDIP oscillator. The speed of the oscillator supported is dependent on the specifications of the PSoC device used.

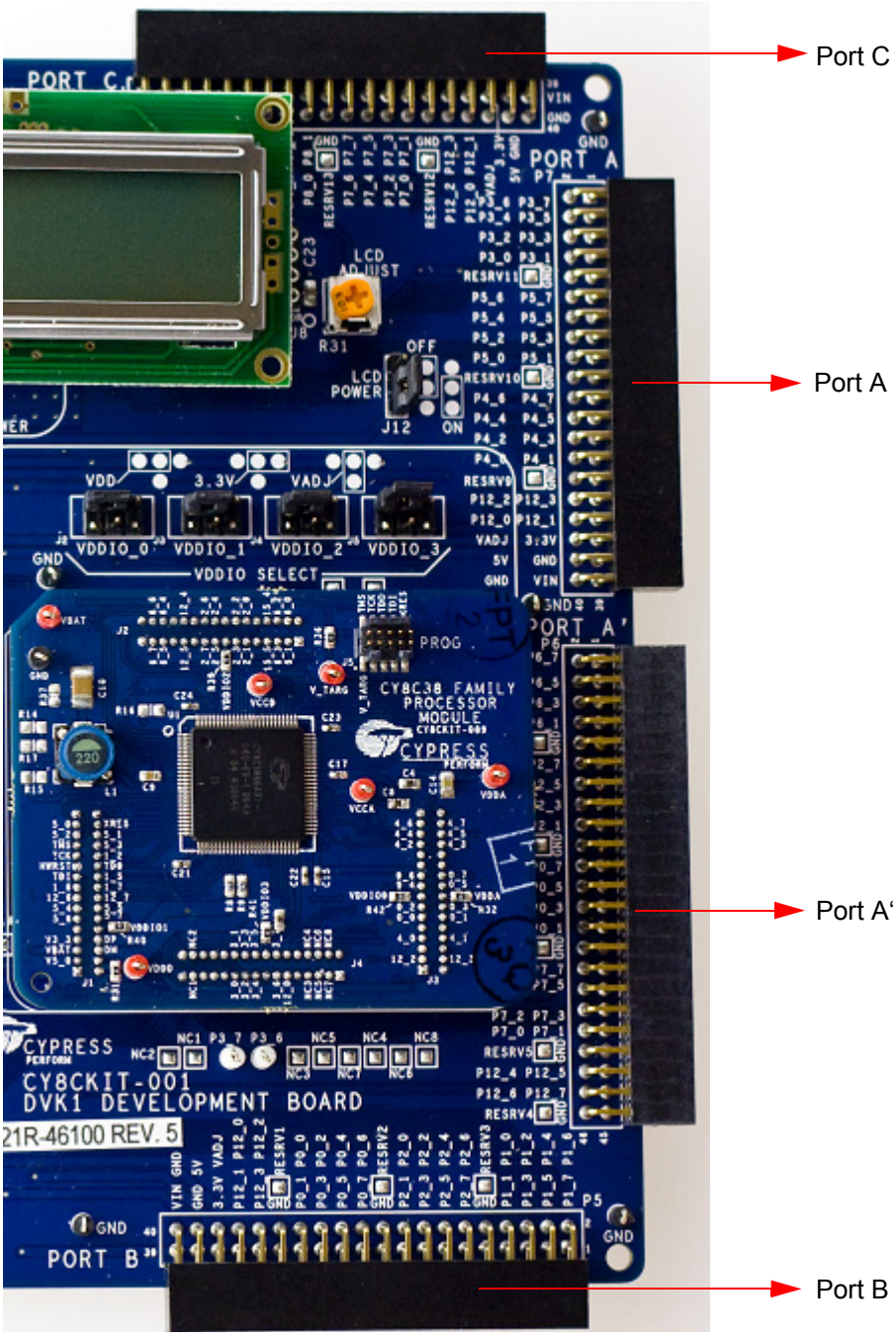
A.1.6.4 P1, P2, P3 and P4 - Processor Module Receptacles

Processor modules provide modularity to this board. Sockets P1-P4 are used to connect a processor module to the board. All supported GPIOs (including special I/Os), along with VDD DIG, VDD ANLG, 5V, 3.3V, VBUS and VBAT (only connected to a surface mount pad on the board) are connected to these receptacles. In addition, each of the VDDIO power pins are connected to these receptacles. The full speed USB D+ and D- signals are also connected to one of the sockets. Processor reset is connected to P1. Any “no connect” pins are brought out to surface mount test pads.

Figure A-8. Mechanical Layout Details for Processor Module Connector



A.1.7 Expansion Ports



The board accommodates I/O expandability. Around the upper, lower and right sides of the board are 0.100" pitch, dual row right angle receptacles, each having at least three full 8-bit ports (one has four full ports). Each also has four special I/O pins available. Three of the ports have power and ground pins as well. The fourth is simply I/O and ground exclusively. These sockets can be used to join the processor module I/Os with external I/Os through the use of daughter boards.

A.1.7.1 Expansion Ports A and A'

Expansion port A can be used as I/O ports with three full 8-bit ports port3, port4 and port5. It has four special I/Os as well as ground and voltage pins. It can be used to join processor module I/Os port3, port4, and port5 with external I/Os through the use of daughter boards.

Expansion port A' can be used as I/O ports with four full 8-bit ports port0, port2, port6 and port7. It has four special I/Os as well as ground pins. It has no voltage pins. It can be used to join processor module I/Os port0, port2, port6 and port7 with external I/Os through the use of daughter boards.

The main use of port A' is that it can be used together with port A to join processor module I/Os port0, port2, port3, port4, port5, port6 and port7 with external I/Os through the use of daughter boards.

A.1.7.2 Expansion Port B

Expansion port B can be used as I/O ports with three full 8-bit ports port0, port1 and port2. It has four special I/Os as well as ground and voltage pins. It can be used to join processor module I/Os port0, port1, and port2 with external I/Os through the use of daughter boards. It is mainly used in devices with fewer I/Os.

A.1.7.3 Expansion Port C

Expansion port C can be used as I/O ports with three full 8-bit ports port7, port8 and port9. It has four special I/Os as well as ground and voltage pins. It can be used to join processor module I/Os port7, port8, and port9 with external I/Os through the use of daughter boards. It is used for devices with a high I/O count.

Appendix B. MiniProg3

B.1 MiniProg3 LEDs

MiniProg3 provides five indicator LEDs:

- Upper Left - Busy: A red LED that lights when an operation (such as programming or debug) is in progress.
- Lower Left - Status: A green LED that lights when the device is enumerated on the USB bus and flashes when the MiniProg3 receives USB traffic.
- Upper Right - Target Power: A red LED that lights to indicate that the MiniProg3 is supplying power to the target connectors. Note that it does not light when target power is detected but not being supplied by MiniProg3.
- Lower Right - Aux: A yellow LED reserved for future use.
- Middle - No Label: A yellow LED that indicates the configuration state of the device. It flashes briefly during the initial configuration of the device. If this LED lights solid, a configuration error has occurred and MiniProg3 must be disconnected from the USB port and reconnected.

B.2 Programming in Powercycle Mode

You should not perform powercycle mode programming with PSoC Programmer. This is due to the way the module is designed. VTARG of the MiniProg3 is wired exclusively to VDDIO1 of the chip on the module. In order for powercycle programming to work, VTARG would need to be wired to VDDD.