IDE Guide

Document # 001-42655 Rev *D

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

Contents

1. Introduction

PSoC® Designer™ 5.1 is the revolutionary Integrated Design Environment (IDE) that you can use to customize PSoC to meet your specific application requirements. PSoC Designer software accelerates system bring-up and time-to-market. Develop your applications using a library of pre-characterized analog and digital peripherals in a drag-and-drop design environment. Then, customize your design leveraging the dynamically generated API libraries of code. Finally, debug and test your designs with the integrated debug environment including in-circuit emulation and standard software debug features.

1.1 Application Overview

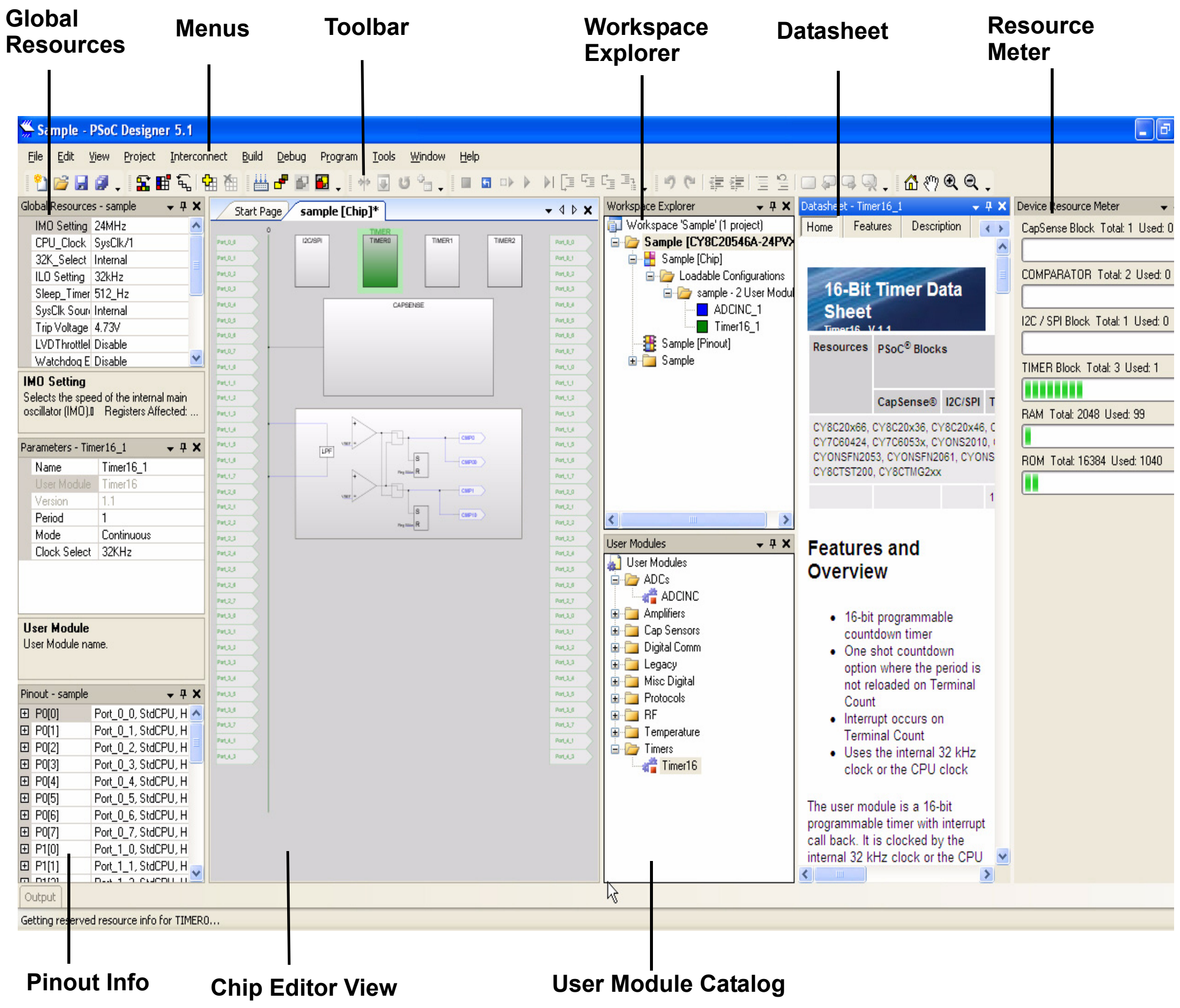
PSoC Designer contains several subsystems: Chip-Level Editor, Code Editor, Build Manager, Project Manager, and Debugger. The interface is split into several active windows that differ depending upon which subsystem you are in. As you move between subsystems, different options are enabled or disabled in the toolbar and menus depending upon the functionality of your PSoC device. The default window layout contains the Chip Editor Workspace Explorer, User Module Catalog, Global Resources, and Datasheet Windows. There are also a number of other windows available from the View menu that show details of different aspects of PSoC Designer.

1.1.1 Chip-Level Editor

If you select the Chip view in the Workspace Explorer, the main view of the project is the Chip-Level Editor. The Chip-Level Editor contains a diagram of the resources available on the chip you have selected; the digital, analog, CapSense®, and other block types that are available on the chip you have selected and the interconnections between them as well as connections to pins. As you place user modules, they will occupy the available resources. You can alter the default placement if you wish. You use this window to route inputs and resources to user modules and user module outputs to other user modules or pins.

You can rearrange the work area to suit your own work style.

Figure 1-1. PSoC Designer Chip Editor



IMO Setting	24MHz
CPU_Clock	SysClk/1
32K_Select	Internal
ILO Setting	32kHz
Sleep_Timer	512_Hz
SysClk Sourc	Internal
Trip Voltage	4.73V
LVDThrottleI	Disable
Watchdog E	Disable

IMO Setting
Selects the speed of the internal main oscillator (IMO). Registers Affected: ...

Name	Timer16_1
User Module	Timer16
Version	1.1
Period	1
Mode	Continuous
Clock Select	32KHz

User Module
User Module name.

P0[0]	Port_0_0, StdCPU, H
P0[1]	Port_0_1, StdCPU, H
P0[2]	Port_0_2, StdCPU, H
P0[3]	Port_0_3, StdCPU, H
P0[4]	Port_0_4, StdCPU, H
P0[5]	Port_0_5, StdCPU, H
P0[6]	Port_0_6, StdCPU, H
P0[7]	Port_0_7, StdCPU, H
P1[0]	Port_1_0, StdCPU, H
P1[1]	Port_1_1, StdCPU, H

Workspace 'Sample' (1 project)
- Sample [CY8C20546A-24PVX]
  - Sample [Chip]
    - Loadable Configurations
      - sample - 2 User Modul
        - ADCINC_1
        - Timer16_1
  - Sample [Pinout]
  - Sample

User Modules
- ADCs
  - ADCINC
- Amplifiers
- Cap Sensors
- Digital Comm
- Legacy
- Misc Digital
- Protocols
- RF
- Temperature
- Timers
  - Timer16

Home Features Description

16-Bit Timer Data Sheet

Resources PSoC® Blocks CapSense® I2C/SPI

CY8C20x66, CY8C20x36, CY8C20x46, CY7C60424, CY7C6053x, CYONS2010, CYONSFN2053, CYONSFN2061, CYONS CY8CTST200, CY8CTMG2xx

Features and Overview

- 16-bit programmable countdown timer
- One shot countdown option where the period is not reloaded on Terminal Count
- Interrupt occurs on Terminal Count
- Uses the internal 32 kHz clock or the CPU clock

The user module is a 16-bit programmable timer with interrupt call back. It is clocked by the internal 32 kHz clock or the CPU

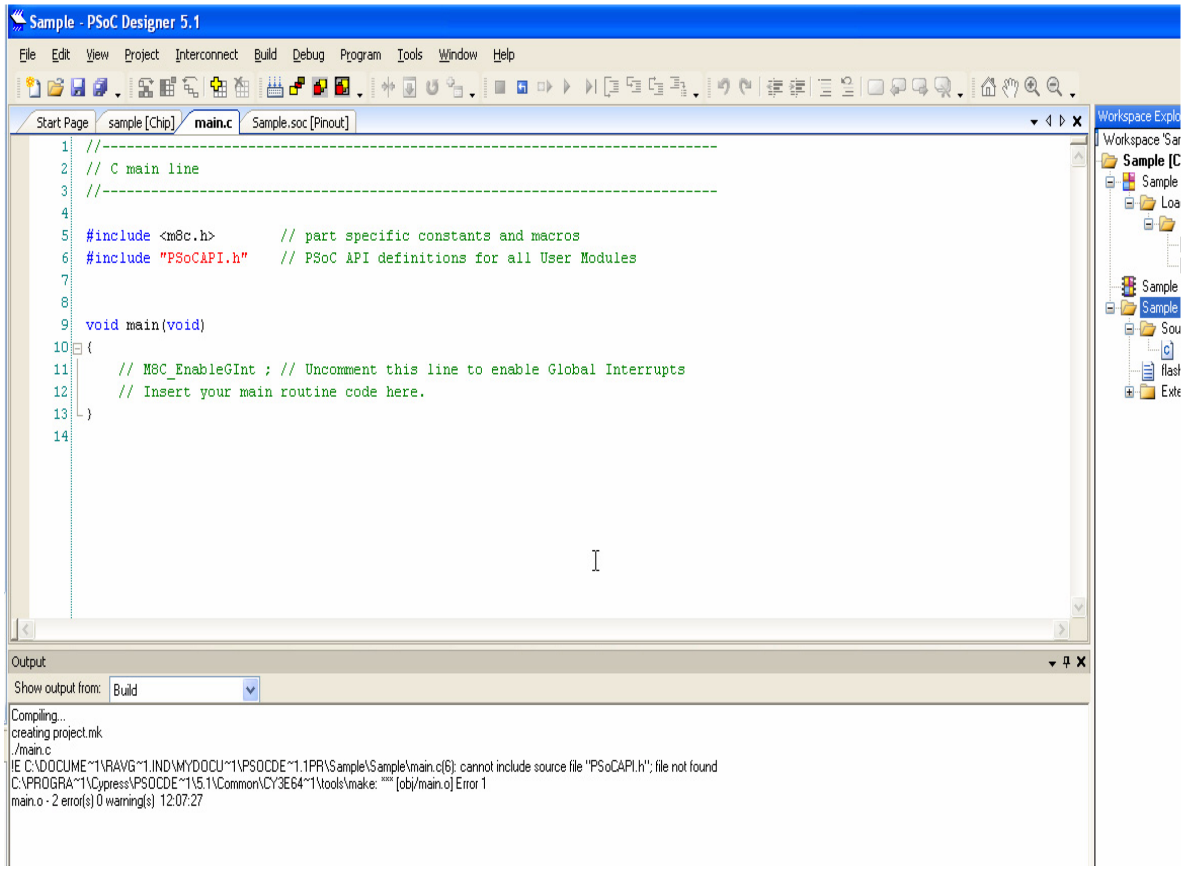
CapSense Block Total: 1 Used: 0
COMPARATOR Total: 2 Used: 0
I2C / SPI Block Total: 1 Used: 0
TIMER Block Total: 3 Used: 1
RAM Total: 2048 Used: 99
ROM Total: 16384 Used: 1040

Getting reserved resource info for TIMER0...

1.1.2 Code Editor

The code editor is a full-featured text editor designed for editing C and assembly code in your project. You can use the code editor to create and edit project files. You can rearrange the work area to suit your own work style.

Figure 1-2. PSoC Designer Code Editor



```
//----------------------------------------------------------------------------
// C main line
//----------------------------------------------------------------------------

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


void main(void)
{
    // M8C_EnableGInt ; // Uncomment this line to enable Global Interrupts
    // Insert your main routine code here.
}

```

Show output from: Build

```
Compiling...
creating project.mk
./main.c
!E C:\DOCUME~1\RAVG~1.IND\MYDOCU~1\PSOCDE~1.1PR\Sample\Sample\main.c(6): cannot include source file "PSoCAPI.h"; file not found
C:\PROGRA~1\Cypress\PSOCDE~1\5.1\Common\CY3E64~1\tools\make: *** [obj/main.o] Error 1
main.o - 2 error(s) 0 warning(s)  12:07:27
```

1.1.3 Build Manager

The Build Manager is a largely invisible utility that controls the various portions of the build process including the compiler (or compilers), assembler, and linker, and manages the process of building your project and preparing it to download to a target device.

The only visible portions of the Build Manager in the PSoC Designer application are the Build menu and the Build options in the Tools > Options... dialog. For more information on the Build Manger, see Appendix B.

1.1.4 Debugger

The debugger has multiple windows that allow you to interact with and observe the code execution on the target PSoC device. The debugger is tightly integrated with the rest of the IDE, and there is no separate debugger view. Instead, there are a number of different windows that you can use to monitor the status of various parts of your target board while debugging, including the following:

- Break Points
- Memory
- Watch Variables

- Events
- Trace
- Output

1.1.5 Getting Help

The Help menu contains several different options for obtaining more information about how to use PSoC Designer. The **Help -> Help Topics** window contains information about how PSoC Designer works. For additional information, the **Help -> Documentation...** selection opens a window showing all of the XLS, TXT and PDF documentation available with PSoC Designer, including this IDE Guide.

When you first launch PSoC Designer, the Start Page opens in the main application window. This start page contains panes that help you get started quickly using PSoC Designer.

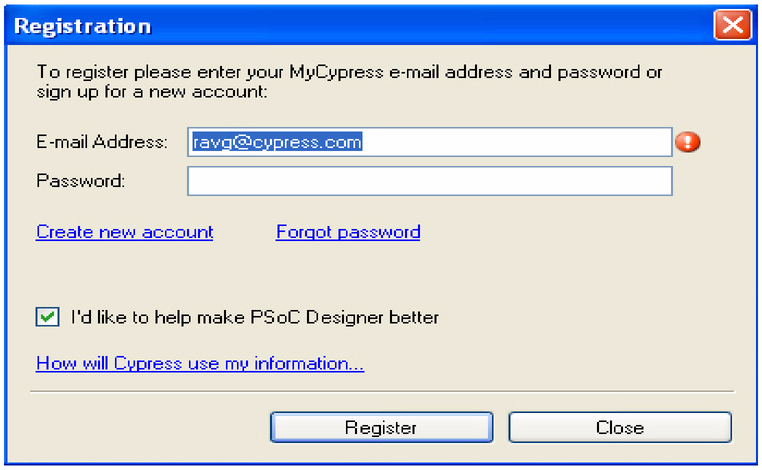
- **Recent Projects** allows you to open any previous saved project, create a new project, or browse to find projects that are not displayed in Recent Projects.
- **Design Catalog** allows you to choose among numerous preconfigured PSoC Designer designs. These are fully functioning PSoC Designer designs, many of which can be built and programmed on Cypress Evaluation Boards and Kits to give you full functioned examples.
- **PSoC Shortcuts** provides a shortcuts to PSoC resources that you may find helpful.

1.1.5.1 Register

After you install the software, the next step is to register the PSoC Designer.

To register your version of PSoC Designer, go to **Help** menu and click **Register**. The registration window opens.

Figure 1-3. Registration Window



Registration

To register please enter your MyCypress e-mail address and password or sign up for a new account:

E-mail Address: ravg@cypress.com

Password:

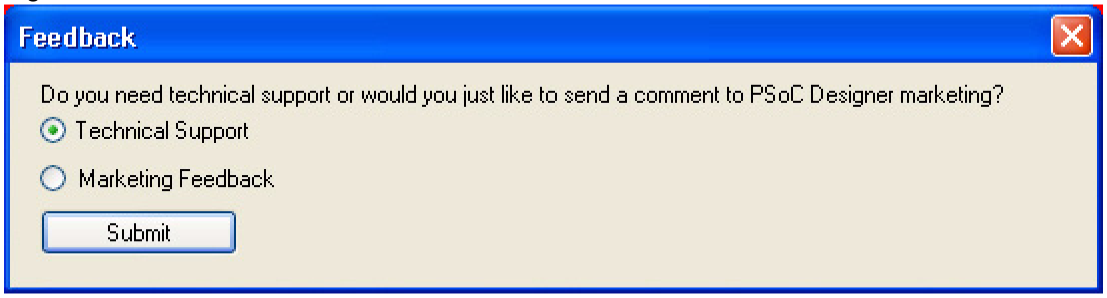
Create new account
Forgot password
I'd like to help make PSoC Designer better
How will Cypress use my information...
Register
Close

1. If you have an account with Cypress, enter your **Email Id** and **Password** else go to step 3.
2. To recover your password, click **Forgot password** link.
3. Click **Create new account** link to register with Cypress as a new user.
4. If you wish to enhance PSoC Designer, select **I'd like to help make PSoC Designer better** box.
5. In case of any privacy concern on how Cypress uses your information. Click **How will Cypress use my information...** link.
6. Click **Register**.

1.1.5.2 *Feedback*

If you need technical support or want to provide feedback about PSoC Designer, select **Help** menu and click **Feedback.** The feedback windows opens.

Figure 1-4. Feedback Window



Feedback

Do you need technical support or would you just like to send a comment to PSoC Designer marketing?

[✓ Technical Support]

Marketing Feedback

1. If you need any clarification on technical features, select **Technical Support**.
2. If your queries related to product information, select **Marketing Feedback**.
3. Click **Submit**.

1.2 Chapter Overviews

This table briefly describes the contents of each chapter in this guide.

Table 1-1. Chapter Overviews

Chapter	Description
Introduction	Describes the purpose of this guide, provides an application overview and descriptions of each chapter, supplies product support and upgrade information, and lists documentation conventions and references for more information.
Chip-Level Editor	Describes the chip-level editor that allows you to work directly with the resources available on a PSoC device, select and configure user modules, and route inputs, outputs, and other resources to and from them.
Code Editor	In this chapter you learn how to create the project code.
Assembler	In this chapter you receive high-level guidance on programming assembly language source files for the PSoC device.
Build Manager	In this chapter you learn the details of building a project, discover more about the C Compiler as well as the basic, transparent functions of the system Linker and Loader, and Librarian.
Debugger	In this chapter you learn how to download your project to the In-Circuit Emulator (ICE), use debug strategies, and program the part.
Flash Protection	Flash Program Memory Protection (FPMP) allows you to select one of four protection (or security) modes for each 64-byte block within the Flash, based upon the particular application.

1.3 Support

Go to http://www.cypress.com/?id=4 for free support. Resources include training seminars, discussion forums, application notes, PSoC consultants, TightLink technical support email/ knowledge base, and application support technicians.

Before using the Cypress support services, know the version of PSoC Designer installed on your system. To quickly determine the version of PSoC Designer, click **Help > About PSoC Designer**.

Cypress provides upgrades and version enhancements for PSoC Designer free of charge. The upgrades are downloadable from Cypress web under Software > PSoC Designer. Also provided are critical updates to system documentation under Design Support or go to http://www.cypress.com.

1.3.1 Technical Support Systems

Enter a support request into the TightLink Technical Support System with a guaranteed response time of four hours or view and participate in discussion threads about a wide variety of PSoC device topics at http://www.cypress.com/support/.

1.3.2 Product Upgrades

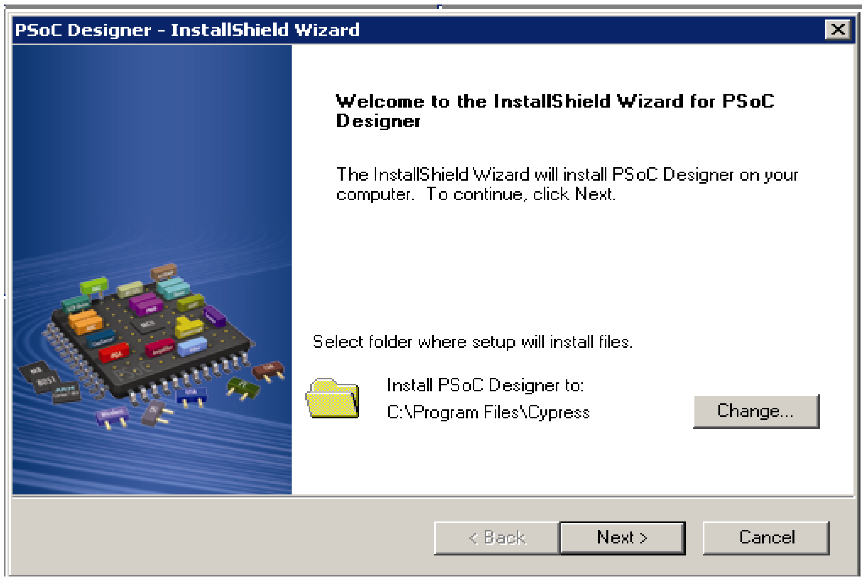
Cypress provides upgrades and version enhancements for PSoC Designer free of charge. The upgrades are downloadable from Cypress web under Software > PSoC Designer. Also provided are critical updates to system documentation under Design Support or go to http://www.cypress.com.

1.4 Installation

The PSoC Designer Installation wizard provides a step by step instruction to install PSoC Designer.

1. Select the Installation Wizard from **Start > All Programs > Cypress > PSoC Designer 5.1 > CyInstaller for PSoC Designer 5.1**. The welcome wizard appears.

Figure 1-5. Welcome Window



PSoC Designer - InstallShield Wizard

Welcome to the InstallShield Wizard for PSoC Designer

The InstallShield Wizard will install PSoC Designer on your computer. To continue, click Next.

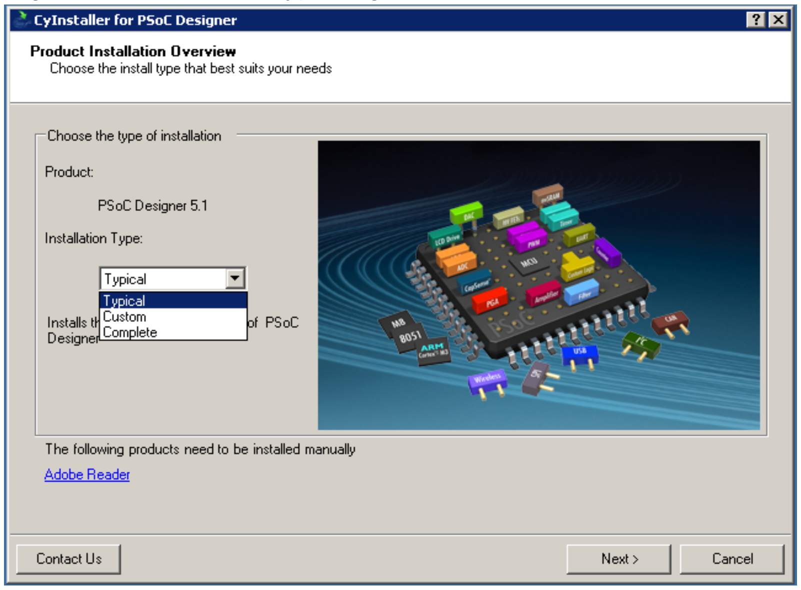
Select folder where setup will install files.

Install PSoC Designer to:
C:\Program Files\Cypress

2. Click **Change**... to change the location of the installation file.
3. Click **Next**.

4. "Choose the installtion type from the **Installation Type** Drop Down box.

Figure 1-6. Installtion Type Page



Product Installation Overview
Choose the install type that best suits your needs

Choose the type of installation

Product:

PSoC Designer 5.1

Installation Type:

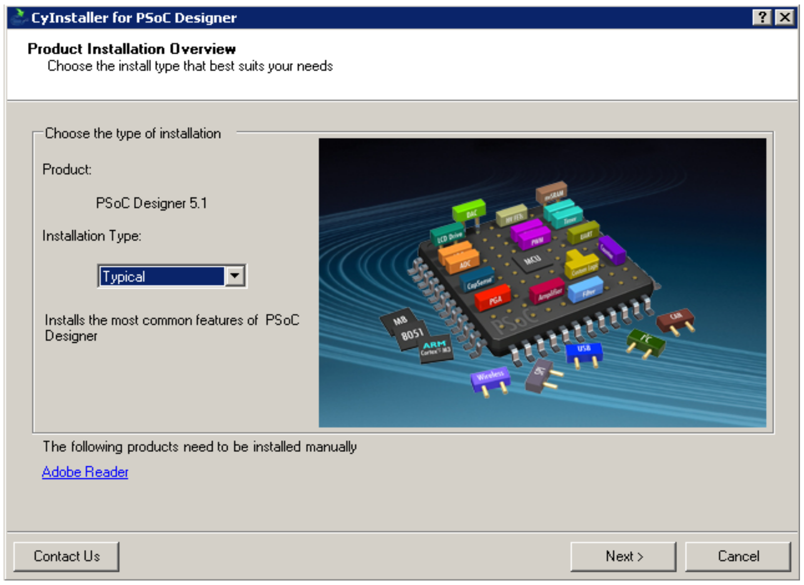
Typical

Installs th… of PSoC Designer

The following products need to be installed manually

5. If you select **Typical** installation type from the drop down box, the minimum set of files required gets installed.
6. If you select **Complete** option, the full package gets installed.
7. If you select **Custom** option, you can choose what exactly you want to install.

Figure 1-7. Select Typical Installation Type option



Product Installation Overview
Choose the install type that best suits your needs

Choose the type of installation

Product:

PSoC Designer 5.1

Installation Type:

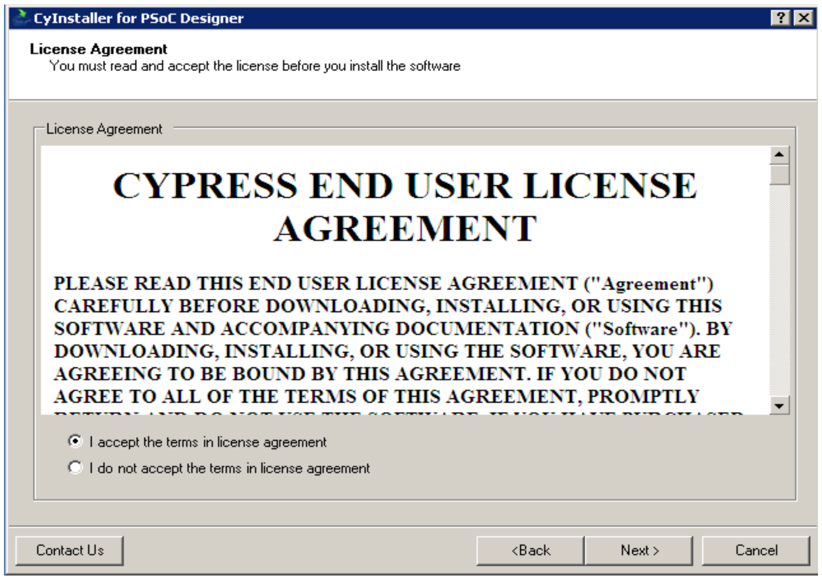
Typical

Installs the most common features of PSoC Designer

The following products need to be installed manually

8. Select **Typical** option from the Installtion Type drop down list, then click **Next**.

Figure 1-8. License Agreement



CyInstaller for PSoC Designer

License Agreement
You must read and accept the license before you install the software

License Agreement

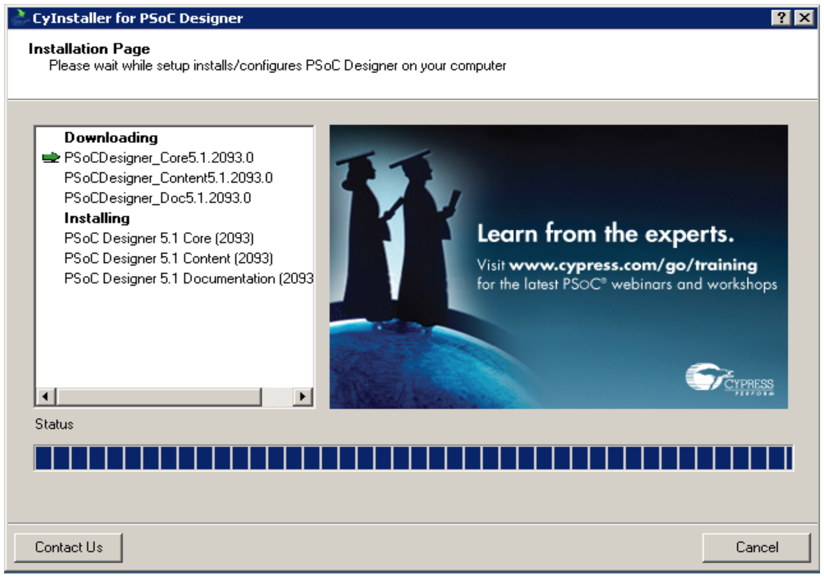
CYPRESS END USER LICENSE AGREEMENT

PLEASE READ THIS END USER LICENSE AGREEMENT ("Agreement") CAREFULLY BEFORE DOWNLOADING, INSTALLING, OR USING THIS SOFTWARE AND ACCOMPANYING DOCUMENTATION ("Software"). BY DOWNLOADING, INSTALLING, OR USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, PROMPTLY

[✓ I accept the terms in license agreement]
I do not accept the terms in license agreement

9. Select the **I accept the above license agreement** checkbox.
10. Click **Next**, to start the installation process

Figure 1-9. Downloads



CyInstaller for PSoC Designer

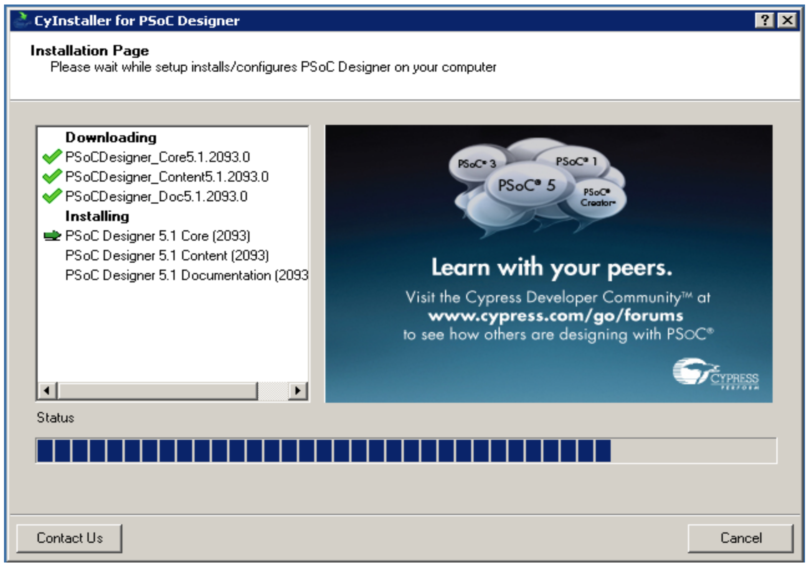
Installation Page
Please wait while setup installs/configures PSoC Designer on your computer

Downloading
PSoCDesigner_Core5.1.2093.0
PSoCDesigner_Content5.1.2093.0
PSoCDesigner_Doc5.1.2093.0
Installing
PSoC Designer 5.1 Core (2093)
PSoC Designer 5.1 Content (2093)
PSoC Designer 5.1 Documentation (2093

Learn from the experts.
Visit www.cypress.com/go/training
for the latest PSoC® webinars and workshops
CYPRESS
Status
Contact Us
Cancel

11. "You can see the downloading information such as Tool name, Tool Version and build number in the page. It downloads the complete build from the source. Then it starts installing the Designer.

Figure 1-10. Installation



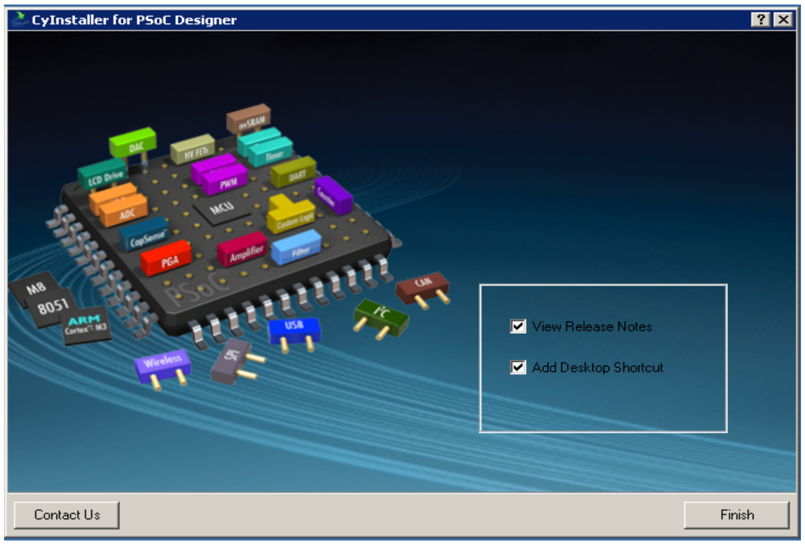
CyInstaller for PSoC Designer

Installation Page
Please wait while setup installs/configures PSoC Designer on your computer

Downloading
✓ PSoCDesigner_Core5.1.2093.0
✓ PSoCDesigner_Content5.1.2093.0
✓ PSoCDesigner_Doc5.1.2093.0
Installing
PSoC Designer 5.1 Core (2093)
PSoC Designer 5.1 Content (2093)
PSoC Designer 5.1 Documentation (2093

PSoC® 3
PSoC® 1
PSoC® 5
PSoC® Creator
Learn with your peers.
Visit the Cypress Developer Community™ at www.cypress.com/go/forums to see how others are designing with PSoC®
Status
Contact Us
Cancel

12. Click **Finish** to close the InstallShield Wizard.



☑ View Release Notes
☑ Add Desktop Shortcut

If you have any problem in installing PSoC Designer. Click **Contact Us**.

1.5 Conventions

Here are the conventions used throughout this guide.

Table 1-2. Documentation Conventions

Convention	Usage
Courier New	Displays file locations and source code: `C:\ …cd\icc\, user entered text`
Italics	Displays file names and reference documentation: *sourcefile.hex*
[bracketed, bold]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
File > New Project	Represents menu paths: File > New Project > Clone
Bold	Displays commands, menu paths and selections, and icon names in procedures: Click the **Debugger** icon, and then click **Next**.
Text in gray boxes	Displays cautions or functionality unique to PSoC Designer or the PSoC device.

1.5.1 Acronyms

These are the acronyms used throughout this guide.

Table 1-3. Acronyms

Acronym	Description
ADC	analog-to-digital converter
API	application programming interface
BOM	bill of material
C	(refers to the C programming language)
DAC	digital-to-analog converter
DRC	design rule checker
EPP	enhanced parallel port
FPMP	Flash program memory protection
grep	global regular expression print
ICE	in-circuit emulator
IDE	integrated development environment
IO	input/output
ISR	interrupt service routine
MCU	microcontroller unit
MHz	megahertz
OBM	on-board monitor
OHCI	open host controller interface
PWM	pulse width modulator
RAM	random access memory
ROM	read only memory

Table 1-3. Acronyms *(continued)*

Acronym	Description
SSC	system supervisory call
UART	universal asynchronous receiver transmitter
UHCI	universal host controller interface
USB	universal serial bus

1.6 References

This guide is part of a larger documentation suite for the PSoC Designer application. It is meant as a reference, not as the complete source of information. For the most up-to-date information, go to http://www.cypress.com. The documentation listed here provides more specific information on a variety of topics:

- PSoC Designer Help Topics (Online Help)
- PSoC Designer Development Kit Getting Started Guide
- PSoC Programmer Guide
- Various PSoC Designer application notes and data sheets

1.7 Revision History

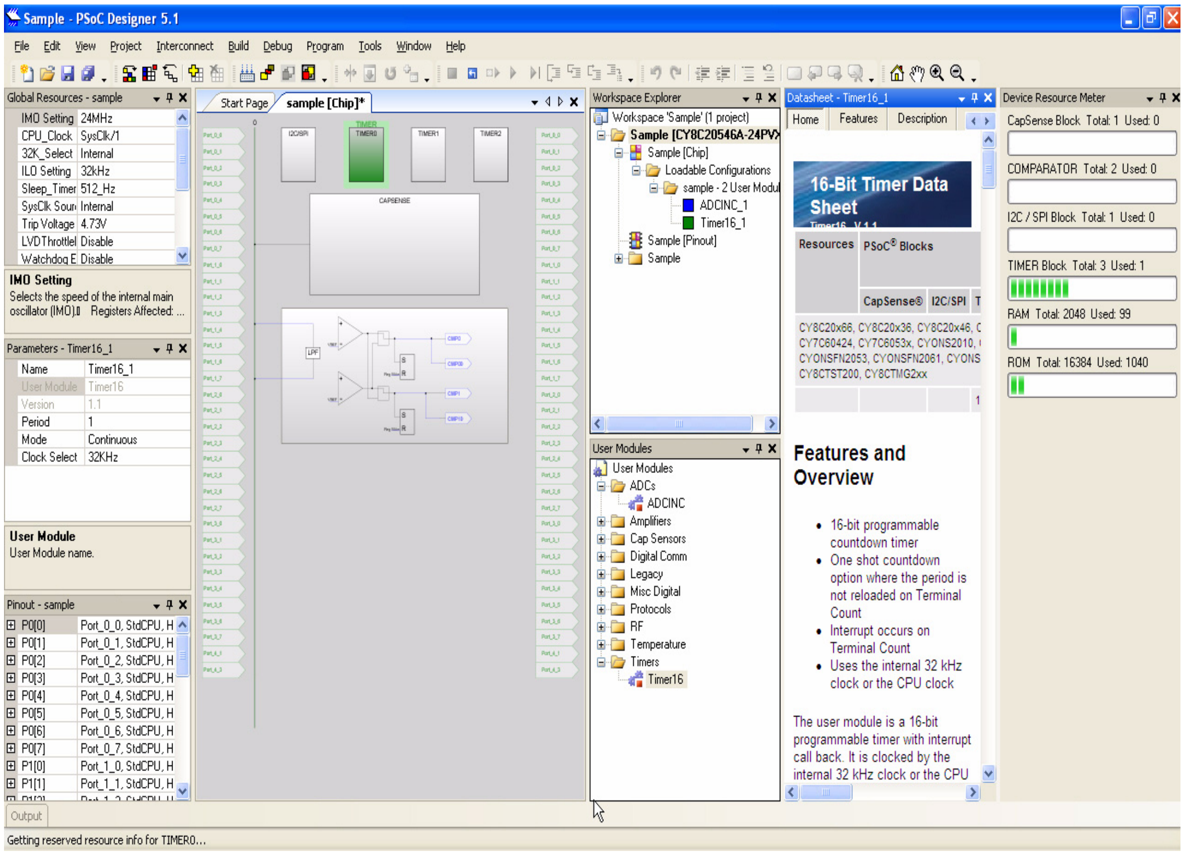
Table 1-4. Revision History

Revision	PDF Creation Date	Origin of Change	Description of Change
**	May 27, 2008	FSU	Put changes to the original PSoC Designer IDE Guide in a new template and assigned a Spec Number.
*A	August 14, 2008	FSU	Changed some screen captures. Added many previously undocumented global parameters.
*B	March 24, 2009	PYRS	Added content relating to compilers and large scale updates
*C	July 14, 2010	FSU/ RAVG	Removed Sytsem Level Editor chapter and other large scale updates. Added Register, Feedback, CyInstaller for PSoC Designer 5.1 and Selecting User Module-section
*D	12/09/2010	RAVG	Comprehensive updates of all chapters and the updates are based on CDTs

2. Chip-Level Editor

The Chip-Level Editor allows you to work directly with the resources available on a PSoC device, select and configure user modules, such as analog to digital converters (ADCs), timers, amplifiers, and others, and route inputs, outputs, and other resources to and from them.

Figure 2-1. Chip-Level Editor Desktop



IMO Setting	24MHz
CPU_Clock	SysClk/1
32K_Select	Internal
ILO Setting	32kHz
Sleep_Timer	512_Hz
SysClk Sourc	Internal
Trip Voltage	4.73V
LVDThrottle	Disable
Watchdog E	Disable

IMO Setting
Selects the speed of the internal main oscillator (IMO). Registers Affected: ...

Name	Timer16_1
User Module	Timer16
Version	1.1
Period	1
Mode	Continuous
Clock Select	32KHz

User Module
User Module name.

P0[0]	Port_0_0, StdCPU, H
P0[1]	Port_0_1, StdCPU, H
P0[2]	Port_0_2, StdCPU, H
P0[3]	Port_0_3, StdCPU, H
P0[4]	Port_0_4, StdCPU, H
P0[5]	Port_0_5, StdCPU, H
P0[6]	Port_0_6, StdCPU, H
P0[7]	Port_0_7, StdCPU, H
P1[0]	Port_1_0, StdCPU, H
P1[1]	Port_1_1, StdCPU, H

16-Bit Timer Data Sheet

Resources	PSoC® Blocks		
	CapSense®	I2C/SPI	T
CY8C20x66, CY8C20x36, CY8C20x46, C CY7C60424, CY7C6053x, CYONS2010, CYONSFN2053, CYONSFN2061, CYONS CY8CTST200, CY8CTMG2xx			
			1

Features and Overview

- 16-bit programmable countdown timer
- One shot countdown option where the period is not reloaded on Terminal Count
- Interrupt occurs on Terminal Count
- Uses the internal 32 kHz clock or the CPU clock

The user module is a 16-bit programmable timer with interrupt call back. It is clocked by the internal 32 kHz clock or the CPU

CapSense Block Total: 1 Used: 0
COMPARATOR Total: 2 Used: 0
I2C / SPI Block Total: 1 Used: 0
TIMER Block Total: 3 Used: 1
RAM Total: 2048 Used: 99
ROM Total: 16384 Used: 1040

Getting reserved resource info for TIMER0...

2.1 Chip-Level Editor Overview

The Chip-Level Editor gives you complete control over Chip-Level Projects resource use, routing, and firmware. You choose a specific chip at the beginning of this process:

Step 1: Create a Project

This is the first step in both processes, but after naming your project, the first thing that you do in a Chip-Level Project is select a PSoC device.

Step 2: Select a PSoC Device

There are a large number of PSoC devices in the PSoC architecture with more being added all the time. Some are general purpose devices with varying amounts of general purpose digital and analog resources while others are more specialized with onboard peripherals suited to specific solutions such as wireless, LED control, or capacitive sensing. Consult the Cypress web site for a wide variety of literature and contact information for people that can help you choose the right device for your design.

Step 3: Choose User Modules

PSoC devices have programmable analog and digital blocks that can be configured for a wide variety of uses. User Modules configure these programmable blocks to behave as a specific peripheral, such as an analog to digital converter, a timer, or a pulse width modulator. You choose user modules based on what you need the PSoC device to do for you.

Step 4: Configure the User Modules

Each user module has a set of parameters that allow you to configure it to meet your needs. For example, a CapSense user module must be configured to detect signals coming from capacitive sensing components in a wide variety of configurations, so it has a large number of configurable parameters. A design rule checker can alert you to potential problems with your design as you work.

Step 5: Connect the User Modules

Each user module will have inputs, outputs, and interrupts that can be routed internally to and from other user modules, and externally to and from pins. The PSoC devices have a very flexible routing system, but resources are limited and it may take some experimentation to find the optimal routing and placement for all of the user modules.

Step 6: Generate Your Project

This prepopulates your project with APIs and libraries that you can use to program your application.

Step 7: Write Your Program

Write your program in C for rapid development, assembly language to get every last drop of performance from the MCU, or a combination of both. You have a choice of third party C compilers for PSoC devices.

Step 8: Build and Debug Your Program

Build and test your program. Use PSoC Designer in conjunction with one of the PSoC emulators. PSoC Designer has a powerful built in debugger.

Step 9: Program the Device

Cypress has a variety of programmers that you can use to program your production parts.

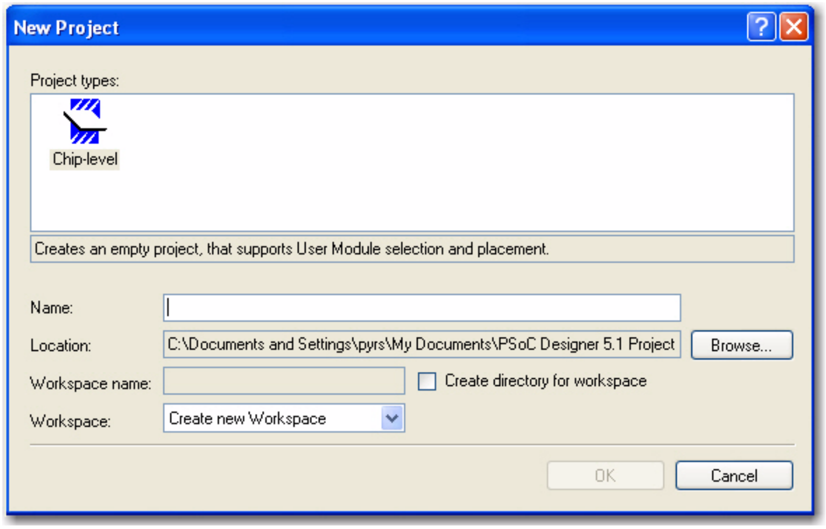
Your design is now complete. The remainder of this chapter is organized just like the above outline with additional details on each of the steps.

2.2 Create a Project

In order to program the desired functionality into a PSoC device, you need to first create a project directory in which the files and device configurations reside.

1. To start a new project, select **New Project...** from the **File** menu.

Figure 2-2. New Project Dialog Box



New Project

Project types:

Chip-level

Creates an empty project, that supports User Module selection and placement.

Name:

Location: C:\Documents and Settings\pyrs\My Documents\PSoC Designer 5.1 Project

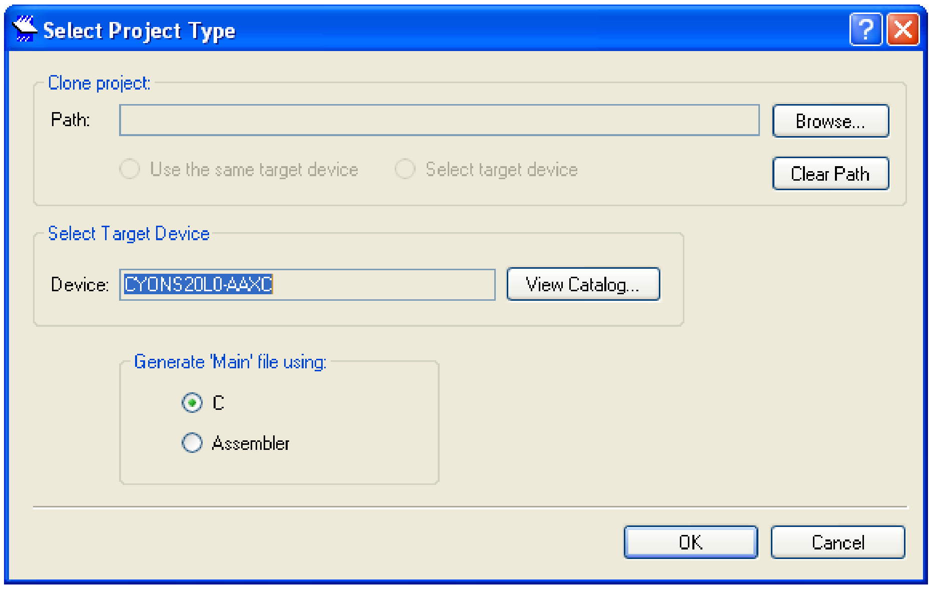
Workspace name:

[ ] Create directory for workspace

Workspace: Create new Workspace

2. Choose a name and location for the project. By default, a project is created inside a workspace with the same name as the project, the project is stored in the project directory. If you plan to create multiple projects in a single workspace (for example, if your project will use multiple PSoC devices), click **Create a directory for workspace** and supply a name for the first project. The Workspace menu provides you the option to Create a new Workspace or to Add to an existing workspace. When you are finished, click **OK.**
3. In the Select Project Type dialog box, click **View Catalog...** to access a detailed list of available parts.

Figure 2-3. Select Project Type Dialog Box



Select Project Type

Clone project:

Path:

( ) Use the same target device ( ) Select target device

Select Target Device

Device: CY8C20L0-AAXC

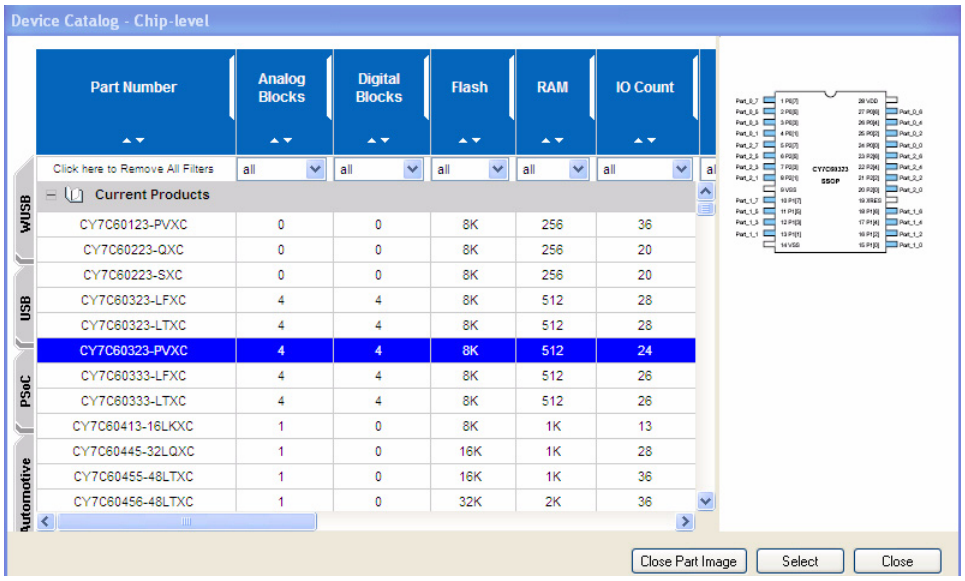
Generate 'Main' file using:

[✓ C]

Assembler

4. In the Device Catalog Dialog Box, highlight your part of choice. Tabs at the left and characteristic selections along the top narrow the list of devices. You have several options in this dialog box including layout display, viewing part image, and sorting part selection (by clicking on a chosen column title). Click **Select** to save your selection and exit the dialog box.

Figure 2-4. Device Catalog Dialog Box



Part Number	Analog Blocks	Digital Blocks	Flash	RAM	IO Count
CY7C60123-PVXC	0	0	8K	256	36
CY7C60223-QXC	0	0	8K	256	20
CY7C60223-SXC	0	0	8K	256	20
CY7C60323-LFXC	4	4	8K	512	28
CY7C60323-LTXC	4	4	8K	512	28
CY7C60323-PVXC	4	4	8K	512	24
CY7C60333-LFXC	4	4	8K	512	26
CY7C60333-LTXC	4	4	8K	512	26
CY7C60413-16LKXC	1	0	8K	1K	13
CY7C60445-32LQXC	1	0	16K	1K	28
CY7C60455-48LTXC	1	0	16K	1K	36
CY7C60456-48LTXC	1	0	32K	2K	36

5. After you select a part, click **C** or **Assembler**, in the Select Project Type dialog box, to designate the language in which you want the system to generate the “main” file.
6. Click **OK**. Your workspace directory with folders is created and is listed in the Workspace Explorer. If the Workspace Explorer is not visible, choose **Workspace Explorer** from the **View** menu.

2.2.1 Clone a Project

Cloning a project is used when you want to convert an existing project to a different PSoC part. The part is referred to as the “base” part.

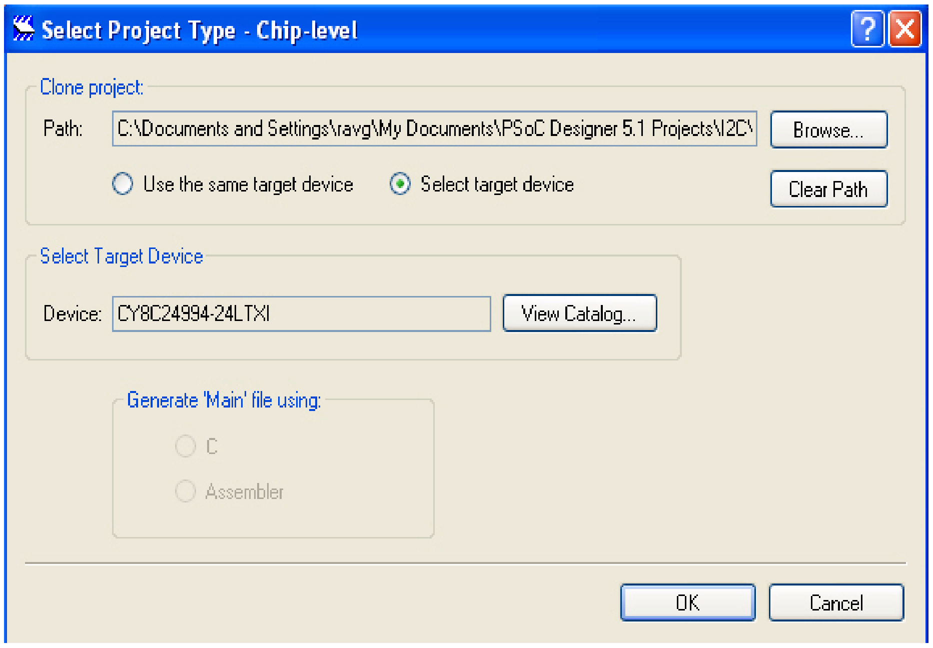
You can clone an existing project at any point of its existence: before, during, or after device configuration, assembly-source programming, or project debugging. Cloning copies the existing project but allows you to change the base device. Use the cloning method to move an existing project from one directory to another, rather than physically moving the files.

You must use the cloning method to change parts within a part family in the middle of a project design. Refer to the Application Notes on the Cypress web site for assistance.

To clone an existing project:

1. From the **File** menu, select **New Project...** You can only clone a Chip-Level project. Select **Chip-level** .
2. Select a name and location for your new application and click **OK**.
3. In the Clone project box click **Browse...** and find the .CMX file of the project you want to clone.
4. You have two radio button “**Use the same target device**” and “**Select target device**”.
5. Choose “**Use the same target device**” radio button to use the same device or choose “**Select target device”** to use new base device.
6. Select **View Catalog...** to select a new device and Click **OK**.

Figure 2-5. Cloning a Project


Select Project Type - Chip-level

Clone project:

Path: C:\Documents and Settings\ravg\My Documents\PSoC Designer 5.1 Projects\I2C\

( ) Use the same target device (●) Select target device

Select Target Device

Device: CY8C24994-24LTXI

Generate 'Main' file using:

( ) C

( ) Assembler

7. If you choose to use the same target device, select Use the same target device radio button and click **OK**.

The clone analysis window appears only if you choose a different target device.

2.2.2 Updating Existing Projects

If you download a newer version of PSoC Designer, you may need to update existing projects created with an earlier version of PSoC Designer. Most project updates are done automatically; however, some need to be done manually depending upon project specifics. Manual project updates are described at the end of this chapter.

To update a PSoC project for compatibility:

1. Open PSoC Designer.
2. Access the project to update.
   At this point, PSoC Designer checks to see if the project is compatible with the new version of PSoC Designer.
3. If your project needs to be updated, the Old Version window appears with the appropriate message text. Click **Update** (or you can update later by selecting **Update later...**).
4. After the update is complete, click **Finish.** Your project is now compatible with the current version of PSoC Designer.

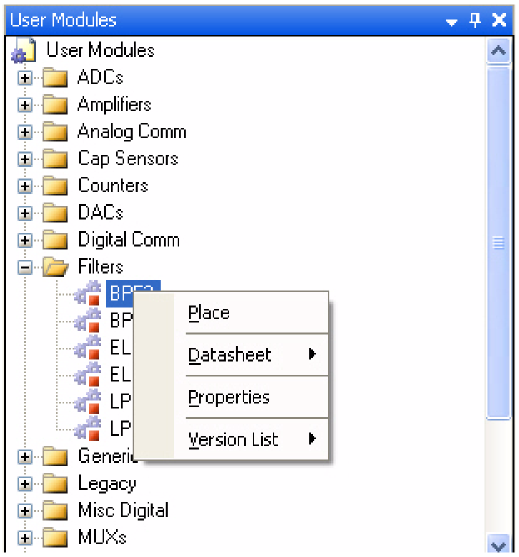
2.3 Selecting User Module

The user modules list is populated in the User Module Window depending on the selection of device you make in the Device Catalog list.

Select a User Module from the **User Modules** Window after you create a new project. A user module is a preconfigured function that is placed and is programmed. It works as a peripheral on the target device.

PSoC Designer provides you multiple ways to select the User module from the User Modules Window.

1. Select the user module you need from the User Modules Window. **Double** click on the selected user module. The selected user module is added into the block placed inside the chip editor.
2. Select the user module you need from the User Modules Window. **Right** click on the selected user module. The pop menu appears. Select **Place** menu. The selected user module is added into the block placed inside the chip editor.



3. Select the user module you need from the **User Module** Window. **Drag** and **drop** inside the Chip Editor window. The user module is added into the block placed inside chip editor window.

Repeat the process for adding each user module in the **Chip Editor** window. You can add or remove the user modules from the chip editor at any time during the device configuration.

Select **View** > **Datasheet Window** from the menu, to view the user modules in a separate window.

2.4 Selecting Multiuser Module

The “Multiple” User Modules (MUM) are designed to support selection between different User Module configurations. The Multi User Module (MUM) selection dialog appears during User Module placement in the Chip Editor, which is specific to the User Module.

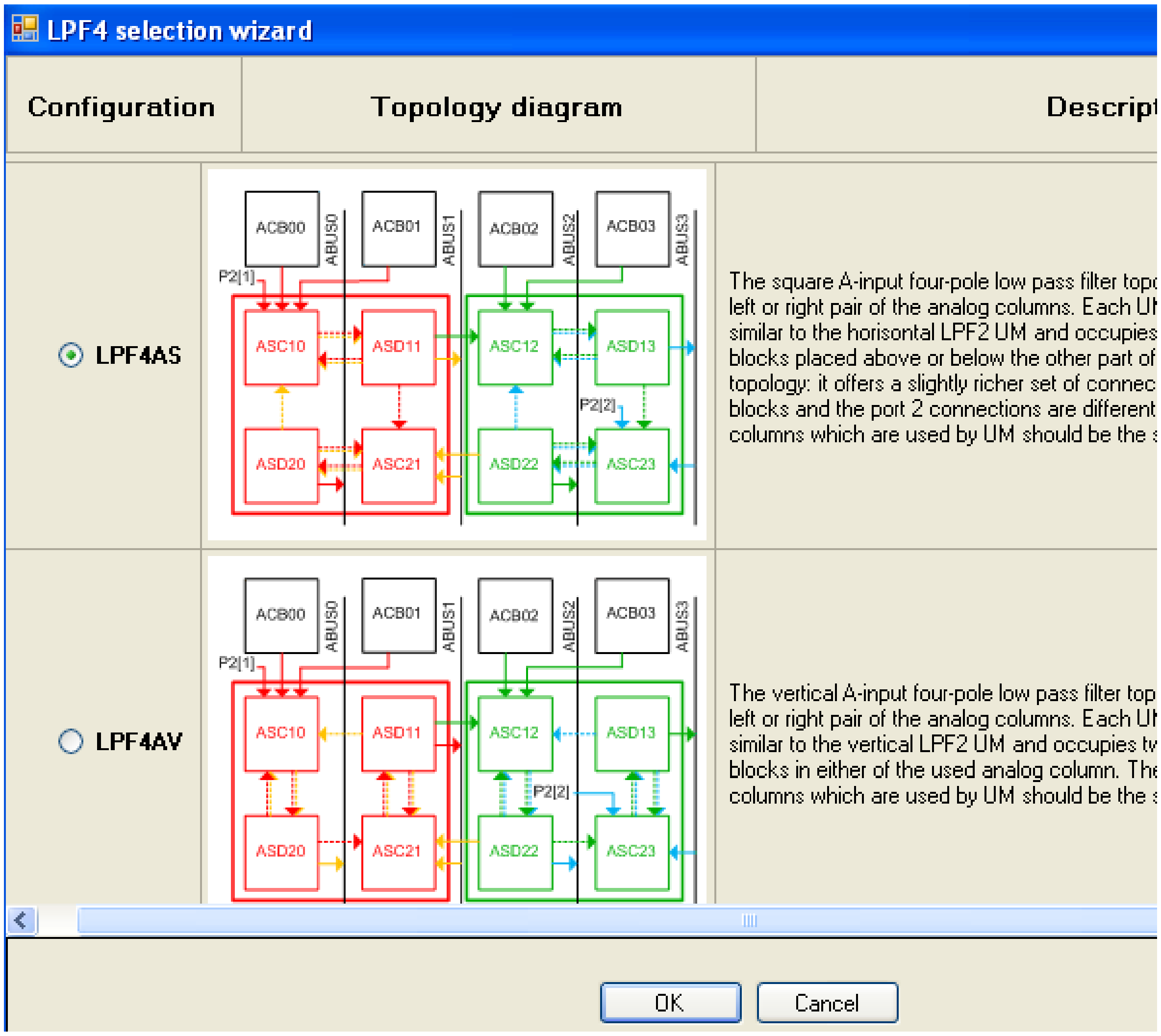
You can select the User Module that suits best to your needs.

The following examples describe the different types of Multi User Module.

- Supported topologies
- Different operation modes
- Available resource usages / CPU performance
- Single Stage / Double Stage Modulator

Select **CY8C29466-24PVXI** device from the catalog. Drag and drop LPF4 filter component to the Chip Editor. The Select Multi User Module dialog appears.

Example 1: Choose between supported topologies



[✓ LPF4AS]

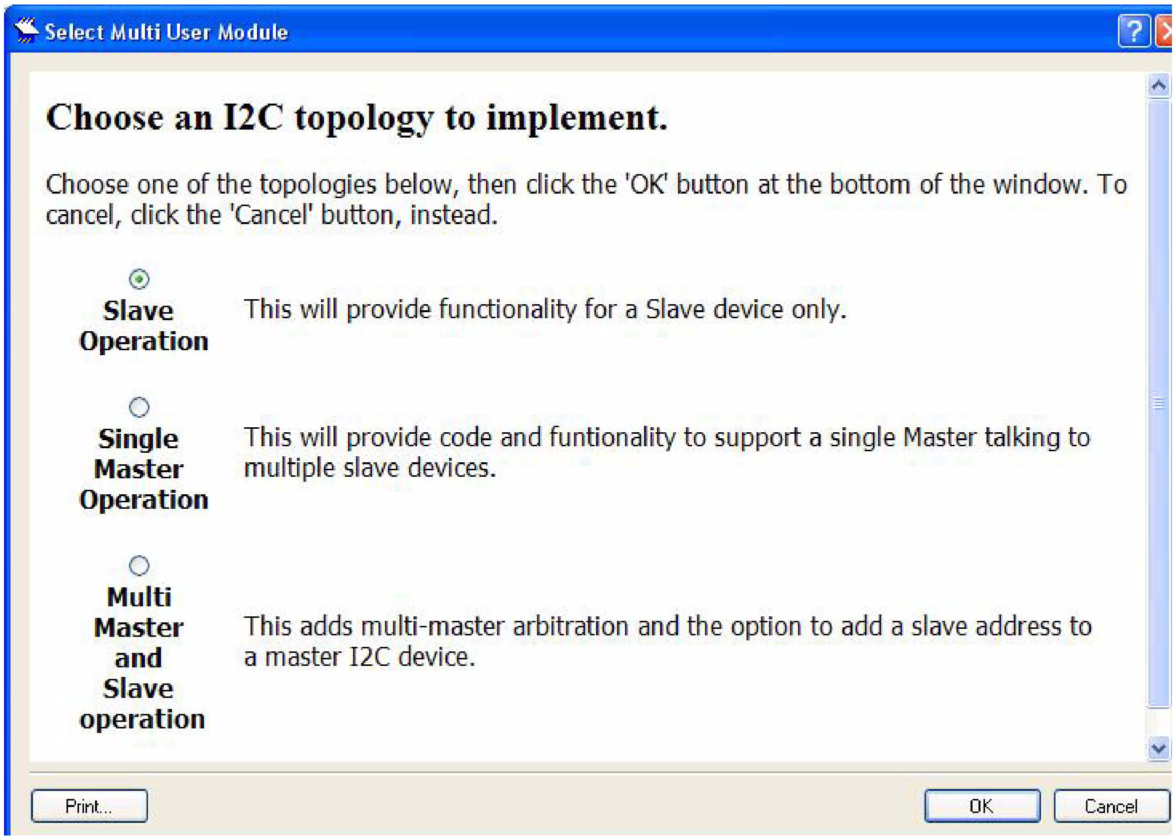
The square A-input four-pole low pass filter topo
left or right pair of the analog columns. Each UI
similar to the horisontal LPF2 UM and occupies
blocks placed above or below the other part of
topology: it offers a slightly richer set of connec
blocks and the port 2 connections are different
columns which are used by UM should be the s

LPF4AV

The vertical A-input four-pole low pass filter top
left or right pair of the analog columns. Each UI
similar to the vertical LPF2 UM and occupies tw
blocks in either of the used analog column. The
columns which are used by UM should be the s

The input signals routing and the occupied analog blocks depend on selected option. For this User Module the selection will not affect functionality and performance. The selection will affect the topology only.

Example 2: Choose Operation Mode



Select Multi User Module

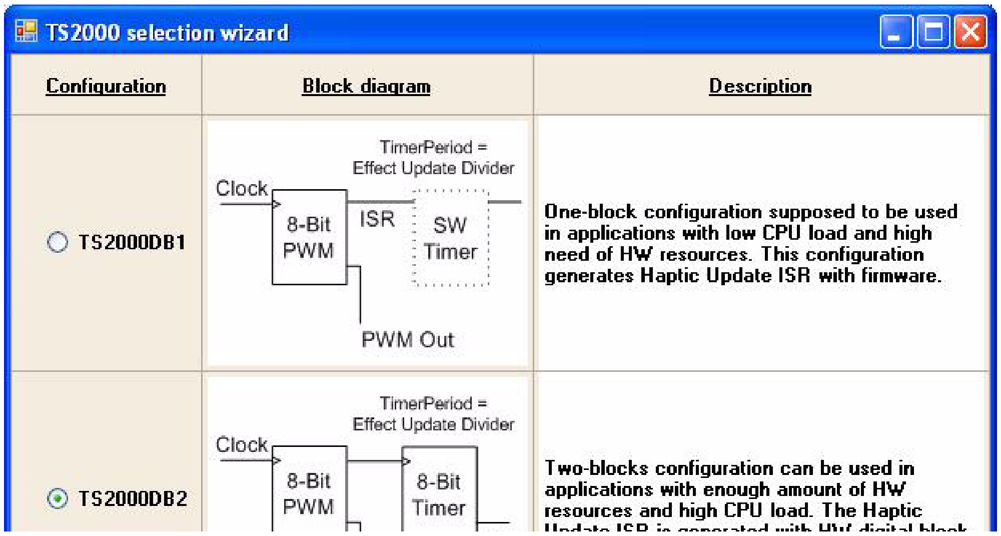
Choose an I2C topology to implement.

Choose one of the topologies below, then click the 'OK' button at the bottom of the window. To cancel, click the 'Cancel' button, instead.

[✓ Slave Operation]	This will provide functionality for a Slave device only.
Single Master Operation	This will provide code and funtionality to support a single Master talking to multiple slave devices.
Multi Master and Slave operation	This adds multi-master arbitration and the option to add a slave address to a master I2C device.

This dialog allows selection of functionality to be obtained from hardware I2C resource. The functionality is totally different from the other. Depending on the selection, the hardware I2C resource is configured to one of available operation modes and proper libraries are created for use in application code.

Example 3: Choosing between Resource Usage / CPU Performance

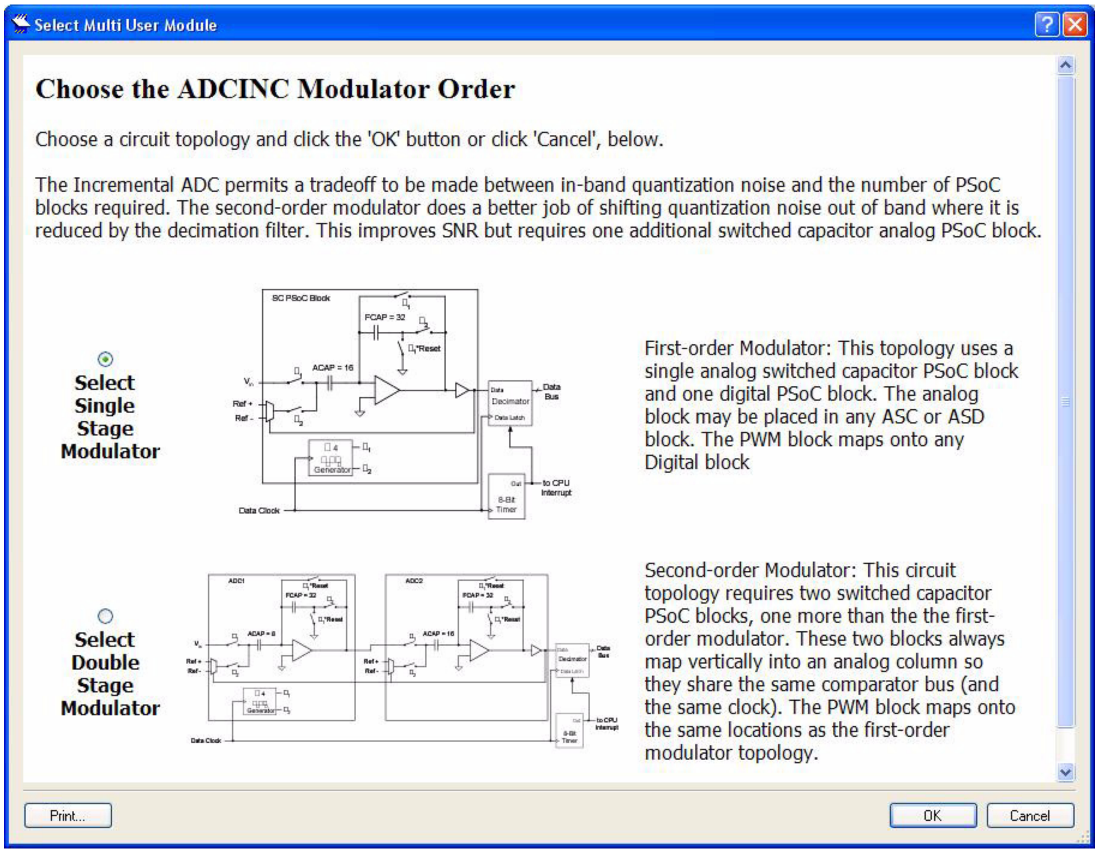


TS2000 selection wizard

Configuration	Block diagram	Description
TS2000DB1	TimerPeriod = Effect Update Divider; Clock; 8-Bit PWM; ISR; SW Timer; PWM Out	One-block configuration supposed to be used in applications with low CPU load and high need of HW resources. This configuration generates Haptic Update ISR with firmware.
[✓ TS2000DB2]	TimerPeriod = Effect Update Divider; Clock; 8-Bit PWM; 8-Bit Timer	Two-blocks configuration can be used in applications with enough amount of HW resources and high CPU load. The Haptic

The first option here saves one digital block but it consumes extra CPU time for servicing the software timer. For the second option the hardware digital block is used as a timer.

Example 4: Single Stage / Double Stage Modulator



Choose the ADCINC Modulator Order

Choose a circuit topology and click the 'OK' button or click 'Cancel', below.

The Incremental ADC permits a tradeoff to be made between in-band quantization noise and the number of PSoC blocks required. The second-order modulator does a better job of shifting quantization noise out of band where it is reduced by the decimation filter. This improves SNR but requires one additional switched capacitor analog PSoC block.

[✓ Select Single Stage Modulator]

First-order Modulator: This topology uses a single analog switched capacitor PSoC block and one digital PSoC block. The analog block may be placed in any ASC or ASD block. The PWM block maps onto any Digital block

Select Double Stage Modulator

Second-order Modulator: This circuit topology requires two switched capacitor PSoC blocks, one more than the the first-order modulator. These two blocks always map vertically into an analog column so they share the same comparator bus (and the same clock). The PWM block maps onto the same locations as the first-order modulator topology.

In the ADCINC Multi User Module dialog, you can choose “Single-Stage modulator” to save analog switched capacitor block or you can choose “Double-Stage modulator” to obtain higher SNR level of application. You can choose either one of the topology that suits your requirements.

2.5 Placing User Modules

Placing user modules is the first step (after creating a project) in configuring your target device. A user module is a preconfigured function that, after placed and programmed, works as a peripheral on the target device.

To place a user module:

1. Locate the desired user module in the the user module catalog. Each user module has a user module data sheet that describes what it is and how to use it. If you do not see the user module data sheet when you click on a user module, select **View > Datasheet Window**. Right-click on the user module and select **Place**. Some user modules have wizards or configuration screens that appear before the user module can be placed. These will differ by user module. The user module will be placed in the first available PSoC block in the **Chip Editor** view.

   The user module block reference names appear above the currently active blocks. For example, an ADC10 has one digital block used as a counter (CNT) and two analog blocks, one for the analog to digital conversion (ADC) and the other for a voltage ramp (RAMP). The name of the user module is separate from these user module block function names. This is because a multiblock user module may have distinct block actions.
2. If you want to use a placement other than the default, click the **Next Allowed Placement** icon to advance the user module to the next available location (active/anchor identified as green, non-

active as blue) and then select required user mosule and place. Click the target placer (identified as green and blue highlights) and drag-and-drop the user module to a new location. If a user module has multiple blocks, it may be possible to drag individual blocks onto a free block. Repeat this procedure until you have identified the exact location for the user module.

The Next Allowed Placement button shows the next possible set of PSoC blocks in which a user module may be placed, regardless of any currently placed user modules. If you cannot place the user module in the highlighted location due to a lack of resources, a Resource Allocation message flashes in the lower-left corner of PSoC Designer. Placement is not possible if another user module occupies the PSoC block, or if a placed user module is using another resource which the highlighted user module requires.

3. When you identify the location, click the **Place User Module** icon, or right-click and select **Place**.

   After you place the module, it appears on the device, color-coded, bearing the designated name of the chosen PSoC block. In the Interconnect frame, the inactive target placers (blue highlights) of multi-block user modules are now identified by a group name across the top.

   Some user modules do not consume visible resources in the Chip Editor view when they are placed. Examples of this include LCD, I^{2}C Master, I^{2}C Slave and software only user modules.
4. At this time, you can print, save, clear or unplace, and name or rename the placed user module.
   - To print your placement view, right-click anywhere in the Chip Editor view and select **Print**.
   - To save your work, click **File > Save Workspace**.
   - To clear all user module placements (i.e., remove them from their location on the PSoC blocks), click **Interconnect > Clear All Placements**. To unplace one particular module, right-click it (in either the Interconnect view or the Workspace Explorer) and select **Unplace** or click the **Unplace User Module** icon. This does not remove user modules from PSoC Designer or from your collection. Your unplaced user modules shown in the Workspace Explorer under **Interconnect > Loadable Configurations > User Modules**.
   - To name or rename user modules, select the user module either in the Workspace Explorer or the Chip Editor view, and type a new name in the user module Parameters window.
5. Repeat this process (steps 1-4) for all user modules in your design.

For each user module you add, the system updates the data in the Resource Meter with the number of occupied PSoC blocks, along with estimated RAM and ROM usage for the current set of selected user modules. The RAM and ROM numbers grow or shrink depending upon wizard settings and other user module parameter adjustments. If you select a user module that requires more resources than are currently available, PSoC Designer does not allow the selection. If you do not see the Device Resource Meter go to the **View** menu and select **Resource Meter**.

If user modules are already placed, then there are some cases when user module placement fails even if it appears that sufficient PSoC blocks remain unallocated. In such cases, the already placed user modules are using resources that the selected user module requires.

There are several user modules that require topology selection (that is filters). Right click on the module in the Workspace Explorer after it is placed and click **Selection Options**. Make the topology choice according to your application.

Some user modules have associated wizards to assist in configuration. To access a wizard, select the user module in the Workspace Explorer and then right click the mouse. If a wizard exists, it appears in the menu choices.

To Remove a User Module:

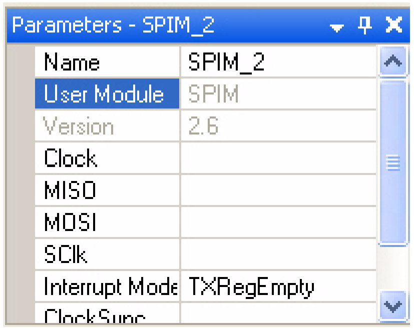
To remove user modules from your collection, select any block of the user module in the Chip Editor and press [**Delete**], or right-click the user module and select **Delete**. This does not remove user modules from PSoC Designer, just from your collection.

If you add or remove user modules after generating application files, you need to regenerate the application files (as well as reconfigure required settings). For further details, see "Generating Application Files" on page 56.

2.5.1 Setting User Module Parameters

Setting User Module Parameters configures the user module to behave the way you want it to and connects the user module to the external pins and other user modules and resources. You connect to user modules through the output and input parameters of the PSoC blocks. The interconnection buses provide interconnection paths between the external pins and to other digital user modules.

After you place the user module, the parameters are updated with applicable names. When you single-click a user module, you view its parameters under parameters. If you do not see the Pameters window, go to the View menu and select Parameters Window. User Module Parameters


Parameters - SPIM_2

Name	SPIM_2
User Module	SPIM
Version	2.6
Clock	
MISO	
MOSI	
SClk	
Interrupt Mode	TXRegEmpty
ClockSync	

To update the User Module Parameters:

1. Click each drop arrow (in parameter value fields) and make your selections.

   Some parameters are integer values. Set these values by clicking the up/down arrows, or double-click the value and type in the value. You can set the values as minimum or maximum.
2. Repeat this process for all placed user modules.

2.5.2 Global Resources

Global Resources are hardware settings that determine the underlying operation of the part (for the entire application). For example, the CPU_clock designates the clock speed of the M8C.

Note that Global Resource options differ slightly for each device family. The registers for enCoRe II apply to CY7C63x23, CY7C638xx and CY7C63310.

To update and save Global Resources:

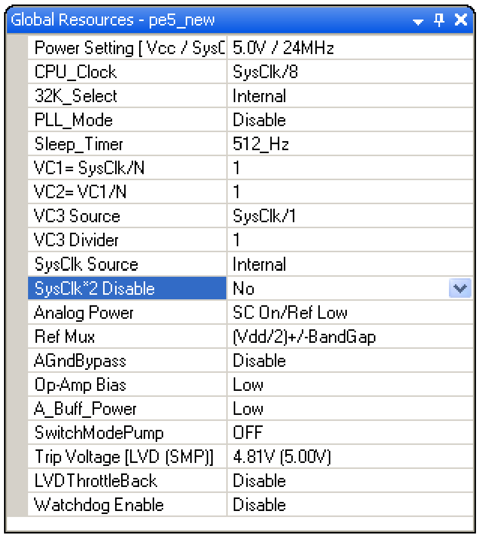
1. Click each drop-arrow (in parameter value fields) and make your selections.

   Some parameters in Global Resources are specified as integer values (such as VC1 and VC2). Set these values by clicking the up/down arrows or double-clicking the value and typing over them. You can set minimum or maximum value in the Global Resources dialog box. Click **OK** to close the dialog box.
2. The current settings for the Global Resources can be saved as default settings. Right-click on any Global Resource name and select **Update Default Values**.

Use these settings for any other project by right-clicking any Global Resource name and selecting **Restore Default Values**. If no custom default values are saved, then the menu item and the right-click to Restore Default Settings restore the factory default Global Resource Settings.

The Global Resources available in PSoC Designer are shown and described briefly. Different PSoC Devices have different global resources. The figure shown is typical.

Figure 2-6. Global Resources Example.



Global Resources - pe5_new

Power Setting [Vcc / SysC	5.0V / 24MHz
CPU_Clock	SysClk/8
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	1
VC2= VC1/N	1
VC3 Source	SysClk/1
VC3 Divider	1
SysClk Source	Internal
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-BandGap
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	Low
SwitchModePump	OFF
Trip Voltage [LVD (SMP)]	4.81V (5.00V)
LVDThrottleBack	Disable
Watchdog Enable	Disable

8 Bit Capture or FreeRun Prescaler

Selects which eight bits of the 16-bit free running timer, that are captured when in the 8-bit mode.

Registers Affected:
TMRCR

32K_Select

The 32K_Select parameter allows selection of the internal 32 kHz oscillator or an external crystal oscillator. A complete discussion of the implications of this selection is found in the *PSoC Technical Reference Manual*.

A_Buf_Power

A_Buf_Power allows the user to select the power level for the analog output buffers of the PSoC. These buffers are used to supply internal analog signals to external pins on the PSoC. Power levels may affect the frequency response and current drive capability of the output buffers. Complete tables for the AC Analog Output Buffer Specifications and DC Analog Output Buffer Specifications are contained in the applicable device data sheet.

AGNDBypass

A provision is made in some versions of the PSoC device to provide an external AGND bypass capacitor to reduce the small amounts of switching noise present on the internal AGND. This feature is switched on and off using the AGNDBypass global parameter. Typical values for the external bypass capacitor are 0.01 μF and should not generally exceed 10 μF. The recommended value is 1 μF.

Analog Power

This parameter controls the power to the analog section of the PSoC. Power is controlled in three stages:

1. All Analog Blocks Off
2. Continuous Time Blocks ON/Switched Capacitor Blocks OFF
3. Continuous Time Blocks ON/Switched Capacitor Blocks ON

For each of the two 'ON' cases, select reference drive levels of high, medium, and low to choose the current drive capability for the internal reference buffers. All selections of this parameter, whether used as a User Module Parameter or this Global Resource, need to agree. This selection affects the total power consumption of the PSoC. Each user module using the reference and the opamp block associated with it adds slightly to the power consumed by the device. Since the internal reference is used as an integral part of most switched capacitor circuits, the current drive capability has an impact on the speed at which the switched capacitor block operates. In general, higher settings for this parameter allow switched capacitor circuits to operate at higher clock rates, at the expense of higher power consumption. To estimate the current (and power) consumption per opamp block, refer to the applicable table in the data sheet for the part: DC Operation Amplifier Specifications (ISOA).

Crystal OSC

Selects the external crystal oscillator when enabled. The external crystal oscillator shares pads CLKIN and CLKOUT with two GPIOs; P0.0 and P0.1, respectively.

Crystal OSC Xgm

XGM is the amplifier transconductance setting and selects the calibration for the external crystal oscillator.

EFTB

The external crystal oscillator is passed through the EFTB filter when this option is enabled.

FreeRun Timer and Free Run Timer/N

Selects clock source for 16-bit free-running timer. The free-running timer generates an interrupt at a 1024-µs rate. It can also generate an interrupt when counter overflow occurs at every 16.384 ms. The combination of the FreeRun Timer and the FreeRun Timer divider are used to obtain the FreeRun Timer rate.

LVD ThrottleBack

This parameter allows you to configure the PSoC to lower its own CPU clock speed under low voltage conditions. Use of this parameter and the bit it controls is discussed in the PSoC Technical Reference Manual. Not all PSoC devices incorporate this parameter and bit.

Opamp Bias

Performance of the internal opamps are tailored based upon the application under development by selecting high or low bias conditions for the analog section of the PSoC. Selecting high bias causes the opamp to consume more current but also increases its bandwidth and switching speed, lowering its output impedance. To estimate the current (and power) consumption per opamp block, including the effect from high or low selection of opamp Bias, refer to the applicable table in the data sheet for the part: DC Operation Amplifier Specifications (ISOA). To estimate the effect on AC opamp parameters, refer to the applicable AC Operational Amplifier Specifications in the device data sheet.

PLL_Mode

The *PSoC Technical Reference Manual* discusses use of the phase-locked loop mode.

Power Setting [Vcc/SysClk Freq]

This parameter allows you to select the SysClock frequency and nominal operating voltage. Based upon the SysClock selected, the Internal Main Oscillator (IMO) is set with appropriate calibration settings. Since many internal clocks are derived from the SysClock, you see significant device power-consumption savings by lowering the SysClock frequency, if the implemented design permits it.

Ref Mux

The Ref Mux source selection is used to control the range and (potential) accuracy of various analog resources. The reference chosen controls the maximum voltage that is input to a switched capacitor circuit and output from a switched capacitor circuit. Both the analog ground level and the peak-to-peak voltage are selected using this parameter. Values specified with the Ref Mux parameter are in pairs and consist of [AGND level ± full scale]. Keep in mind that selecting Vdd (supply voltage) as a reference level couples Vdd changes into the AGND input of internal resources. This directly affects absolute accuracy of measurements. Using the internal bandgap reference results in better accuracy but may not offer an exact Vdd/2 input reference. Choices of ± full-scale values also offer a number of options. These full-scale values may be based on the PSoC internal references or on external input voltages. The ± full scale values present constraints similar to those for AGND in terms of Vdd variation and absolute range of input/output. Individual design criteria dictate the best selection for the AGND and ± full-scale values. Further discussion of the analog reference can be found in the *PSoC Technical Reference Manual*.

Supply Voltage

Selects the nominal operating voltage to be either 3.3V or 5V.

SwitchModePump

An integrated switch mode pump circuit is available for operation of the device from very low voltage sources. The pump requires a few external components and can be configured to automatically turn on as supply voltage drops. Further discussion of the switch mode pump is found in the *PSoC Technical Reference Manual*.

SysClk_Source and SysClk*2 Disable

These parameters allow you to select the 24 MHz system SysClock from an internal or external source. The SysClock*2 Disable parameter allows the internal 48 MHz clock to be shut off. A complete discussion of system clocking is found in the *PSoC Technical Reference Manual*.

Trip Voltage [LVD (SMP)]

A precision POR circuit is integrated into the PSoC. This parameter allows the user to select voltage levels that the PSoC uses to internally monitor its supply voltage. Two levels are specified in the parameter with the syntax <LVD (SMP)>. LVD is the value at which the internal low voltage comparator asserts its control signal. SMP is the level at which the integrated switch mode pump is enabled. Although selection of SMP is implicit in the selection of LVD, if no switch mode pump circuitry is used, the part is reset if supply voltage falls too low. At the point when the supply voltage exceeds the threshold level, the part resumes operation as if the power was switched off and on (POR). Further discussion of the switch mode pump and low voltage detect is found in the *PSoC Technical Reference Manual*.

VC1 and VC2

These resources are clocks that can be chained to provide various internal clock frequencies used for digital or analog blocks. A complete discussion of system clocking is found in the *PSoC Technical Reference Manual*.

VC3_Source and VC3_Divider

VC3 is a system clock resource similar to the VC1 and VC2 resources. The main difference between it and VC1 and VC2 is that VC3 may be chained from one of several clock sources and may not be used as an input clock as flexibly as VC1 and VC2. You cannot use it as a direct input to the analog section of the PSoC. It can be used as an input to a digital PSoC block and then used to derive a clock that can be used in many more places. For this reason, it is important to evaluate clocking options as a PSoC design is being developed. Often, rearranging clock sources according to where they are most easily connected solves clocking problems. A complete discussion of system clocking is found in the *PSoC Technical Reference Manual*.

Global Resources for USB Chips

Capture Clock

Selects clock source for the Timer Capture Clock (TCAPCLK).
Registers Affected:
TMRCLKCR

Capture Clock /N

Selects whether the capture timer data registers holds the data from the first detected edge or the most recent detected edge.
Registers Affected:
TMRCLKCR

Capture Edge

Selects whether the capture timer data registers holds the data from the first detected edge or the most recent detected edge.
Registers Affected:
TMRCR

CLKOUT Source

Selects one of the clocks, internal SysClk, external, low power 32 kHz, or CPUCLK to be output directly on port P0[1].
Registers Affected:
CLKIOCR

CPU_Clock

Selects the source of SysClk as the Internal Main Oscillator (IMO) or the external source from Port 1[4]. External clock source input in enCoRe II is P0.0 (not P1.4 like in PSoC devices)
Registers Affected:
CPUCLKCR

CPU Clock/N

Selects the CPU clock speed, from 187 kHz to 24 MHz. Derived from the SysClk. This setting affects the Power on Reset level in order to prevent the CPU from running outside of its Vdd specification.

Registers Affected:
OSC_CR0

Low V Detect

Selects the level of the supply voltage at which the Low voltage detect interrupt is generated.
Registers Affected:
LVDCR

Sleep_Timer

Selects the timing of the sleep interrupt between 1 Hz and 512 Hz. The watchdog timer is affected for every 3 sleep timer cycles when it is enabled.
Registers Affected:
OSC_CR0

Timer Clock

Selects clock source for the 12-bit down counting internal timer (TMRCLK)
Registers Affected:
TMRCLKCR

Timer Clock/N

Selects the value by which to divide the source of TMRCLK to obtain TMRCLK.
Registers Affected:
TMRCLKCR

USB_Clock

Selects the clock source for USB SIE. Internal 24 MHz IMO or external clock source at P0.0
Registers Affected:
CPUCLKCR

USB Clock/2

Provides an option to divide the USB clock source by 2, depending on clock source and frequency. When USB clock is the internal 24 MHz oscillator, then the divide by 2 option is always enabled.
Registers Affected:
CPUCLKCR

V Reset

Selects the Power on Reset (POR) voltage level.
Registers Affected:
LVDCR

V Reg

A 3.3V (125 mA) regulator output is places on P1[2] when enabled. This must ONLY be enabled when the supply voltage is above 4.35V. A 1uF min, 2uF max capacitor is required on Vreg output.
Registers Affected:
VREGCR

V Keep-alive

Allows voltage regulator to source upto 20 ìA of current when voltage regulator is disabled.
Registers Affected:
VREGCR

Watchdog Enable

Enables watchdog timer that will result in a CPU reset if not serviced. Timing is based on a counter that counts 3 sleep timer events.
Registers Affected:
CPU_SCR

8 Bit Capture Prescaler

Selects which eight bits of the 16-bit free running timer, that are captured when in the 8-bit mode.
Registers Affected:
TMRCR

2.6 Project Backup Folder

PSoC Designer maintains a backup folder in the project directory for files that were removed from the source tree. This includes files that are manually removed and files removed due to cloning or code generation. The backup folder only retains the version of the file that was last removed. The files are named identically to the original project file and the `\lib` directory is not retained (i.e., library files are placed directly under the backup folder).

2.7 Specifying Interconnects

Interconnectivity allows communication between user modules, PSoC blocks, pins, and other on-chip resources.

Connections are shown as lines between elements, special symbols, or flag connectors. Flag connectors are used when the connection is made to a point where drawing a line results in a cluttered display, with the legend indicating the origin of the connection. Connections to pins are shown as lines from interconnection buses. The interconnection bus structure depends on the PSoC device selected and can consist of one or more levels of buses between the digital PSoC blocks and the pins.

Connections between analog PSoC blocks and pins are made through the analog input muxes and output buses.

Pin names are duplicated in several places, since they are multifunctional, and are highlighted when used with lines showing their current connection state. The location of the pin to which a line is drawn indicates the usage of the pin. Lines drawn to the pins on the left edge indicate that the pins are used as inputs, while the right edge indicates the pins are used as output.

Pins in the upper groups indicate connection to the digital network, while lower groups indicate analog connections. Lines drawn from multiple locations on the same pin indicate that the shown combination is electrically valid.

To specify interconnections, double-click the **Interconnect** folder in the Workspace Explorer.

User module interconnections consist of connections to surrounding PSoC blocks, output bus, input bus, internal system clocks and references, external pins, and analog output buffers. Multiplexers may also be configured to route signals throughout the PSoC block architecture.

Digital PSoC blocks are connected through the GlobalIN and GlobalOUT buses to external pins and to other digital PSoC blocks. There are 8 GlobalInOdd/GlobalInEven, and 8 GlobalOutOdd/ GlobalOutEven line buses, numbered 0 through 7. For external pin connections, the number of the Global bus line corresponds to the bit number of the associated port. For example, GlobalInOdd_0 / GlobalInEven_0 can connect to pins associated with P0[0], P1[0], P2[0], etc. The GlobalOUT buses can drive the inputs to other digital PSoC blocks. However, all GlobalOUT lines do not reach all digital PSoC blocks. Refer to the *PSoC Technical Reference Manual* for details on the global bus interconnections.

When setting output parameters to the GlobalOUT lines, only one PSoC block drives a single GlobalOUT line at a time. GlobalOUT lines used by a user module are not available to other user modules for output. For example, if two timer user modules are placed and the first timer is set to use GlobalOUTOdd_1 / GlobalOUTEven_1 for output, attempting to set the output for the second timer to GlobalOUTOdd_1 / GlobalOUTEven_1 fails.

2.7.1 Connecting User Modules

These procedures show you how to make certain types of connections.

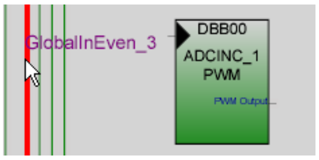
Global In

Global In connections apply to a PSoC device in this manner:

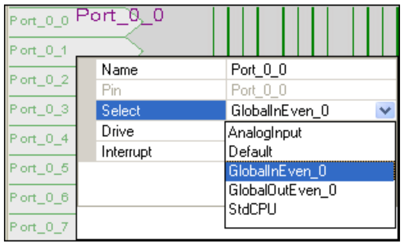
- CY8C25xxx/26xxx as Global In: Input Port Connections.
- All other PSoC devices as Global In Odd and Global In Even: Input Port Connections and Global Connections.

To set Global In connections:

1. Click on the target Globalxxx vertical line.

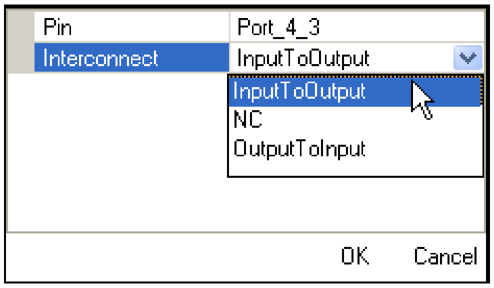


2. Select the pin to connect to.



Name	Port_0_0
Pin	Port_0_0
Select	GlobalInEven_0
Drive	
Interrupt	

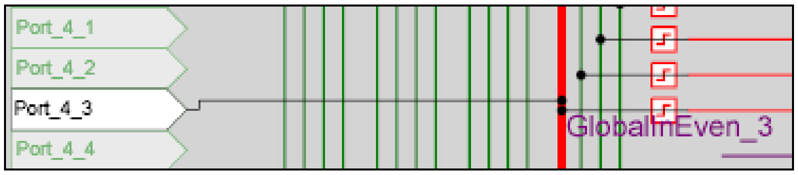
3. Select the global input to output connection (if active).



Pin	Port_4_3
Interconnect	InputToOutput

4. Click **OK.**

You see a line connecting the digital input port to the global vertical line.



Global Out

Global Out connections apply to a PSoC device in this manner:

- CY8C25xxx/26xxx as Global Out: Output Port Connections.
- All other PSoC devices as Global Out Odd and Global Out Even: Output Port Connections and Global Connections.

To set Global Out connections:

1. Click on the target Globalxxx vertical line.
2. Select the global input to output connection (if active) and the port.
3. Click **OK**.

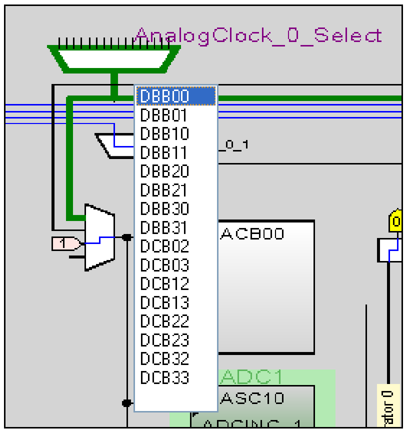
   You see a line connecting the digital output port to the global vertical line.

Analog Clock Select

To set Analog Clock Select connections:

1. Click on the target AnalogClock_x_Select Mux.

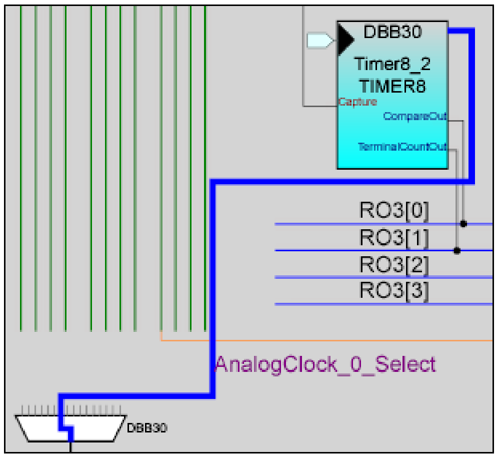
Figure 2-7. The AnalogClock_0_Select Mux



2. Select a DBAxx or DBBxx PSoC block (as applies).

You see a line from the right side of DBxxx to the input of the AnalogClock_x_Select Mux. The mux switch shows a connection to this input.

Figure 2-8. The AnalogClock_0_Select Set to Connect to DBB30

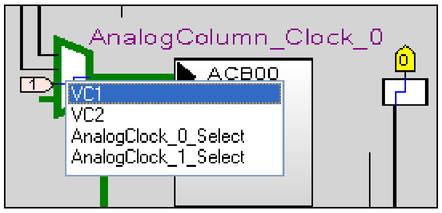


Analog Column Clock

To set Analog Column Clock connections:

1. Click on the target AnalogColumn_Clock_x Mux.

Figure 2-9. Setting the AnalogColumn_Clock_0 Mux



2. Select a device-specific option from the menu.

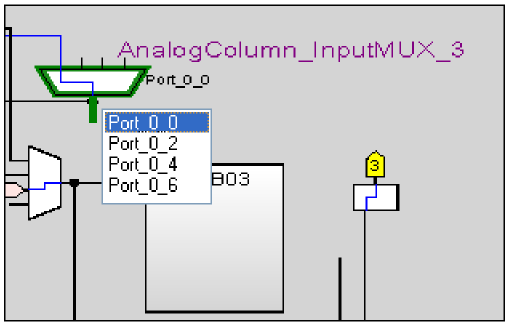
   You see that the AnalogColumn_Clock_x Mux has a line connecting your chosen option to the mux output.

Analog Column Input Mux

To set Analog Column Input Mux connections:

1. Click on the target AnalogColumn_InputMUX_x.

Figure 2-10. Setting the AnalogColumn_InputMUX_3



2. Select a port from the menu.

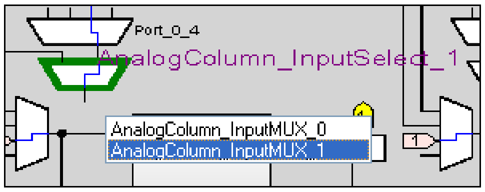
You see a connection between the output of AnalogColumn_InputMUX_x and the analog input port.

Analog Column Input Select

To set Analog Column Input Select connections:

1. Click on the target AnalogColumn_InputSelect_x.

Figure 2-11. Setting the AnalogColumn_InputSelect_1



2. Select appropriate AnalogColumn_InputMUX_x from the menu.

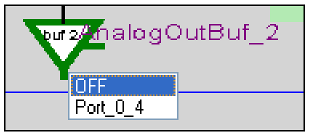
   You see that your chosen AnalogColumn_InputSelect_x has a line inside that connects the output of AnalogColumn_InputMUX_x to its output.

Analog Output Buffer

The Analog Output Buffers can be connected to the associated port pin or turned off. To set Analog Output Buffer connections:

1. Click on the target AnalogOutBuf_x.

Figure 2-12. Setting the AnalogOutBuf_2



2. Select a port from the menu.

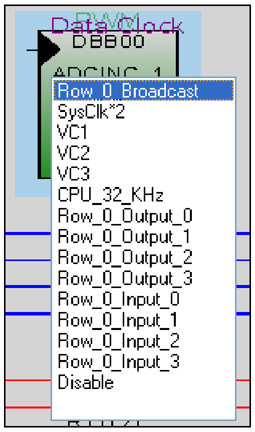
   You see a line that connects the AnalogOutBuf_x triangle to the analog output port.

Clock Input for a Digital Block

To set Clock Input connections on a digital block:

1. Click the clock input triangle on the digital block where your target user module is placed. Note that the clock input triangle is not active for all blocks when a user module uses more than one block. Also note that the name “Clock Input” is determined by a specified user module parameter.

Figure 2-13. Setting the Clock Input for an ADCINC User Module



2. Select an option from the menu. You see your chosen input option displayed next to the clock input triangle.

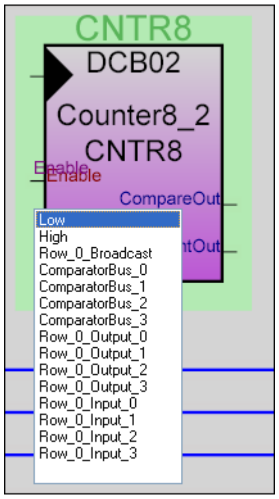
   Your choice option also appears in the Control Clock field under User Module Parameters (where you can click the drop-arrow to change your selection).

Enable Input for a Digital Block

To set the Enable Input connection on a digital block:

1. Click the Enable text label on the digital block where your target user module is placed. Note that the name Enable Input is determined by a specified user module parameter.

Figure 2-14. Setting the Enable Input for an 8-bit Counter User Module



2. Select an option from the menu.

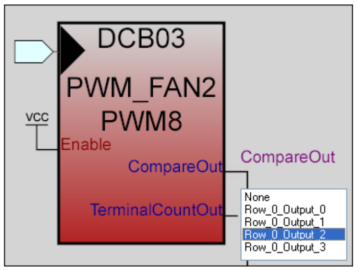
   You see your chosen input option displayed next to the Enable text label. Your choice option also appears in the Enable field under User Module Parameters (where you can click the drop-arrow to change your selection).

Output for a Digital Block

To set Output connections on a digital block:

1. Click the Output text label on the digital block where your target user module was placed. Note that the name Output is determined by a specified user module parameter.

Figure 2-15. Setting the CompareOut on an 8-bit Pulse Width Modulator User Module



2. Select an option from the menu (None, Global_OUT_x for CY8C25xxx/26xxx, or Row_x_Output_x for all other PSoC devices).

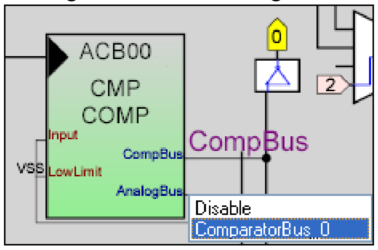
   You see your chosen option displayed (with a connection) next to the Output text label. Your choice option also appears in the Output field under User Module Parameters (where you can click the drop-arrow to change your selection).

RBotMux for a CT Analog Block

To select a RBotMux for a CT Analog Block, follow this procedure. You can use this procedure when the NMux, PMux, AnalogBus, or CompBus CT Analog Block apply, as well as for ACMux, BMux, AnalogBus, or CompBus SC Analog Blocks.

1. Click the RBotMux text label on the analog block where your target user module was placed. Note that the name RBotMux is determined by a specified user module parameter.

Figure 2-16. Setting the Comparator Bus on a Comparator User Module



2. Select an option from the menu.

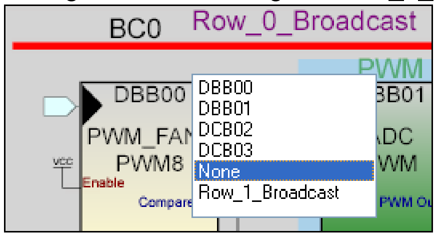
   You see your chosen option displayed next to the RBotMux text label. Your choice option also appears in the RBotMux field under User Module Parameters (where you can click the drop-arrow to change your selection).

Row Broadcast

Row Broadcast connections do not apply to CY8C25xxx/26xxx parts. To set Row Broadcast connections:

1. Click the Row_0_Broadcast (BC0) or Row_1_Broadcast (BC1) horizontal line.

   Figure 2-17. Setting the Row_0_Broadcast Line



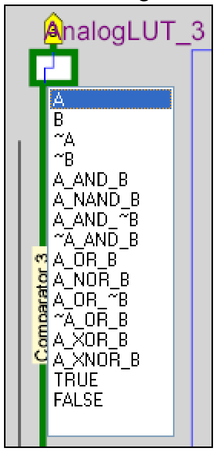
2. Select an option from the menu.

   You see a line connecting to a digital PSoC block or to the other Row Broadcast, depending on the option you chose.

Comparator Analog LUT

Comparator Analog LUT connections do not apply to CY8C25xxx/26xxx parts. To set Comparator Analog LUT connections:

1. Click the AnalogLUT_x box. (Its symbol is identified in the Comparator x line along each column of analog PSoC blocks.)



2. Select an option from the menu.

   You see connections on the device interface reflecting your A or B selection with associated symbols.

2.7.2 Digital Interconnect Row Input Window

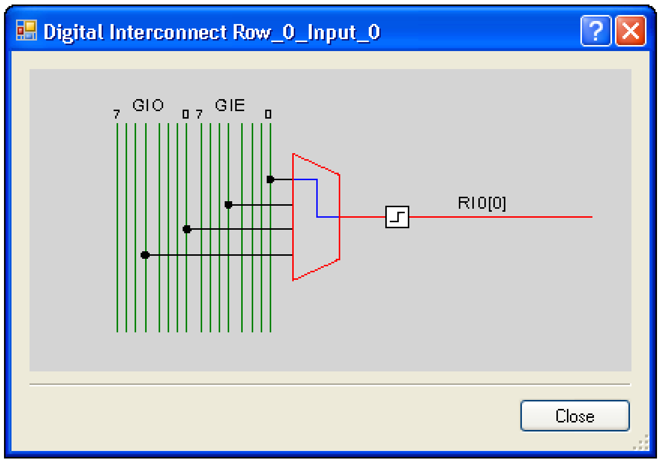
The Digital Interconnect Row Input window connections do not apply to CY8C25xxx/26xxx parts.

Connection to Global Input

To set a Connection to Global Input:

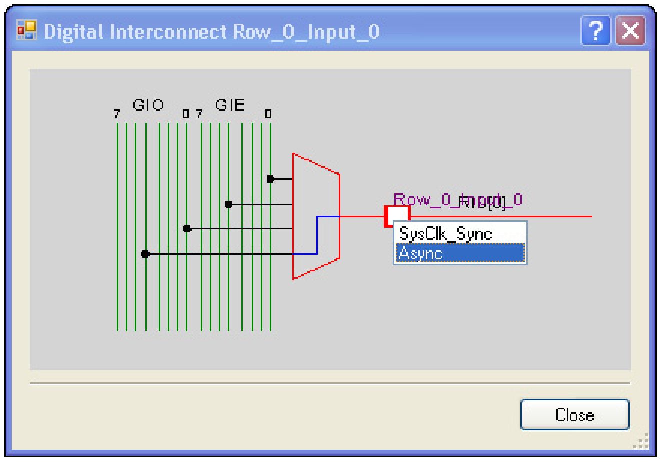
1. Click on the white box or the line of the target Row_x_Input_x. (A tool tip will appear to identify your selection.)

Figure 2-18. Digital Interconnect Row Input



2. Click on the Row_x_Input_x Mux in the Digital Interconnect Row Input floating window and select a Global Input from the menu. (You immediately see a connection from the mux to the Global Input vertical line.) In this floating window you can also click the white box to toggle the Synchronization value for Row_x_Input_x. Options include SysClk_Sync and Async

Figure 2-19. Synchronization Options for Digital Interconnect Row Inputs



3. Click **Close** when finished.

2.7.3 Digital Interconnect Row Output Window

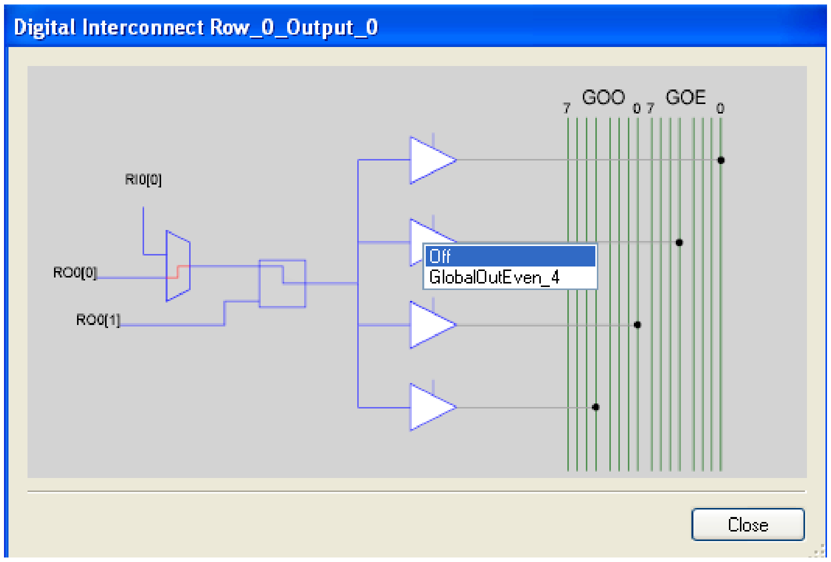
Digital Interconnect Row Output Window connections do not apply to CY8C25xxx/26xxx parts.

Row Logic Table Input

To set Row Logic Table Input connections:

1. Click on the target Row_x_Output_x Logic Table Box.

Figure 2-20. Digital Interconnect Row Output



2. Click on the Row_x_LogicTable_Input_x Mux in the Digital Interconnect Row Output floating window and select an input or output option from the menu.
3. Click **Close** when finished.

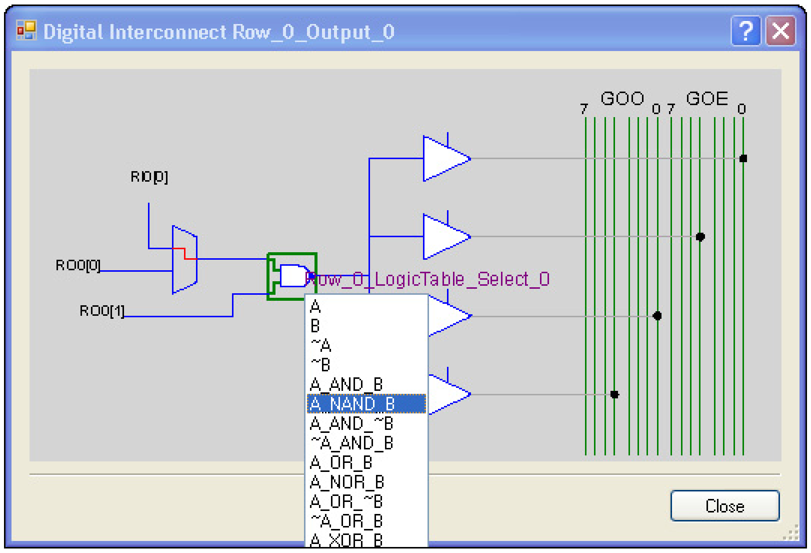
   You see connections on the device interface reflecting your row input or output selection.

Row Logic Table Select

To set Row Logic Table Select connections:

1. Click on the target Row_x_Output_x Logic Table Box.
2. Click on the Row_x_LogicTable_Select_x logical operation box in the Digital Interconnect Row Output floating window and select an option from the menu

Figure 2-21. Logical Operations in Digital Interconnect Row Output



3. Click **Close** when finished.

   You see connections on the device interface reflecting your A or B input selection with associated symbol.

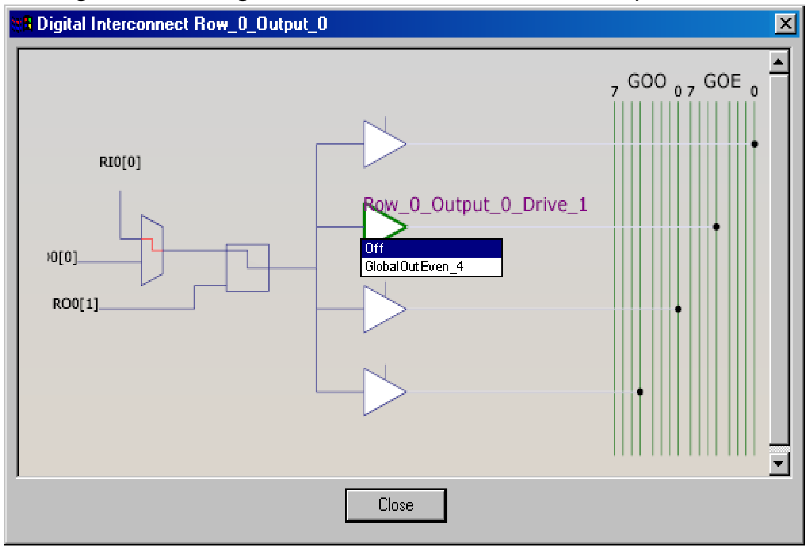
Connection to Global Output

To set connections to Global Output:

1. Click on the target Row_x_Output_x Logic Table Box.
2. Click on the target Row_x_Output_x_Drive_x triangle in the Digital Interconnect Row Output floating window and select an option from the menu.

   After you open the Digital Interconnect Row Global Output window, you can select Row Logic Table Input, Row Logic Table Select, and Connections to Global Output without closing the window.

Figure 2-22. Digital Interconnect Row Global Output



3. Click the **Close** button when finished.

   You see a connection from the Row_x_Output_x Logic Table Box to the chosen GlobalOutEven_x vertical line.

2.8 Specifying the Pinout

Specifying the pinouts is the next step to configuring your target device. This converts the pins to the configurable PSoC resources.

To restore the default pinout, click the Restore Default Pinout button .

Be careful when connecting to pins. The pin settings can be modified either by setting elements to connect to pins or by setting the pin directly.

Setting the pin directly connects the pin to the appropriate element and disconnects it from any other element. To have multiple connections to the same pin, make connections from the element to the pin. For example, suppose a connection to a pin, an analog input mux and an analog output buffer, simultaneously, is desired. P0[2] can connect to the analog input mux for column 1 and to the analog output buffer for column 3. The connections must be made from the analog input mux and the analog output buffer. Setting the pin to Default disconnects the pin from both digital buses, but does not affect analog connections.

2.8.1 Port Connections

You make port connections in three ways:

- Click the port icons and make settings in the device interface
- Click the pin and make settings in the device pinout
- Change port-related fields in the Pinout window

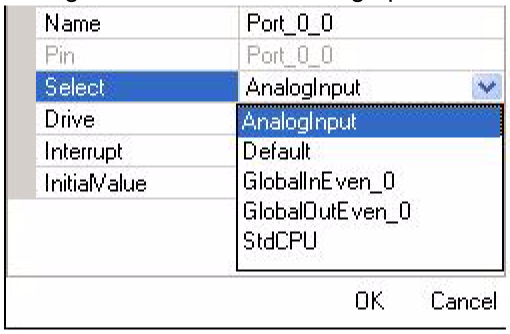
These procedures show you how to make certain types of port connections.

Analog Input

To set Analog Input connections.

1. Click on the target Port_0_x.
2. From the **Select** menu select **AnalogInput**.

Figure 2-23. Select AnalogInput for a Port



Name	Port_0_0
Pin	Port_0_0
Select	AnalogInput
Drive	
Interrupt	
InitialValue	

AnalogInput
Default
GlobalInEven_0
GlobalOutEven_0
StdCPU

3. Click **OK**.

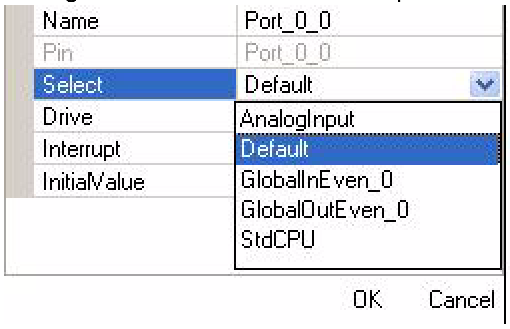
   On the device you see the new designation color coded according to the legend along side the device. The port name and selection also appears in the port-related fields underneath Pinout window (where you can click the drop-arrow to change your selection).

Default Input

To set Default Input connections:

1. Click on the target Port_x_x.
2. From the **Select** menu select **Default**.

Figure 2-24. Select Default Input for a Port



Name	Port_0_0
Pin	Port_0_0
Select	Default
Drive	
Interrupt	
InitialValue	

AnalogInput
Default
GlobalInEven_0
GlobalOutEven_0
StdCPU

3. Click **OK**.

   On the device pinout frame you see the designation color coded according to the legend along side the device. The port name, the Select column value of StdCPU, and the drive mode of High

Z Analog also appears in the port-related fields underneath Pinout window (where you can click the drop-arrows to change your selections).

Global_IN_x

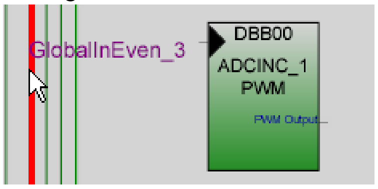
Global_IN_x connections apply to a PSoC device in this manner:

- CY8C25xxx/26xxx as Global_IN_x.
- All other PSoC devices as GlobalIn[Odd/Even]_x.

To set Global_IN_x connections:

1. Click on the target Port_x_x.
2. From the **Select** menu select the device-specific **Global IN** option.

Figure 2-25. Select Global IN for a Port



3. Click **OK**.

   On the device you see the designation color coded according to the legend next to the device. The port name, the Select column value of your chosen option, and the drive mode of High Z appear in the port-related fields underneath Pinout window (where you can click the drop-arrows to change your selections).

   You also see a line between the digital input port and the Global IN vertical line.

Global_OUT_x

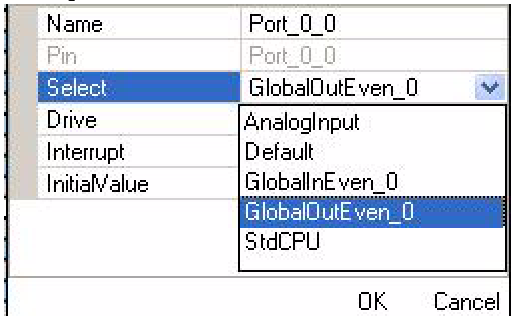
Global_OUT_x connections apply to a PSoC device in this manner:

- CY8C25xxx/26xxx as Global_OUT_x.
- All other PSoC devices as GlobalOUT[Odd/Even]_x.

To set Global_OUT_x connections:

1. Click on the target Port_x_x.
2. From the **Select** menu select the device-specific **Global OUT** option.

Figure 2-26. Select Global OUT for a Port



Name	Port_0_0
Pin	Port_0_0
Select	GlobalOutEven_0
Drive	
Interrupt	
InitialValue	

3. Click **OK**.

   On the device you see the designation color coded according to the legend next to the device. The port name, the Select column value of your chosen option, and the drive mode of Strong appear in the port-related fields underneath Pinout window (where you can click the drop-arrows to change your selections).

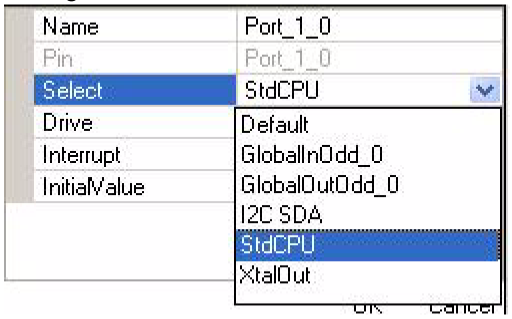
You also see a line between the Global OUT vertical line and the digital output port.

StdCPU

To set StdCPU connections:

1. Click on the target Port_x_x or select the port from the menu.
2. From the **Select** menu select **StdCPU**.

Figure 2-27. Select StdCPU for a Port



Name	Port_1_0
Pin	Port_1_0
Select	StdCPU
Drive	
Interrupt	
InitialValue	

3. Click **OK**.

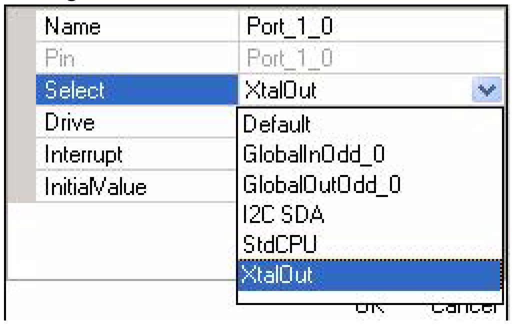
   On the device you see the designation color coded according to the legend next to the device. The port name and StdCPU also appear in the port-related fields underneath Pinout window (where you can click the drop-arrow to change your selection).

XtalOut

To set the XtalOut connection:

1. Click on Port_1_0 (P1[1]) or select Port_1_0 from the menu.
2. From the **Select** menu select **XtalOut**.

Figure 2-28. Select XtalOut for Port 1 0



Name	Port_1_0
Pin	Port_1_0
Select	XtalOut
Drive	
Interrupt	
InitialValue	

3. Click **OK**.

   On the device you see the designation color coded according to the legend next to the device. The port name, XtalOut, and the drive mode of High Z also appear in the port-related fields underneath Pinout window (where you can click the drop-arrow to change your selection).

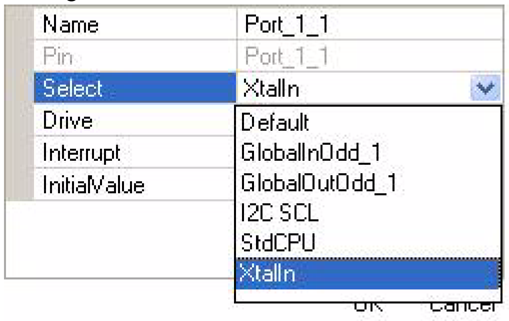
XtalIn

To set the XtalIn connection:

1. Click on Port_1_1 (P1[1]) or select Port_1_1 from the menu.

2. From the **Select** menu select **XtalIn**.

Figure 2-29. Select CXtalOut for Port 1 1



Name	Port_1_1
Pin	Port_1_1
Select	XtalIn
Drive	
Interrupt	
InitialValue	

Default
GlobalInOdd_1
GlobalOutOdd_1
I2C SCL
StdCPU
XtalIn

3. Click **OK**.

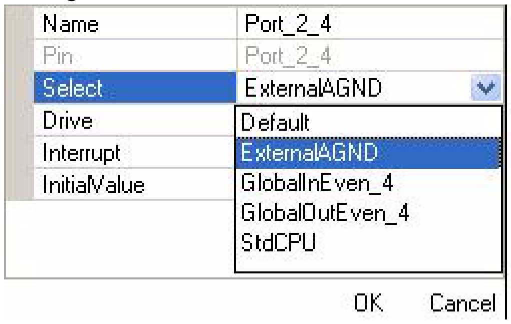
   On the device you see the designation color coded according to the legend next to the device. The port name, XtalIn, and the drive mode of High Z also appear in the port-related fields underneath Pinout window (where you can click the drop-arrow to change your selection).

ExternalGND

To set the ExternalGND connection:

1. Click on Port_2_4 (P2[4]) or select Port_2_4 from the menu.
2. From the **Select** menu select **ExternalAGND**.

Figure 2-30. Set External Ground for Port 2 4



Name	Port_2_4
Pin	Port_2_4
Select	ExternalAGND
Drive	
Interrupt	
InitialValue	

Default
ExternalAGND
GlobalInEven_4
GlobalOutEven_4
StdCPU

3. Click **OK**.

   On the device you see the designation color coded according to the legend next to the device. The port name and ExternalGND appear in the port-related fields underneath Pinout window (where you can click the drop-arrow to change your selection).

   In the device interface you see that all lines from P2[4] are gone.

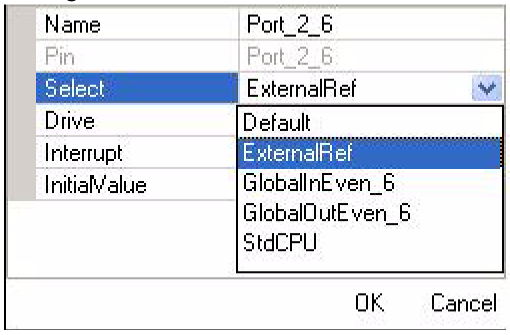
Ext Ref

To set the Ext Ref connection:

1. Click on Port_2_6 (P2[6]) or select P2[6] from the menu.

2. From the **Select** menu select **ExtRef**.

   Figure 2-31. Set External Reference for Port 2 6



Name	Port_2_6
Pin	Port_2_6
Select	ExternalRef
Drive	
Interrupt	
InitialValue	

3. Click **OK**.

   On the device you see the designation color coded according to the legend next to the device. The port name and Ext Ref also appear in the port-related fields underneath Pinout window (where you can click the drop-arrow to change your selection).

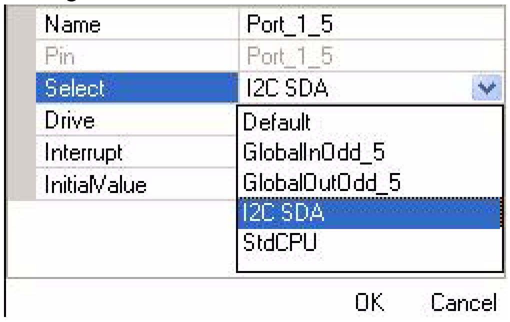
   In the device interface you see that all lines from P2[6] are gone.

I2C SDA

To set the I2C SDA connection

1. Click on Port_1_5 (P1[5]) or select P1[5] from the menu.
2. From the **Select** menu select **I2C SDA**.

   Figure 2-32.



Name	Port_1_5
Pin	Port_1_5
Select	I2C SDA
Drive	
Interrupt	
InitialValue	

3. Click **OK**.

   On the device you see the designation color coded according to the legend next to the device. The port name, I2C SDA, and the drive mode of Open Drain High also appear in the port-related fields underneath Pinout window (where you can click the drop-arrow to change your selection).

   In the device interface you see that all lines from Port_1_5 are gone.

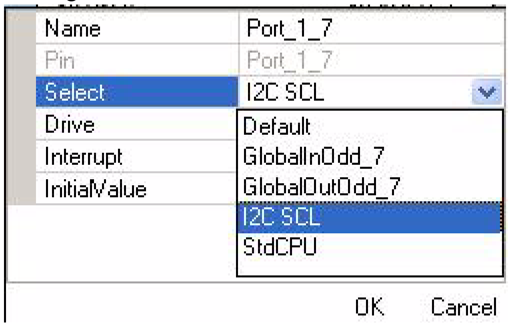
I2C SCL

To set the I2C SCL connection

1. Click on Port_1_7 (P1[7]) or select P1[7] from the menu.

2. From the **Select** menu select **I2C SCL**.

   Figure 2-33.



Name	Port_1_7
Pin	Port_1_7
Select	I2C SCL
Drive	
Interrupt	
InitialValue	

3. Click **OK**.

   On the device you see the designation color coded according to the legend next to the device. The port name, I2C SCL, and the drive mode of Open Drain High also appear in the port-related fields underneath Pinout window (where you can click the drop-arrow to change your selection).

   In the device interface you see that all lines from Port_1_7 are gone.

2.8.2 Port Drive Modes

Port drive modes can be specified in one location, in three ways:

- Click the port icons and make settings in the device interface
- Click the pin and make settings in the device pinout
- Change port-related fields in the Pinout window.

Depending on the architecture, PSoC Devices have either four or eight drive modes. All PSoC devices have High Z, Pull Up, Pull Down, and Strong. Most PSoC devices also have High Z Analog, Open Drain High, Open Drain Low, and Strong Slow. To specify a port drive mode:

1. Click on the target Port_x_x.
2. From the **Drive** menu select the port drive mode option.

   The port name and the drive mode of your choice appears in the port-related fields in the Pinout window (where you can click the drop-arrows to change your selections).

2.8.3 Port Interrupts

These procedures show you how to work with certain types of port interrupts.

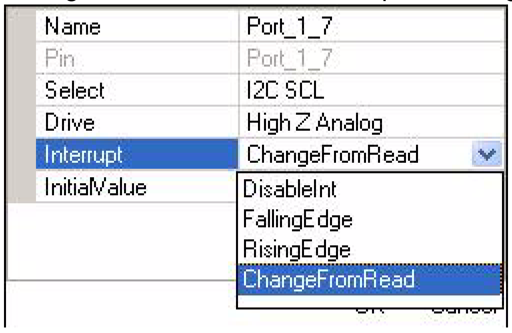
ChangeFromRead

To specify a ChangeFromRead interrupt:

1. Click on the target Port_x_x.

2. From the **Interrupt** menu select **ChangeFromRead**.

Figure 2-34. Set Port Interrupt to Change From Read



Name	Port_1_7
Pin	Port_1_7
Select	I2C SCL
Drive	High Z Analog
Interrupt	ChangeFromRead
InitialValue	

3. Click **OK**.

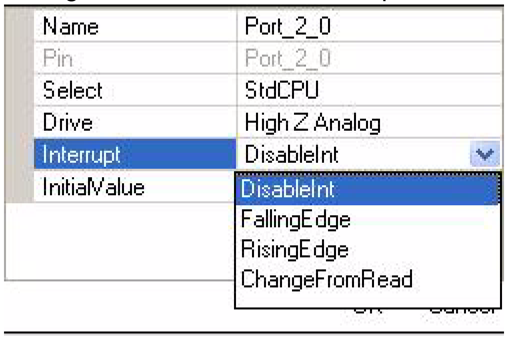
   The port name and ChangeFromRead appears in the port-related fields in the Pinout window (where you can click the drop-arrows to change your selections).

DisableInt

To disable interrupts:

1. Click on the target Port_x_x.
2. From the **Interrupt** menu select **DisableInt**.

Figure 2-35. Set Port Interrupt to Disable



Name	Port_2_0
Pin	Port_2_0
Select	StdCPU
Drive	High Z Analog
Interrupt	DisableInt
InitialValue	

3. Click **OK**.

   The port name and DisableInt appears in the port-related fields in the Pinout window (where you can click the drop-arrows to change your selections).

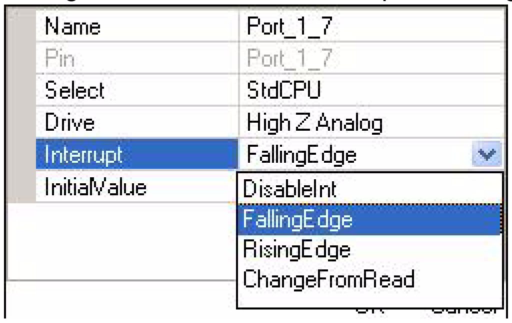
FallingEdge

To specify FallingEdge interrupt:

1. Click on the target Port_x_x.

2. From the **Interrupt** menu select **FallingEdge**.

Figure 2-36. Set Port Interrupt to Falling Edge



Name	Port_1_7
Pin	Port_1_7
Select	StdCPU
Drive	High Z Analog
Interrupt	FallingEdge
InitialValue	

3. Click **OK**.

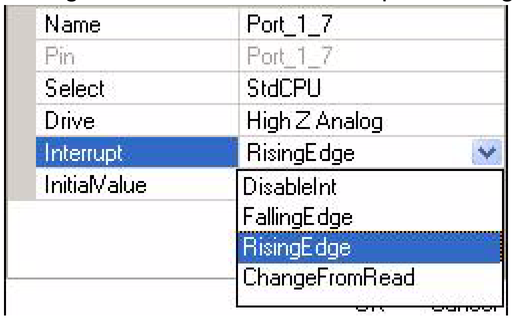
   The port name and **FallingEdge** appears in the port-related fields in the Pinout window (where you can click the drop-arrows to change your selections).

RisingEdge

To specify RisingEdge interrupt:

1. Click on the target Port_x_x.
2. From the **Interrupt** menu select **RisingEdge**.

Figure 2-37. Set Port Interrupt to Rising Edge



Name	Port_1_7
Pin	Port_1_7
Select	StdCPU
Drive	High Z Analog
Interrupt	RisingEdge
InitialValue	

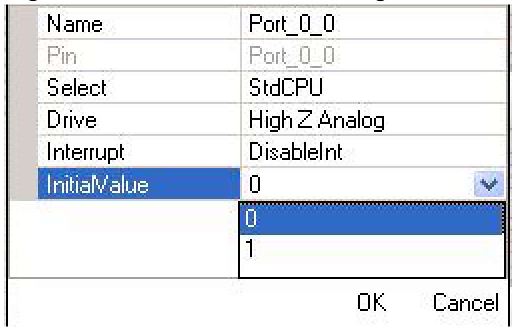
3. Click **OK**.

   The port name and RisingEdge appears in the port-related fields in the Pinout window (where you can click the drop-arrows to change your selections).

2.8.4 InitialValue

A new InitialValue feature has been added to all pins in PSoC 1 devices. This allows you to control the initial value of each pin. The InitialValue is automatically set based on the drive mode of the pin, but you can alter this value if needed.

Figure 2-38. Initial Value Setting



Name	Port_0_0
Pin	Port_0_0
Select	StdCPU
Drive	High Z Analog
Interrupt	DisableInt
InitialValue	0

Table 2-1. Drive Modes and InitialValue

Drive Mode	Default InitialValue Setting
Strong	0
Strong Slow	0
High Z	0
High Z Analog	0
Open Drain Low	1
Open Drain High	0
Pull Up	1
Pull Down	0

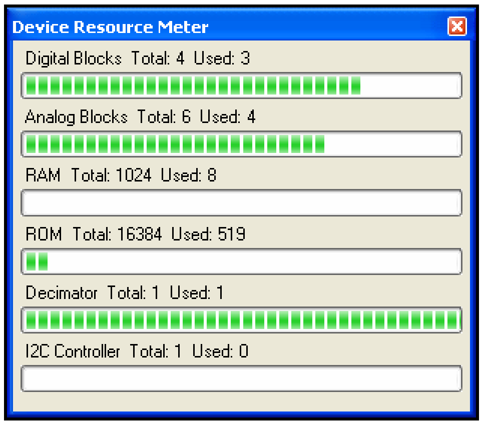
2.9 Tracking Device Space

Tracking the available space and memory of configurations for your device is something you do intermittently throughout the process of configuring your target device. You need to monitor device space and memory resources so you are aware, on an ongoing basis, of the capacity and limitations you are working with on the microcontroller unit (MCU).

You can monitor device space and memory with the Resource Meter. If you do not see the Resource meter, select **Resource Meter** from the **View** menu. Resources are updated as each user module is placed.

The resource meter tracks Analog Blocks, Digital Blocks, RAM, ROM, and the use of device specific special resources such as the decimator, CapSense™ blocks, or I^{2}C controller. As you place user modules, you can view how many analog and digital PSoC blocks you have available and how many you have used. RAM and ROM monitors track the amount RAM and ROM required to employ each selected user module.

Figure 2-39. PSoC Block Resource Meter


Device Resource Meter

Digital Blocks Total: 4 Used: 3

Analog Blocks Total: 6 Used: 4

RAM Total: 1024 Used: 8

ROM Total: 16384 Used: 519

Decimator Total: 1 Used: 1

I2C Controller Total: 1 Used: 0

2.10 Design Rule Checker

The Design Rule Checker (DRC) operates on a collection of predetermined rules associated with elements in a project database. After started, the DRC runs and then communicates the results of a “rule” evaluation.

The DRC is designed to point out potential errors or rule violations in your project that might eventually pose problems. The DRC does not impose limitations or prevent you from proceeding with your project “as is.” It simply notifies you of PSoC user module, software, and hardware elements you may not be aware of when configuring and sourcing your device. It is an additional tool to provide support for user-configuration.

The PSoC Designer collection of rules is being updated on an ongoing basis. A few sample DRC rules include:

- A project uses a Phase Locked Loop (PLL) but has not been configured with an External Crystal
- The device is set to 24 MHz and 3V operation
- The device is set to 48 MHz and 3V for Digital Clock Operation
- Failure to set required parameters or connections
- P0[1] and P0[0] Pins not High-Z with External Crystal
- 3.3 V Indicating ICE is 5V Supply Only
- Global Bus with Signal Pulse Width < 1/12 MHz
- Phase Consistency Between Output to Input of SC blocks
- PWM/Counter/Timer with Pulse Width > Period
- Inappropriate Ground and Reference Level Selections

To run the Design Rule Checker, go to: Tools > Add-in Tools > Design Rule Checker.

In a matter of seconds, you can review the results of the rule evaluation in the Output window after selecting "Code Generation" in the "Show output from box".

You can run the DRC at any time or any number of times during project development. To run it automatically each time you generate application files, go to: Tools > Options... > Tools.

2.11 Generating Application Files

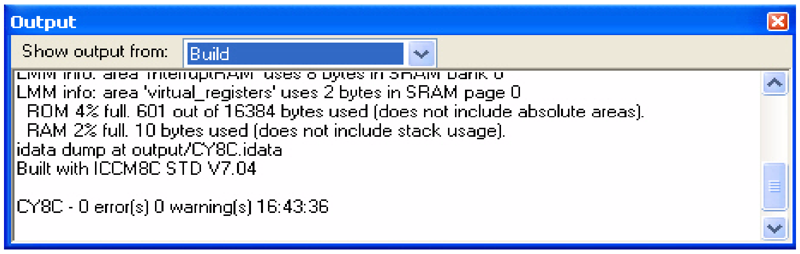
Generating application files is the final step to configuring your target device. When you generate application files, PSoC Designer takes all device configurations and updates existing assembly source and C Compiler code and generates API (Application Programming Interface) and ISR (Interrupt Service Routine) shells. At this time, the system also creates a data sheet based on your part configurations that can be accessed in the Chip-Level Editor (see “Configuration Data Sheets” on page 58). Generate Application is executed automatically during the code editor build process, when needed.

After this process is complete, you can enter the Code Editor and begin programming the desired functionality into your (now configured) device. For further details regarding programming, see “Code Editor” on page 66 and “Assembler” on page 77.

You can generate application files from any view. To do this, click the Generate/Build Application icon ▦.

Full details of the build are sent to the Output window. If the Output window is not visible, select **Output** from the **View** menu.

Figure 2-40. The Output Window with Build Messages



```
LMM info: area 'InterruptRAM' uses 8 bytes in SRAM bank 0
LMM info: area 'virtual_registers' uses 2 bytes in SRAM page 0
  ROM 4% full. 601 out of 16384 bytes used (does not include absolute areas).
  RAM 2% full. 10 bytes used (does not include stack usage).
idata dump at output/CY8C.idata
Built with ICCM8C STD V7.04

CY8C - 0 error(s) 0 warning(s) 16:43:36
```

NOTE: It is important to note that if you modify any device configurations, you must re-generate the application files before you resume source programming.

2.12 Source Files Generated by Generate Project Operation

The following table lists and describes the source files generated by the Generate Project operation.

Table 2-2. Source Files Generated by Generate Application

Name	Overwritten	Description
`…/lib/boot.asm`	Yes	Boot code and initial interrupt table
`…/lib/BuildMsg.txt`		
`…/lib/opts.txt`		
`…/lib/<User module name>.asm`	Yes	User module API source
`…/lib/<User module name>.h`	Yes	User module API C include header
`…/lib/<User module name>.inc`	Yes	User module API assembly include
`…/lib/<User module name>INT.asm`	Yes*	User module interrupt .asm file (if needed)
`…/lib/<Project Name>_GlobalParams.h`	Yes	
`…/lib/<Project Name>_GlobalParams.inc`	Yes	Project parameters include
`…/lib/PSoCConfig.asm`	Yes	Configuration loaded upon system access
`…/lib/<Project Name>PSoCAPI.h`	Yes	Project API include header
`…/lib/<Project Name>PSoCAPI.inc`		
`…/lib/PSoCCOnfig.asm`		
`…/lib/PSoCCOnfigTBL.asm`	Yes	Contains chip configuration
`…/lib/PSoCGPIOINT.asm`		
`…/lib/PSoCGPIOINT.h`		
`…/lib/PSoCGPIOINT.inc`		

* User code markers can be used to preserve sections in the file.

NOTE: If you undo placement of a user module but leave it in your selected collection and generate application files, associated .asm files remain (just not be updated). If you undo placement and delete a user module from your collection then generate application files, all associated .asm files are deleted (removed from source tree project files).

2.12.1 About the boot.asm File

When device configuration files are generated, the *boot.asm* file is updated. Among other things, this file includes a jump table for interrupt handlers. (Additional details regarding this file are found in "File Definitions and Recommendations" on page 93.)

The entries in the interrupt table are handled automatically for interrupts employed by user modules. For example, a Timer8 User Module uses an interrupt. The interrupt-vector number depends on which PSoC block is assigned to the Timer8 instance; vector 2 for PSoC digital block 0, vector 3 for block 1, and so on.

During the device configuration process, the ISR name is added to the appropriate interrupt-vector number. The interrupt handler is included in a file that is named instance_nameINT.asm, where *instance_name* is the name given to the user module. For example, if the user module is named Timer8_1, then the ISR source file is named *Timer8_1INT.asm*. All API files generated during the device configuration process follow this naming convention. The following are the API files that would be generated for a user module named Timer8_1:

- *Timer8_1.inc*
- *Timer8_1.h*
- *Timer8_1.asm*
- *Timer8_1INT.asm*

The *boot.asm* file is based on a file named *boot.tpl*. You can make changes to *boot.tpl* and those changes are reflected in *boot.asm* whenever the application is generated. Do not change any strings with the form `` `@INTERRUPT_nn` where nn = 0 to 15``. These substitution strings are used when device configuration application files are generated. However, you can replace substitution strings if you safely define the interrupt vector and install your own handler. If there is no interrupt handler for a particular interrupt vector, the comment string "`// call void_handler`" is inserted in place of the substitution string.

NOTE: If you install an interrupt handler and make changes directly to *boot.asm*, the changes are not preserved if application generation is executed after you make the changes. If you make changes to *boot.asm* that you do not want overwritten, hard code the change in *boot.tpl* (template for *boot.asm*).

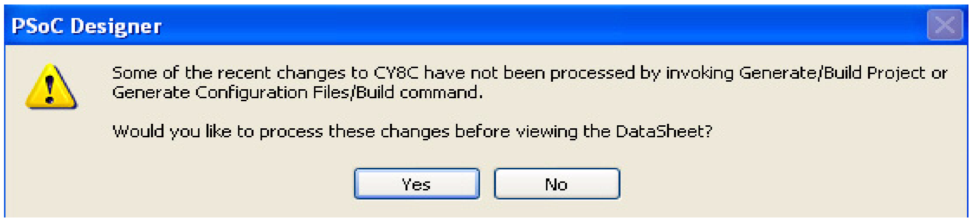
2.13 Configuration Data Sheets

After you have configured your device and generated application files, you can produce, view, or print a data sheet based upon how you configured your project device. The configuration data sheet is self-contained in its own folder in the project directory and can be viewed independently of PSoC Designer by opening configreport.xml in Internet Explorer. (If you need to move or send someone the file, you must move/send the entire directory of \ConfigDataSheet.)

To produce and view a data sheet:

1. Select View > Configuration Data Sheet for 'project_name'. This opens an independent browser window that shows the current configuration data sheet.

   NOTE: If changes were made in the Chip Editor and a "Generate Application" was not performed, then PSoC Designer shows a dialog box asking for verification before performing Generate Application. If you select No, then the data sheet reflects the configuration after the changes were made.



PSoC Designer

Some of the recent changes to CY8C have not been processed by invoking Generate/Build Project or Generate Configuration Files/Build command.

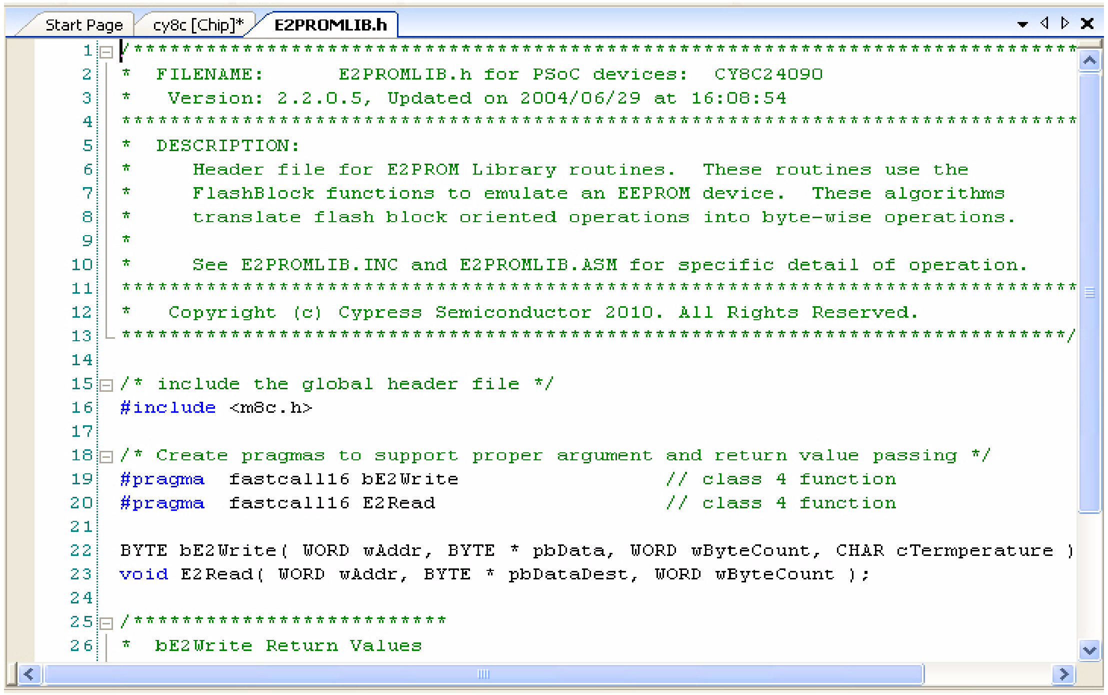
Would you like to process these changes before viewing the DataSheet?

2. To print the data sheet click the standard Print... button or File > Print..

2.14 APIs and ISRs

APIs (Application Programming Interfaces) and ISRs (Interrupt Service Routines) are also generated during the device configuration process in the form of **INT.asm*, *.h*, and *.inc* files. These files provide the device interface and interrupt activity framework for source programming. Figure 2-41, illustrates an .h file for configurations of a 16-bit PWM (Pulse Width Modulator) created during application-code generation:

Figure 2-41. PWM_FAN0.h

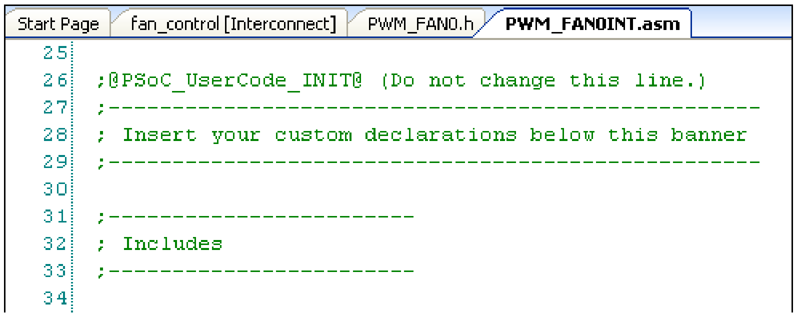


```
/*****************************************************************************
 *  FILENAME:      E2PROMLIB.h for PSoC devices:  CY8C24090
 *   Version: 2.2.0.5, Updated on 2004/06/29 at 16:08:54
 *****************************************************************************
 *  DESCRIPTION:
 *     Header file for E2PROM Library routines.  These routines use the
 *     FlashBlock functions to emulate an EEPROM device.  These algorithms
 *     translate flash block oriented operations into byte-wise operations.
 *
 *     See E2PROMLIB.INC and E2PROMLIB.ASM for specific detail of operation.
 *****************************************************************************
 *   Copyright (c) Cypress Semiconductor 2010. All Rights Reserved.
 ****************************************************************************/

/* include the global header file */
#include <m8c.h>

/* Create pragmas to support proper argument and return value passing */
#pragma  fastcall16 bE2Write                 // class 4 function
#pragma  fastcall16 E2Read                   // class 4 function

BYTE bE2Write( WORD wAddr, BYTE * pbData, WORD wByteCount, CHAR cTemperature )
void E2Read( WORD wAddr, BYTE * pbDataDest, WORD wByteCount );

/**************************
 *  bE2Write Return Values
```

After you generate the device configuration application code, the files for APIs and ISRs are located in the source tree of Workspace Explorer under the Library Source Files and Library Header Files folders.

NOTE: If you modify any ISR file and then re-generate your application, changes **are not** overwritten if they are placed between user code markers included in the **int.asm* file. Source code outside of the user code marker regions is overwritten and is always re-generated. However, if a user module is renamed and the application is re-generated, any user modifications within the user code markers are not updated with the instance name. Any use of the user module instance name within user code markers must be manually updated.

Figure 2-42. Place Your Custom Code Here



```

;@PSoC_UserCode_INIT@ (Do not change this line.)
;---------------------------------------------------
; Insert your custom declarations below this banner
;---------------------------------------------------

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

```

2.14.1 Working with ISRs

The interrupts depends on the device that we use:

- Reset
- Supply Monitor

- 4 Analog Columns
- VC3
- GPIO
- 16 Digital Blocks
- I2C
- Sleep Timer

The configurable interrupts include 16 digital blocks and 4 analog columns. The definition (for example, interrupt vector action) of a configurable interrupt depends on the user module that occupies the block or uses the analog column.

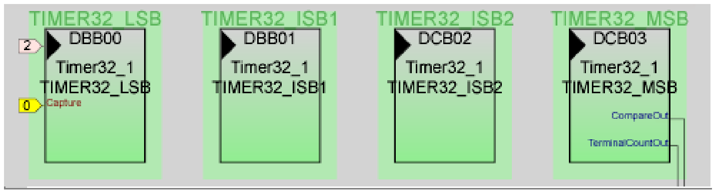
The Chip-Level Editor handles the details of getting user module parameters into source code, so that the project is configured correctly at startup and exposes subroutines that make for ease-of-use. Exposing subroutines that make user module parameters easy to use involves PSoC Designer adding files to your project. These files are known as Application Program Interfaces (APIs). Typically, one of these user module files, added to your project, is an interrupt handler.

Aside from adding API files to your project, the Chip-Level Editor also inserts a call or jump to the user module's interrupt handler in the startup source file, *boot.asm*.

2.14.2 Interrupt Vectors and the Chip-Level Editor

Figure 2-43 shows an example of how an interrupt handler is dispatched in the interrupt vector table, using a device from the CY8C27xxx part family. Shown below is the Timer32 User Module mapped to PSoC blocks 00, 01, 02, and 03. An interrupt is generated by the hardware when terminal count is reached. The last PSoC Block (or MSB byte) of Timer32 generates the terminal count interrupt.

Figure 2-43. Timer32 on Four Digital PSoC Blocks



When the application is generated, code is produced for the Timer32_1 User Module. The interrupt vector table is also altered with the addition of the call to the timer interrupt handler in *boot.asm*.

```
AREA TOP (ROM, ABS, CON)
    org   0                      ;Reset Interrupt Vector
IF    (TOOLCHAIN & HITECH)
;   jmp   __Start                ;C compiler fills in this vector
ELSE
    jmp   __Start                ;First instruction executed following a Reset
ENDIF
    org   04h                    ;Low Voltage Detect (LVD) Interrupt Vector
    halt                         ;Stop execution if power falls too low
    org   08h                    ;Analog Column 0 Interrupt Vector
    // call void_handler
    reti
    org   0Ch                    ;Analog Column 1 Interrupt Vector
    // call void_handler
    reti
    org   10h                    ;Analog Column 2 Interrupt Vector
    // call void_handler
    reti
    org   14h                    ;Analog Column 3 Interrupt Vector
    // call void_handler
    reti
    org   18h                    ;VC3 Interrupt Vector
    // call void_handler
    reti
    org   1Ch                    ;GPIO Interrupt Vector
    // call void_handler
    reti
    org   20h                    ;PSoC Block DBB00 Interrupt Vector
    // call void_handler
    reti
    org   24h                    ;PSoC Block DBB01 Interrupt Vector
    // call void_handler
    reti
    org   28h                    ;PSoC Block DCB02 Interrupt Vector
    // call void_handler
    reti
    org   2Ch                    ;PSoC Block DCB03 Interrupt Vector
    ljmp    _Timer32_1_ISR
```

```
reti
org   30h                        ;PSoC Block DBB10 Interrupt Vector
// call void_handler
reti
org   34h                        ;PSoC Block DBB11 Interrupt Vector
// call void_handler
reti
org   38h                        ;PSoC Block DCB12 Interrupt Vector
// call void_handler
reti
org   3Ch                        ;PSoC Block DCB13 Interrupt Vector
// call void_handler
reti
org   60h                        ;PSoC I2C Interrupt Vector
// call void_handler
reti
org   64h                        ;Sleep Timer Interrupt Vector
// call void_handler
reti
```

Table 2-3 shows how *boot.asm* vector names map to fixed, analog column, and PSoC block (configurable) interrupts. Not all of these are shown in the code example.

Table 2-3. *boot.asm* Interrupt Names

Address	Data Sheet Interrupt Name	Type
00h	Reset	Fixed
04h	Supply Monitor	Fixed
08h	Analog Column 0	Analog Column
0Ch	Analog Column 1	Analog Column
10h	Analog Column 2	Analog Column
14h	Analog Column 3	Analog Column
18h	VC3	Fixed
1Ch	GPIO	Fixed
20h	DBB00	PSoC Block
24h	DBB01	PSoC Block
28h	DCB02	PSoC Block
2Ch	DCB03	PSoC Block
30h	DBB10	PSoC Block
34h	DBB11	PSoC Block
38h	DCB12	PSoC Block
3Ch	DCB13	PSoC Block
40h	DBB20	PSoC Block

Table 2-3. *boot.asm* Interrupt Names *(continued)*

44h	DBB21	PSoC Block
48h	DCB22	PSoC Block
4Ch	DCB23	PSoC Block
50h	DBB30	PSoC Block
54h	DBB31	PSoC Block
58h	DCB32	PSoC Block
5Ch	DCB33	PSoC Block
60h	I2C	Fixed
64h	Sleep Timer	Fixed

Continuing the example, 2Ch corresponds to DCB03. There are no interrupt handlers at DBB00, DBB01, and DCB02 (20h, 24h, and 28h) because a 32-bit Timer User Module only requires the interrupt at the end of the chain.

In many cases the actual interrupt handling code is “stubbed” out. You can modify the content of this stubbed handler to suit your needs. Any subsequent device reconfiguration will not overwrite your work in the handler if the modification is done in *boot.tpl*.

2.15 Dynamic Reconfiguration

The PSoC resources are configured using latch-based registers. These registers can be changed on-the-fly, allowing for new functions to be created as needed during the execution of the application program.

Reconfiguring resources in this manner is called Dynamic Reconfiguration. User modules are an abstraction of register settings that enable a high-level function. A set of user modules is called a configuration. The application can switch in and out of configurations in real-time, allowing for overuse of the chip resources. This is akin to memory overlaying. It is up to the application to ensure that configurations are not reconfigured while they are being used.

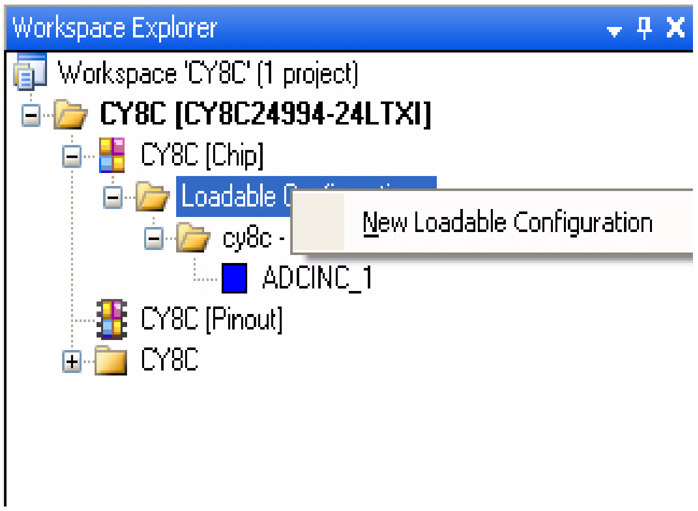
A loadable configuration consists of one or more placed user modules with module parameters, Global Resources, set pinouts, and generated application files. PSoC projects can consist of one or multiple loadable configurations.

2.15.1 Adding Configurations

To add loadable configurations to your **PSoC project**:

1. Right click the **Loadable Configuration** folder in the Workspace Explorer and select **New Loadable Configuration**.

Figure 2-44. Add a New Loadable Configuration



2. You see a new folder with a default name of Config*x* where *x* is the number of alternate configurations.

   Select the configuration folders to switch from one configuration to the other.

   There is always at least one folder with the project name when a project is created. This folder represents the base configuration. The base configuration has special characteristics. You cannot delete the base configuration. The new configuration, by default, has global settings and pin settings identical to the base configuration. Additional configuration folders appear in alphabetical order from left to right, beginning after the base configuration tab.
3. To change the name right-click the folder and select **Rename**. The new name appears on the folder.

   NOTE: One requirement for Dynamic Reconfiguration is that user module instance names must be unique across all configurations. This requirement eliminates confusion in code generation. Otherwise, all other icon and menu-item functions are identical to projects that do not employ additional configurations.
4. Proceed with the configuration process (i.e., selecting and placing user modules, setting up parameters, and specifying pinout).

2.15.2 Deleting Configurations

To delete a loadable configuration from your **PSoC project**:

1. Right click on the loadable configuration and select **Delete**.

After you delete a configuration, all associated source files are removed from the project (if application files were generated).

2.15.3 Renaming Configurations

To rename a loadable configuration in your **PSoC project**:

1. Right click the loadable configuration and click **Rename**.
2. Type the new name.
3. Press [**Enter**] or click your cursor somewhere outside the folder.

2.15.4 Employing Dynamic Reconfiguration

These sections discuss how global parameters, pin settings, and code generation are dynamically reconfigured.

2.15.4.1 Global Parameters

When using Dynamic Reconfiguration, global parameters are set in the same manner as single configurations. However, changes to the base configuration global parameters are propagated to all additional configurations. Therefore, global parameter changes made to an additional configuration are done locally to that particular configuration. For instance, if some global parameter #1 has specific value in “Base” loadable configuration, this global parameter #1 will have the same value in “New” loadable configuration. But not vice versa: if some global parameter #2 has specific value in “New” loadable configuration, this global parameter #2 will have the default (or “Base” specific) value in “Base” loadable configuration.”

2.15.4.2 Port Pin Settings

When using Dynamic Reconfiguration, port pin settings are similar to global parameters in that all settings in the base configuration are propagated to additional configurations. When manually set, port pin settings become local to the configuration. For instance, if Port_x_y pin is occupied (or has the specific value of Select, Drive, Interrupt, Initial Value parameters) in “Base” loadable configuration, this Port_x_y will be also occupied (or will have the specific value of Select, Drive, Interrupt, Initial Value parameters) in “New” loadable configuration. But not vice versa: if Port_x1_y1 pin is occupied (or has the specific value of Select, Drive, Interrupt, Initial Value parameters) in “New” loadable configuration, this Port_x1_y1 will be free in “Base” loadable configuration.

To set port pin interrupts:

1. Open the Chip Editor of Chip-Level Editor.
2. Click the pin you want to set and select the Interrupt type you want.
3. The default pin interrupt setting is Disable. If all pin interrupts are set to disable, there is no additional code generated for the pin interrupts. If at least one pin is set to a value other than disable, code generation performs some additional operations.

In the *boot.asm* file, the vector table is modified so that the GPIO interrupt vector has an entry with the name PSoC_GPIO_ISR. Additional files being generated are:

- *PSoCGPIOINT.asm*
- *PSoCGPIOINT.inc*

PSoCGPIOINT.asm – This file contains an export and a placeholder so you can enter its pin interrupt handling code. Enter user code between the user code markers where appropriate. This file is re-generated for each code generation cycle, but the user code will be carried forward if it is within the user code markers.

NOTE: When opening an old project that contains a *PSoCGPIOINT.asm* file where user code is entered, the user code must be copied from the backup copy in the `\Backup` folder into the newly generated *PSoCGPIOINT.asm* file.

PSoCGPIOINT.inc – This file contains equates that are useful in writing the pin interrupt handling code. For each pin (with enabled interrupt or custom name), a set of equates are generated that define symbols for the data address and bit, and for the interrupt mask address and bit associated with the pin. The naming convention for the equates is:

- `CustomPinName_Data_ADDR`
- `CustomPinName_MASK`
- `CustomPinName_IntEn_ADDR`

- `CustomPinName_Bypass_ADDR`
- `CustomPinName_DriveMode_0_ADDR`
- `CustomPinName_DriveMode_1_ADDR`
- `CustomPinName_IntCtrl_0_ADDR`
- `CustomPinName_IntCtrl_1_ADDR`

The `CustomPinName` used in the substitution is replaced by the name entered for the pin during code generation. Custom pin naming allows you to change the name of the pin. The name field is included in the pin parameter area of the pinout diagram.

The Name column in the Pin Parameter Grid shows the names assigned to each of the pins. The default name shows the port and bit number. To rename the pin, double-click the name field and type the custom name. Note that the name must not include any embedded spaces.

The effect of the name is primarily used in code generation when the pin interrupt is enabled. The pin name is appended to the equates that are used to represent the address and bit position associated with the pin for interrupt enabling and disabling, as well as testing the state of the port data.

2.15.4.3 Code Editor

There are no direct changes in Code Editor with regards to Dynamic Reconfiguration. The additional files generated are placed in the Library Source and Library Headers folders of the source tree. Library source files that are associated with an additional configuration are shown under the SAME folder that files of a base configuration are shown under. This partitions the files so that the source tree view is not excessively long.

2.15.4.4 Code Generation

When configurations are present, additional code is generated to enable the application to load or operate with the configurations. *PSoCConfig.asm* is generated.

PSoCConfig.asm

The static file *PSoCConfig.asm* contains:

Exports and code for:

- `LoadConfigInit` – Configuration initialization function
- `LoadConfig_projectname` – Configuration loading function

Code only for:

- `LoadConfig` – General load registers from a table

For projects with additional configurations, a variable is added to the project that tracks the loaded configurations. The `LoadConfig_projectname` function sets the appropriate bit in the active configuration status variable.

Additional functions named `LoadConfig_ConfigurationName` are generated with exports that load the respective configuration.

For each `LoadConfig_xxx` function, an `UnloadConfig_xxx` function is generated and exported to unload each configuration, including the base configuration. The `UnloadConfig_xxx` functions are similar to the `LoadConfig_xxx` functions except that they load an `UnloadConfigTBL_xxx_Bankn` table and clear a bit in the active configuration status variable. In these functions, the global registers are restored to a state that depends on the currently active configuration.

With regard to the base configuration, `UnloadConfig_xxx` and `ReloadConfig_xxx` functions are also generated. These functions load and unload only user modules contained in the base configuration. When the base configuration is unloaded, the `ReloadConfig_xxx` function must be used to restore the base configuration user modules. The `ReloadConfig_xxx` function ensures the integrity of the write only shadow registers. Respective load tables are generated for these functions in the *PSoCConfigTBL.asm* file.

An additional unload function is generated as `UnloadConfig_Total`. The `UnloadConfig_Total` function loads these tables:

- `UnloadConfigTBL_Total_Bank0`
- `UnloadConfigTBL_Total_Bank1`

These tables include the unload registers and values for all PSoC blocks. The active configuration status variable is also set to ‘0’. The global registers are not set by this function.

The name of the base configuration matches the name of the project. The project name is changed to match the base configuration name if you change the name of the base configuration (from the project name).

A C callable version of each function is defined and exported so that these functions are called from a C program.

PSoCConfigTBL.asm

The *PSoCConfigTBL.asm* file contains the personalization data tables used by the functions defined in *PSoCConfig.asm*. For static configurations, there are only two tables defined; `LoadConfigTBL_projectname_Bank0` and `LoadConfigTBL_projectname_Bank1`, which support the `LoadConfig_projectname` function. These tables personalize the entire global register set and all registers associated with PSoC blocks that are used by user modules placed in the project.

For projects with additional configurations, a pair of tables are generated for each `LoadConfig_xxx` function generated in *PSoCConfig.asm*. The naming convention follows the same pattern as `LoadConfig_xxx` and uses two tables: `LoadConfigTBL_xxx_Bank0` and `LoadConfigTBL_xxx_Bank1`. These tables are used by `UnloadConfig_xxx`. The labels for these tables are exported at the top of the file.

Loading – The tables for the additional configurations’ loading function differ from the base configuration load table. The additional configuration tables only include those registers associated with PSoC blocks that are used by user modules placed in the project, the global registers with settings that differ from the base configuration. If the additional configuration has no changes to the global parameters or pin settings, only the placed user module registers are included in the tables.

Unloading – The tables for additional configurations’ unloading functions include registers that de-activate any PSoC blocks that were used by placed user modules, and all global registers which were modified when the configuration was loaded. The registers and the values for the PSoC blocks are determined by a list in the device description for bit fields to set when unloading a user module, and are set according to the type of PSoC block. The exceptions are the `UnloadConfigTBL_Total_Bankn` tables, which include the registers for unloading all PSoC blocks.

boot.asm

The *boot.asm* file is generated similarly to a project that has no additional configurations, unless there are one or more configurations that have user modules placed in such a way that common interrupt vectors are used between configurations. In this case, the vector entry in the interrupt vector table will show the line `ljmp Dispatch_INTERRUPT_n` instead of a user module defined ISR.

2.15.4.5 *PSoCDynamic Files*

Four files are generated when additional configurations are present in a project:

- *PSoCDynamic.inc*
- *PSoCDynamic.asm*
- *PSoCDynamicINT.asm*
- PSoCDynamic.h

PSoCDynamic.inc

The *PSoCDynamic.inc* file is always generated. It contains a set of equates that represent the bit position in the active configuration status variable, and the offset to index the byte in which the status bit resides, if the number of configurations exceeds eight. A third equate for each configuration indicates an integer index representing the ordinal value of the configuration.

PSocDynamic.asm

The *PSoCDynamic.asm* file is always generated. It contains exports and functions that test whether or not a configuration is loaded. The naming convention for these functions is `IsOverlayName-Loaded`.

PSoCDynamicINT.asm

The *PSoCDynamicINT.asm* file is generated only when the user module placement between configurations results in both configurations using a common interrupt vector. The reference to the `Dispatch_INTERRUPT_n` function is resolved in this file. For each conflicting interrupt vector, one of these ISR dispatch sets is generated. The ISR dispatch has a code section that tests the active configuration and loads the appropriate table offset into a jump table immediately following the code. The length of the jump table and the number of tests depends on the number of user modules that need the common vector, rather than the total number of configurations. The number of conflicts can equal the number of configurations, if each configuration utilizes the common interrupt vector. Generally, there will be fewer interrupt conflicts on a per-vector basis.

PSoCDynamic.h

The PSoCDynamic.h file is always generated. It contains externs of functions prototypes that load, reload, unload a separate dynamic configuration, unload all configurations, and test whether or not a configuration is loaded.

2.15.4.6 *Limitations*

The new displays are based on a bitmap of loaded configurations maintained by the `LoadConfig` and `UnloadConfig` routines, which are generated by the Chip-Level Editor. This bitmap can get out of synchronization with the actual device configuration in several ways:

- The bitmap's RAM area can be accidentally overwritten.
- If overlapping (conflicting) configurations are loaded at the same time, the register labels will be scrambled.
- If an overlapping configuration is loaded and then unloaded, register labels from the original configuration will be used, even though some PSoC blocks will have been cleared by the last `UnloadConfig` routine.

3. Code Editor

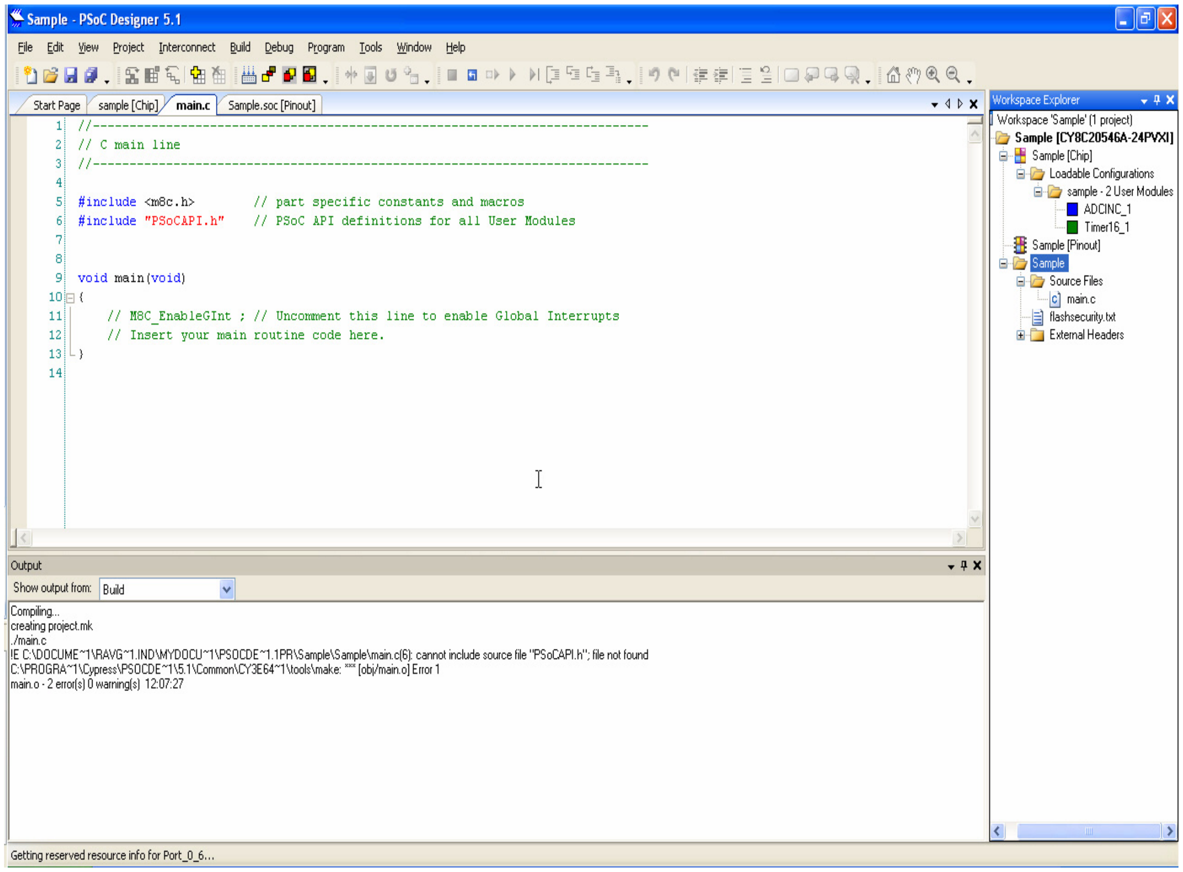
In this chapter you learn how to create the project code.

3.1 File Definitions and Recommendations

After you complete your device configuration, you are ready to create the application code. This is done in the Code Editor subsystem.

To access the Code Editor, double click any source file in the Workspace Explorer.

Figure 3-1. Code Editor View



```
//-----------------------------------------------------------------------------
// C main line
//-----------------------------------------------------------------------------

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


void main(void)
{
    // M8C_EnableGInt ; // Uncomment this line to enable Global Interrupts
    // Insert your main routine code here.
}
```

```
Compiling...
creating project.mk
./main.c
!E C:\DOCUME~1\RAVG~1.IND\MYDOCU~1\PSOCDE~1.1PR\Sample\Sample\main.c(6): cannot include source file "PSoCAPI.h"; file not found
C:\PROGRA~1\Cypress\PSOCDE~1\5.1\Common\CY3E64~1\tools\make: *** [obj/main.o] Error 1
main.o - 2 error(s) 0 warning(s)  12:07:27
```

The Workspace Explorer is shown in the right frame of Figure 3-1. This tree maintains the list of files that include configurations files, user module source and header files, boot files, and user application code.

3.1.1 File Types and Extensions

When you create a project, a root directory and "backup" is the fourth folder that is included in the root directory location when a new project is created. The name of the root directory is the project name and the names of the three folders are lib (Library), obj (Objects), and output (for files generated by a project build).

- The lib folder contains user module Library Source and Library Header files.
- The obj folder contains intermediate files generated during the compiling/assembling of *.c* and assembly source files.
- The output folder contains the *project.hex* file (used for debugging and device programming), the listing file, and other files that contain debug information.

Table 3-1 lists the PSoC Designer project file types and extensions. Most of these files are editable and appear in the left frame of the system interface inside the folder bearing the project name. For more details regarding files and recommended usage see "Project File System" on page 71.

Table 3-1. File Types and Extensions

Type	Extension	Location	Description
Address Map	*.mp*	…`\output` folder under project directory	Generated during the build process. Identifies global symbol addresses and other attributes of output.
ASM Include[a]	*.inc*	ASM Include Headers in source tree	Editable Assembly language include file (generated for APIs).
Assembly of C	*.s*	Found near the C file	Assembly generated from the C source code.
Assembly Source[a]	*.asm*	Source Files\ Library Source in source tree	Editable assembly language source file (created initially, added, or generated for APIs).
C Header	*.h*	C Headers in source tree	Editable language include file (generated for APIs).
C Source[a]	*.c*	Source Files in source tree	Compiler language file that can be added to the project.
CFG File	*.cfg*	Folder under project directory	Project configuration file that can be imported and exported for Dynamic Reconfiguration.
CMX File	*.cmx*		
Debug Symbols	*.dbg*	…\output folder under project directory	Generated during the build process. Used by the Debugger subsystem.
Full Program Listing	*.lst*	…\output folder under project directory	Full program listing. Used by the Single-Step ASM function.
HEX File	*.hex*	…\output folder under project directory	Output file in Intel HEX format generated during the build process. This file alone will be downloaded to the ICE for project debugging.
Library/Archive	*.a*	…\lib\libpsoc.a but Libraries can be anyplace	A collection of object files, created by *ilibw.exe*.
Make	*.mk*	Menu under Project > Open local.mk file	Customize the Build/Make process for a particular PSoC Designer project.
Object Module	*.o*	…\obj folder under project directory	Intermediate, relocatable object file generated during assembly and compilation.
Project Database[a]	*.soc*	Project directory	Project file accessed under File > Open Project.
Relative Source Listing	*.lis*	…\obj folder under project directory	Relative address listing file generated by the assembler.

Table 3-1. File Types and Extensions<Italic> (continued)

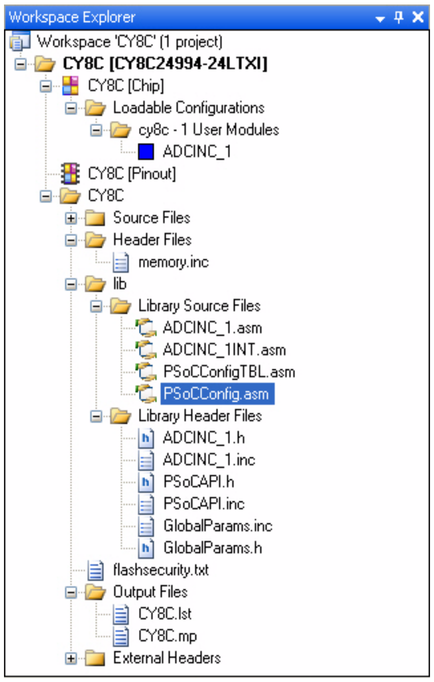
Type	Extension	Location	Description
ROM File	*.rom*	…\output folder under project directory	This file is a legacy (M8A M8B) program image output file.
Template	*.tpl*	Project directory	Editable template file.
Template[a]	*.tpl*	Installation directory under …\Templates then copied to project directory	Template files used to generate project files (*boot.tpl* > *boot.asm*).
Text Document	*.txt*	Project directory	Text document that contains system information.
WNP File	*.wnp*	Project directory	Persistence file unique to PSoC Designer. Contains project information restored each time the project is opened.
XML Document[a]	*.xml*	Project directory	Device resource file.

a. If you are using a version control system to track project process, copy the above checked files including *m8c.inc* (as the only .inc file) and not including *boot.asm* (as it is recreated during the device configuration process). Also include any **INT.asm* files that have been modified. All other project files will be regenerated during the device application configuration process.

3.1.2 Project File System

The project file system (workspace explorer) setup is like a standard file system. To access and edit files simply double-click the target file. Open files appear in the main window to the right of the source tree. The maximum number of characters allowed per line is 2,048.

Figure 3-2. Source Tree



```
Workspace Explorer
Workspace 'CY8C' (1 project)
  CY8C [CY8C24994-24LTXI]
    CY8C [Chip]
      Loadable Configurations
        cy8c - 1 User Modules
          ADCINC_1
    CY8C [Pinout]
    CY8C
      Source Files
      Header Files
        memory.inc
      lib
        Library Source Files
          ADCINC_1.asm
          ADCINC_1INT.asm
          PSoCConfigTBL.asm
          PSoCConfig.asm
        Library Header Files
          ADCINC_1.h
          ADCINC_1.inc
          PSoCAPI.h
          PSoCAPI.inc
          GlobalParams.inc
          GlobalParams.h
      flashsecurity.txt
      Output Files
        CY8C.lst
        CY8C.mp
      External Headers
```

The source tree contains these file folders:

Chip Folder – Contains the Loadable Configurations folder that contains one or more configurations. Each of the loadable configurations contains the user modules for the configuration. For more information see “Dynamic Reconfiguration” on page 63.

Source Files Folder – Contains assembly language code and C Compiler files generated by the system and user modules.

Headers Files and Library Headers Files – Contains include files added by device configurations and user modules.

Library Source Files – Contains the project configuration *.asm* as well other project-specific reference files generated by device configuration.

The .lst and .mp files are always available in the source tree under Output Files folder. Because these files are generated output from your assembled and linked source, they are read only.

3.1.3 *boot.asm*

This startup file resides in the source tree under Source Files and is important because it defines the boot sequence. The components of the boot sequence are:

- Defines and allocates the reset and interrupt vectors.
- Initializes device configuration.
- Initializes C environment if using the C Compiler.
- Calls `main` to begin executing the application code.

When a project is created, the template file, *boot.tpl*, is copied into the project directory. Each time the project is generated, the *boot.asm* file is generated from the local *boot.tpl* file.

boot.asm is re-generated every time device configurations change and application files are generated. This is done to make certain that interrupt handlers are consistent with the configuration. If you make changes to *boot.asm* that you do not want overwritten, modify the local project *boot.tpl* file and then re-generate file.

3.1.4 *main.asm/main.c*

If the C complier is not enabled, then the *main.asm* file is generated for applications written in Assembly language. If the C Compiler is enabled, the *main.c* file is generated for a C program. This file resides in the source tree under Source Files and is important because it holds the `_main` label that is referenced from the boot sequence.

3.1.5 *PSoCConfig.asm*

This is a required Library Source file because it contains the configuration that is loaded at system power-up.

PSoC Designer overwrites *PSoCConfig.asm* when a device configuration changes and application files are regenerated, with no exceptions. To manipulate bits, all part register values reside in this file for your reference.

3.1.6 Additional Generated Files

Additional files are generated in association with user modules and Dynamic Reconfiguration.

PSoCGPIOINT.inc – This file contains additional information pertaining to pin GPIO write only register shadows. If a pin group is defined in a register set for which register shadows are allocated, then a set of three macros are defined for each register shadow to read, set, or clear the particular bit within the register associated with the pin. The names of the macros are keyed to the custom name assigned to the pin and are:

- `GetCustomName_registerName`
- `SetCustomName_registerName`

- `ClearCustomName_registerName`

`CustomName` is the custom name set for the pin, and `registerName` is the associated register name for which a register shadow is allocated.

The registerName registers vary with the chip device description and include all registers associated with the GPIO ports. For the CY8C25xxx/26xxx device family, registers include:

- Bypass
- DriveMode_0
- DriveMode_1
- IntCtrl_0
- IntCtrl_1
- IntEn

For all other PSoC device families, registers include:

- GlobalSelect
- DriveMode_0
- DriveMode_1
- DriveMode_2
- IntCtrl_0
- IntCtrl_1
- IntEn

The register shadow allocation is determined by user modules and Dynamic Reconfiguration. As the register allocation changes, the macro generation changes accordingly.

PSoCGPIOINT.h – This file contains the same information as *PSoCGPIOINT.inc* except that it is in a form needed for C code. In the case of the register shadows, this file does not generate macros, but rather defines a symbol that allows manipulation of the shadow as a global variable. For each register shadow associated with a custom pin definition, a variable named `CustomName _registerNameShadow` is defined, where `CustomName` and `registerName` are the same as previously defined for *PSoCGPIOINT.inc*. The variable name is then used to manipulate the shadow register. For example, to set a pin value to ‘1’ within the port, do this:

```
CustomName_registerNameShadow |= CustomName_MASK;
CustomName_registerName_ADDR = CustomName_registerNameShadow;
```

GlobalParams.h – This file has the same contents as *GlobalParams.inc*, except it also has `#define` statements.

3.2 Working in Code Editor

Before you begin adding and modifying files, take a few moments to navigate Code Editor, take inventory of your current files, and map out what you plan to do and how you plan to do it.

3.2.1 Modifying Files

When you are ready to program and modify C and assembly language source files, double-click the target file located in the file source tree. The file opens and appears in the main active window. You can open multiple files simultaneously. Table 3-2 details the menu options available for modifying source files.

Table 3-2. Menu Options for Modifying Source Files

Icon	Option	Menu	Shortcut	Feature
	Compile/Assemble	Build > Compile 'file_name'	[Ctrl] [F7]	Compiles or assembles the open, active file (*.c* or .asm)
	Build Current Project	Build > Build 'project_name' Project	[F7]	Builds the current project
	Generate and Build All Projects	Build > Generate/Build All Projects	[Shift] [F6]	Builds all the project
	Generate and Build Current Projects	Build > Generate/Build current project	[F6]	Generates configuration files Builds the entire project and links applicable files
	New File	File > New File...	[Ctrl] [N]	Adds a new file to the project
	Open File	File > Open File...	[Ctrl] [O]	Opens an existing file in the project
	Increase Indent			Increase Indents of
	Decrease Indent			Decrease Indents of
	Comment		[Ctrl][E]+[C]	Comments selected text
	Uncomment		[Ctrl][E]+[U]	Uncomments selected text
	Toggle Bookmark	Edit > Bookmarks > Toggle Bookmark	[Ctrl] [B] + [T]	Toggles the bookmark: Sets/ removes user-defined bookmarks used to navigate source files
	Clear Bookmarks	Edit > Bookmarks > Clear Bookmarks	[Ctrl] [B] + [C]	Clears all user-defined bookmarks
	Next Bookmark	Edit > Bookmarks > Next Bookmark	[Ctrl] [B] + [N]	Goes to next bookmark

Table 3-2. Menu Options for Modifying Source Files<Italic> (continued)

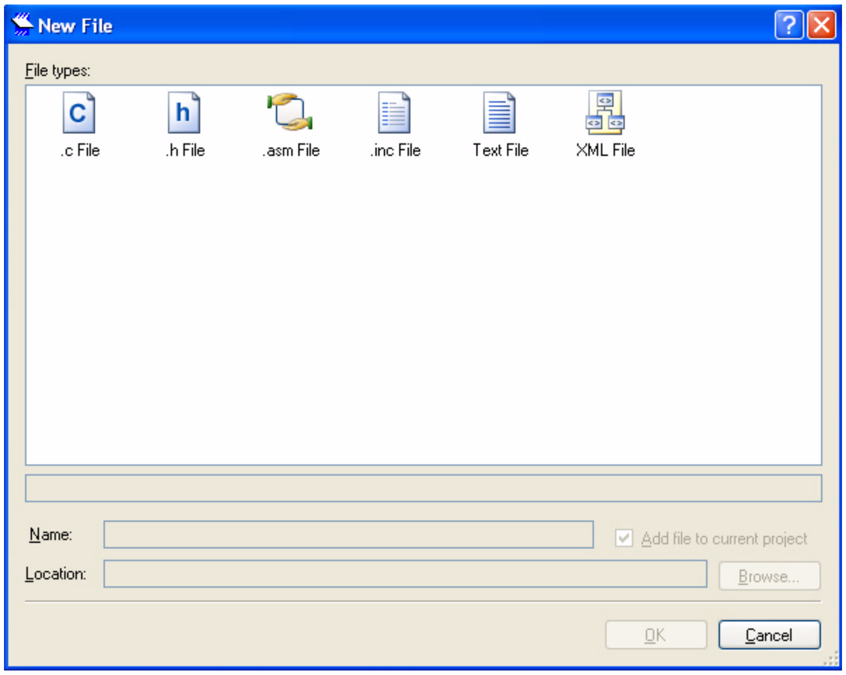
Icon	Option	Menu	Shortcut	Feature
	Previous Bookmark	Edit > Bookmarks > Previous Bookmark	[Ctrl] [B] + [P]	Goes to previous bookmark
	Find Text	Edit > Find and Replace	[Ctrl] [F]	Find specified text
	Undo	Edit > Undo	[Ctrl] [Z]	Undo last action
	Redo	Edit > Redo	[Ctrl] [Y]	Redo last action

3.2.2 Adding New Files

To add a file:

1. Click the **New File** icon or select **File > New File**...
2. In the New File dialog box, select a file from the File types.
3. In the Name field, type the name for the file.
4. The current project directory is the default destination for your file. Uncheck the Add to current project field and click **Browse...** to identify a different location if you do not want the default. The Browse button is only enabled if you uncheck the Add to current project field.

Figure 3-3. New File Dialog Box



5. When finished, click **OK**.

   Your new file is added to the file source tree and appears in the main active window.

3.2.3 Adding Existing Files

You are also able to add existing source files to your project (either C or assembly). Do this by accessing **Project > Add File...** and identifying the source file (by locating the file with the file dialog). Keep in mind that you add a copy of your original file to the project, not the original itself.

If the existing file you want to add is under a `lib` folder (...`\lib`), this file is added to the Library Source tree and resides in the `lib` folder of the project.

3.2.4 Removing Files

You can remove files from your project in one of two ways:

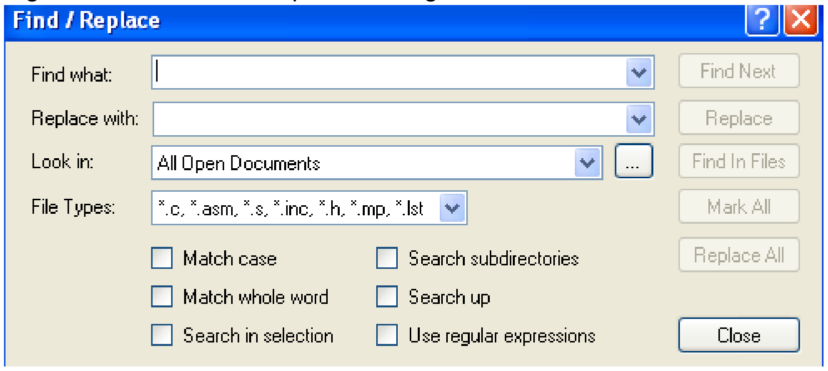
1. To remove the file, right-click on the file,in the source tree and Select **Exclude From Project.**
2. Go to **Project** menu > click **Exclude From Project**.

3.2.5 Searching Files

You can search for text in one or more files using the **Find** / **Replace** item in the **Edit** menu.

1. Click **Edit** > **Find and Replace**. The Find and Replace menu is enabled only when source code editor window is active.

Figure 3-4. Find and Replace Dialog Box



Look in: All Open Documents

File Types: *.c, *.asm, *.s, *.inc, *.h, *.mp, *.lst

2. In the **Find what** field, type the text that you want to search for or click the drop-arrow to choose a previous search pattern.
3. In the **Replace** with field, type the replacement text.
4. Select an option from **Look In** drop down list.
   - Select **Current** Document option to find a text in the same document.
   - Select **All Open** Document option to find a text in the all the documents.
   - Select **Current Project** option to find a text across the project.
5. Click **Browse** button to select the folder that will be searched.
6. To find next occurrences of the “**Find what**” text, click **Find Next** button.
7. To **replace** an occurrence of the “**Find what**” text, click **Replace** button.
8. To **replace all** occurrences of the “**Find what**” text, click **Replace All** button.
9. You can **mark all** the occurrences of the “**Find what**” text in the document using **Mark All** button.
10. If the **Match case** check box is selected, it searches only for strings that match the case of the “**Find what**” text.
11. If **Match whole word** check box is selected, it searches the whole “**Find what**” text surrounded by white space or punctuation.
12. If the **Search subdirectories** check box is selected, all subdirectory folders are searched.

13. If the **Search in selection** check box is selected, it searches only the text in the current selection.
14. The **Search up** check box is selected, the document is searched from the cursor position toward the begining of the document. This option is enabled only when you select the Current Document from the “**Look in**” drop down list.
15. If the **Use regular expressions** check box is checked, the find and replace text pattern fields will accept typical 'Use regular expressions' syntax.
16. Click the **Close** button to close the Find/Replace dialog.

Note The **Mark All** and **Replace All** functions work only when Look in option is set to **Current** Document or **All Open** Documents.

4. Assembler

In this chapter you receive high-level guidance on programming assembly language source files for the PSoC device. For comprehensive details, see the *PSoC Designer Assembly Language User Guide*.

4.1 Accessing the Assembler

The assembler is an application accessed from within PSoC Designer, much like the C Compiler. This application is run as a batch process. It operates on assembly language source to produce executable code. This code is then compiled and built into a single executable file that is downloaded into the In-Circuit Emulator (ICE), where the functionality of the PSoC device is emulated and debugged, aftger that it can be programmed into a target device.

The project source files appear in the left frame, called the source tree. Double-click individual files so they appear in the main active window where you add and modify code using the standard cut, copy, paste edit icons.

4.2 The M8C Microprocessor (MCU)

The Microprocessor (MCU) is an enhanced 8-bit microprocessor core. It is optimized to be small and fast.

There are five internal registers, see Table 4-1. All registers are 8 bits wide except the PC, which is composed of two 8-bit registers (PCH and PCL) which together form a 16-bit register.

Table 4-1. MCU Internal Registers

Register	Abbreviation
Accumulator	A
Flag	F
Index	X
Stack Pointer	SP
Program Counter	PC

4.2.1 Address Spaces

There are three separate address spaces implemented in the Assembler:

- **Register Space (REG)** – Accessed through the MOV and LOGICAL instructions. There are 8 address bits available to access the register space, plus an extended address bit via the Flag register bit 4.
- **Data RAM Space** – Contains the data/program stack and space for variable storage. All the read and write instructions, as well as instructions which operate on the stacks, use data RAM space. Data RAM addresses are 8 bits wide.

  The M8C is able to directly access 256 bytes of RAM. Some PSoC devices have more than a 256-byte RAM page. These devices access multiple RAM pages using a combination of page mode bits in the Flag and Paging registers of the register address space. See the PSoC device data sheets and the *PSoC Technical Reference Manual* for details.
- **Program Memory Space** - composed of the Supervisory ROM and the on-chip Flash program store. Flash is organized into 64-byte blocks. The user need not be concerned with program store page boundaries, as the M8C automatically increments the 16-bit PC register on every instruction making the block boundaries invisible to user code. Instructions occurring on a 256-byte Flash page boundary (with the exception of jmp instructions) incur an extra M8C clock cycle, as the upper byte of the PC register is incremented.

4.2.2 Instruction Format

Instruction addressing is divided into two groups:

- **Logic, Arithmetic, and Data Movement Functions (Unconditional)** –
  These are 1-, 2-, or 3-byte instructions. The first byte of the instruction contains the opcode for that instruction. In 2/3 byte instructions, the second/third byte contains either a data value or an address.
- **Jump and Call Instructions, including INDEX (Conditional)** –
  Most jumps, plus CALL and INDEX, are 2-byte instructions. The opcode is contained in the upper 4 bits of the first instruction byte and the destination address is stored in the remaining 12 bits. For program memory sizes larger than 4 KB, a 3-byte format is used.

4.2.3 Addressing Modes

Ten addressing modes are supported. For examples of each see the *PSoC Designer Assembly Language User Guide*.

- Source Immediate
- Source Direct
- Source Indexed
- Destination Direct
- Destination Indexed
- Destination Direct Source Immediate
- Destination Indexed Source Immediate
- Destination Direct Source Direct
- Source Indirect Post Increment
- Destination Indirect Post Increment

4.2.4 Destination of Instruction Results

The result of a given instruction is stored in the destination, which is placed next to the opcode in the assembly code. This allows for a given result to be stored in a location other than the accumulator. Direct and indexed addressed data RAM locations, as well as the X register, are additional destinations for some instructions. The AND instruction, in Table 4-2, is a good illustration of this feature (i2 = second instruction byte, i3 = third instruction byte). The ordering of the operands within the instruction determines where the result of the instruction is stored.

Table 4-2. Destination of AND Instruction

Syntax	Operation
AND A, expr	acc ← acc & i2
AND A, [expr]	acc ← acc & [i2]
AND A, [X + expr]	acc ← acc & [x + i2]
AND [expr], A	[i2] ← acc & [i2]
AND [X + expr], A	[x + i2] ← acc & [x + i2]
AND [expr], expr	[i2] ← i3 & [i2]
AND [X + expr], expr	[x + i2] ← i3 & [x + i2]

4.3 Assembly File Syntax

Assembly language instructions reside in source files with *.asm* extensions in the source tree of the Workspace Explorer. Each line of the source file may contain five keyword types of information. Table 4-3 supplies critical details about each keyword type.

Table 4-3. Keyword Types

Keyword Type	Critical Details
Label	Symbolic name followed by a colon (:)
Mnemonic	character string representing an M8C instruction
Operands	Arguments to M8C instructions
Expression	A command, interpreted by the Assembler, to control the generation of machine code
Comment	May follow operands or expressions and starts in any column if first non-space character is either a C++-style comment (//) or semi-colon (;).

Instructions in an assembly file have one operation on a single line. For readability, separate each keyword type by tabbing once or twice (approximately 5-10 white spaces). .

Avoid use of the following characters in path and file names (they are problematic): \ / : * ? " < > | &

+ , ; = [ ] % $ ` '.

4.4 List File Format

When you build a project, a listing file with an *.lst* extension is created. The listing shows how the assembly program is mapped into a section of code beginning at address 0. The linking (building) process will resolve the final addresses. This file doesn't provide a listing of errors and warnings! This file is created each time the build completes without errors or warnings.

.lst files are viewed after a project build in the Debugger subsystem under the Output Files folder of the source tree.

Also generated during a build (in addition to the *.lst* file) are *.rom*, *.mp*, *.dbg*, and *.hex* files. The *.hex* is used for debugging and programming. The *.mp* contains global symbol addresses and other attributes of output.

4.5 Assembler Directives

The PSoC Designer Assembler allows the assembler directives listed in Table 4-4. See the *PSoC Designer Assembly Language User Guide* for descriptions and sample listings of supported assembler directives.

Table 4-4. Assembler Directives

Symbol	Assembler Directive
AREA	Area
ASCIZ	NULL Terminated ASCII String
BLK	RAM Byte Block
BLKW	RAM Word Block
DB	Define Byte
DS	Define ASCII String
DSU	Define UNICODE String
DW	Define Word
DWL	Define Word with Little Endian Ordering
ELSE	Alternative Result of IF Directive
ENDIF	End Conditional Assembly
ENDM	End Macro
EQU	Equate Label to Valuable Value
EXPORT	Export
IF	Start Conditional Assembly
INCLUDE	Include Source File
.LITERAL, .ENDLITERAL	Prevent Code Compression of Data
MACRO/ENDM	Start Macro Definition Start/End
ORG	Area Origin
.SECTION, .ENDSECTION	Section for Dead-Code Elimination
Suspend - OR F,0 Resume - ADD SP,0	Suspend and Resume Code Compressor

4.6 Compile and Assemble Files

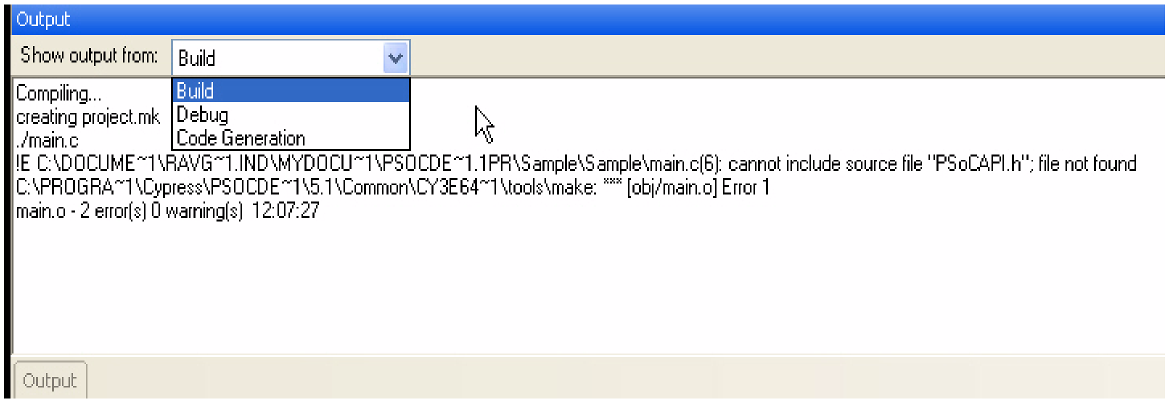
After you complete programming all assembly language source (in addition to any *.c* source), you are ready to compile and assemble the group of files. Compiling translates source code into object code. (The Linker then combines modules and supplies real values to symbolic addresses, thereby producing machine code.) Each time you compile and assemble, the most prominent, open source file is compiled. PSoC Designer can decipher the difference between C and assembly language files, and compile and assemble accordingly.

To compile the source files for your project, click the **Compile** icon .

PSoC Designer employs a make utility. Each time you click the Compile/Assemble or Build icon, the utility automatically determines which files of a large application (manual or generated) were modified and need recompiling, then issues commands to recompile them. For further details, see *make.pdf* in the `\Documentation\Supporting Documents` subdirectory of the PSoC Designer installation directory.

The Output (or error-tracking) window is where the status of file compiling and assembling resides. Each time you compile and assemble files, the Output window is cleared and the current status is entered as the process occurs.

Figure 4-1. Output Status Window



```
Compiling...
creating project.mk
./main.c
!E C:\DOCUME~1\RAVG~1.IND\MYDOCU~1\PSOCDE~1.1PR\Sample\Sample\main.c(6): cannot include source file "PSoCAPI.h"; file not found
C:\PROGRA~1\Cypress\PSOCDE~1\5.1\Common\CY3E64~1\tools\make: *** [obj/main.o] Error 1
main.o - 2 error(s) 0 warning(s)  12:07:27
```

When compiling is complete, you can see the number of errors. Zero errors signify that the compilation and assemblage was successful. One or more errors indicate problems with one or more files.

This process reveals syntax errors. Such errors include `missing input data` and `undeclared identifier`. For a list of all identified compile (and build) errors with solutions see the *PSoC Designer Assembly Language User Guide*. For further details on compiling and building see "Build Manager" on page 109 in this guide.

At any time you can ensure a clean compile and assemble (or build) by accessing **Build** > **Clean 'project_name' Project**, then clicking the **Compile** or **Build** icon. The "clean" deletes all `lib\libPSoc.a`, `obj\*.o`, and `lib\obj\*.o` files. These files are regenerated upon a compile or build (in addition to normal compile and build activity).

4.7 Calling Assembly Functions From C

When one C function calls another, the compiler uses a simple layout for passing arguments that the caller and callee use to initialize and the access the values. Although you can use the same layout when a C function calls an assembly language routine, it is best to use of the alternate fastcall16 calling convention. Fastcall16 is directly supported by the C compiler though use of a pragma directive and is often more efficient than the convention used by C. In fact, fastcall16 is identical to the C calling convention except for simple cases when the parameters are passed and/or returned in the CPU A and X registers. All user module API functions implement the fastcall16 interface for this reason.

There are four conditions to meet when using the fastcall16 interface:

- The function must be tagged with a C #pragma fastcall16 directive
- The function needs a C function prototype
- The assembly function name must be the C function name prefixed with an underscore character (_).
- The assembly function name must be exported.

For example, an assembly function that is passed a single byte as a parameter and has no return value looks like this:

C function declaration (typically in a *.h* header file)

```
#pragma fastcall16 send_byte
void send_byte(char val);
```

C function call (in a *.c* file)

```
send_byte(0x37);
```

Assembly function definition (in an *.asm* file)

```
export _send_byte
; Fastcall16 inputs (single byte)
;     A – data value
; Fastcall16 return value (none)
_send_byte:
mov reg[ PRT1DR],A
ret
```

An assembly function that is passed two bytes and returns one byte might look like this:

C function declaration (typically in a *.h* header file)

```
#pragma fastcall16 read_indexed_reg
char read_indexed_reg( char bank, char index);
```

C function call (in a *.c* file)

```
val = read_indexed_reg(0x01, index);
```

Assembly function definition (in an *.asm* file)

```
export read_indexed_reg
; Read byte from specified IO register
; Fastcall16 inputs (two single bytes)
;     A – bank number (0 or non-zero)
;     X – register number
; Fastcall16 return value (single byte)
;     A – read data
_read_indexed_reg:
   cpl A
   jnz get_data:
   or F, FLAG_XIO_MASK; switch to bank 1
get_data:
   mov A, reg[X]
   and F, ~FLAG_XIO_MASK; make sure we're in bank 0
   ret
```

Functions with more complex input parameters or return values can be written using these tables.

Table 4-5. Pragma Fastcall16 Conventions for Argument Passing

Argument Type	Register	Argument Register
Single Byte	A	The argument is passed in A.
Two Single Bytes	A, X	The first argument is passed in A, the second in X.
Double Byte	X, A	The MSB is passed in X, the LSB in A.
Pointer	A, X	The MSB is passed in A, the LSB in X.
All Others	None	Arguments are stored on the stack in standard byte order and in reverse order or appearance. In other words, the MSB of the last actual parameter is pushed first and the LSB of the first actual parameter is pushed last.

Table 4-6. Pragma Fastcall16 Conventions for Return Value

Return Type	Return Register	Comment
Single Byte	A	The argument is returned in A.
Double Byte	X, A	The MSB is passed in X, the LSB in A.
Pointer	A, X	The MSB is passed in A, the LSB in X.
All Others	None	Use a pass-by-reference parameter or global variable instead of returning arguments longer than 16 bits.

Note that the #pragma fastcall16 has replaced #pragma fastcall and use of #pragma fastcall is deprecated.

5. Build Manager

In this chapter you learn the details of building a project, discover more about the C Compiler as well as the basic, transparent functions of the system Linker and Loader, and Librarian. For comprehensive details on the C Compiler, see the *PSoC Designer C Language Compiler User Guide*.

5.1 Working in the Build Manager

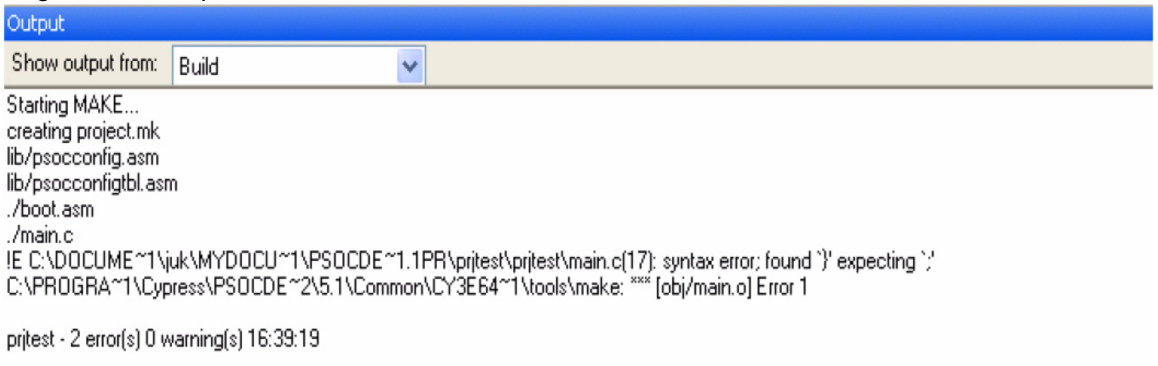
Building a project compiles and assembles source files and library source files selectively. PSoC Designer uses GNU Make version 3.79 to manage the build process. Each time you Compile or Build, make determines which source files were modified and issues commands to recompile them and link the project. For further details, see make.pdf in the `\Documentation\Supporting Documents` subdirectory of PSoC Designer install directory.

The build process performs the compile and assemble of project files then links to all the project's object modules (and libraries), creating a .hex file that is easily downloaded for debugging. To build the current project, either click the Build icon, select Build > Build 'project name' Project from the menu, or press [F7]. The build process creates object modules in the obj or `\lib\obj` subdirectory of the project directory. The linking produces the final project image in the output folder of the project directory.

The .hex file can be downloaded to the ICE. Other files in the folder provide references for the Debugger. Each compiler may generate a different file format and file extension. The .lst file contains a complete listing of the project, the .dbg file contains debug information, the .mp file contains a memory map, and the .idata file contains initialized data. At any time you can ensure a clean build by accessing Project > Clean 'project name' Project. The "clean" deletes all `lib\libpsoc.a`, `obj\*.o` and `lib\obj\*.o` files. (These are intermediate/object files generated during the compilation).

Each time you build your project, the details in the Output window is cleared and the current status is entered as the process occurs.

Figure 5-1. Output Window



```
Starting MAKE...
creating project.mk
lib/psocconfig.asm
lib/psocconfigtbl.asm
./boot.asm
./main.c
!E C:\DOCUME~1\juk\MYDOCU~1\PSOCDE~1.1PR\prjtest\prjtest\main.c(17): syntax error; found `}' expecting `;'
C:\PROGRA~1\Cypress\PSOCDE~2\5.1\Common\CY3E64~1\tools\make: *** [obj/main.o] Error 1

prjtest - 2 error(s) 0 warning(s) 16:39:19
```

When the build is complete, you can see the number of errors and warnings. Zero errors signify a successful build. One or more errors indicate problems with one or more files. If there are errors, the program image (.hex file) is available for download to the ICE. For a list of all identified compile and build errors with solutions see the Assembly Language User Guide or the C Compiler User Guide.

Table 5-1. Build Menu Options

Menu	Shortcut	Icon	Description
Generate/Build 'project name' Project	F6		Populates your project with APIs/libraries and builds the active project. If you add or remove User Modules after generating application files, you need to regenerate the application files as well as reconfigure required settings. For further details, see "“Generating Application Files” on page 56. If the project generation is disabled (locked) from the Project > Settings... > Chip Editor menu, this menu and shortcut would be grayed out
Generate/Build All Projects	Shift+F6		Same as above. All projects in the workspace will be regenerated and rebuilt
Generate Configuration Files for 'project name' Project	Ctrl+F6		This menu option populates your project with APIs/libraries. If you add or remove user modules after generating application files, you need to regenerate the application files as well as reconfigure required settings. For further details, see “Generating Application Files” on page 56”. If the project generation is disabled (locked) from the Project > Setting... > Chip Editor menu, this menu and shortcut would be grayed out.
Generate Configuration Files for All Projects			Same with above. All projects in the workspace will be regenerated.
Compile <file name>	Ctrl+F7		Compiles the source file that is active in the code editor
Build 'project name' Project	F7		Builds the active project. The make utility automatically determines which files were modified and need recompilation, and then issues commands to recompile them. For further detail, see make.pdf in the `Documentation\Supporting Documents` subdirectory of the PSoC Designer installation directory.
Rebuild 'project name' Project			Deletes all of the output files and build all source files.
Clean 'project name' Project			Deletes all lib\libpsoc.a, obj*.o, and lib\obj\*.o files. These files are regenerated upon a compile or build (in addition to normal compile and build activity).

5.2 C Compiler

In addition to the development tools provided by Cypress Semiconductor, third party development tools are available for PSoC devices. This gives developers a choice of tools when working with PSoC devices. For information on how to install and use third party compilers with PSoC Designer, refer to documentation supplied by the manufacturer of the tool.

The iMAGEcraft compiler enables you to quickly create a complete C application for a PSoC device. Its built-in macro assembler allows assembly language code to seamlessly merge with C code.

The compiler compiles each *.c* source file to an *.s* assembly file. The assembler then translates each *.asm* or *.s* file into a relocatable object file, *.o*. After all the files are translated into object files, the builder and linker combine them together to form an executable file.

The iMAGEcraft C Compiler comes complete with embedded libraries providing port and bus operations, standard keypad and display support, and extended math functionality. For comprehensive details on the C Compiler, see the *C Language Compiler User Guide*.

To set compiler options in PSoC Designer, select **Project** > **Settings...** > **Build** > **Compiler**. You can select a compiler option from the compilers you have installed. Depending on the compiler selected, the settings will differ.

5.2.1 ImageCraft Compiler Options

The ImageCraft specific compiler configuration options are as follows:

- **Macro defines** specifies macros on the command line to the compiler.
- **Macro undefines** undefines any predefined compiler macros.
- Checking **Optimize math functions for speed** causes math functions optimized for speed to be included in the application at the cost of additional Flash and/or RAM footprint.
- The **Enable paging** checkbox is used to enable or disable large memory model appliations (applications using more than 256 bytes of RAM) on target chips with more than 256 bytes of RAM. Unchecking this box for these chip restricts RAM usage to the first 256 bytes and decreases program execution time and size associated with manipulating RAM paging registers.
- **Stack page** is an indicator of the RAM page on which the stack will be allocated for a large memory model application.
- **Stack page offset** enables setting the start address of the stack for a large memory model application such that the stack page can be shared between the stack and static variables.
- **Code compression techologies** are used to reduce the application's Flash footprint.

Condensation (duplicate code), is a search of the binary code image for instruction sequences that occur multiple times. These instruction sequences are placed into subroutines. Each occurrence of a repeated instruction sequence is then replaced with a call to the applicable subroutine.

Sublimation (eliminate unused user module APIs) is the elimination of unused assembly code bounded by the .section and endsection directives in AREA UserModules. If execution flow does not go to the label immediately below the .section directive, the entire block of code up to the next endsection directive is removed.

Refer to the ImageCraft C Compiler Guide for more information.

5.2.2 HI-TECH Compliler Options

Listed below are the HI-TECH specific compiler configuration options:

- **Macro defines** allows you to define macros on the command line to the compiler.
- **Macro undefines** allows you to undefine any predefined compiler macros.
- **Warning Level** specifies the minimum warning message level allowed for output.
- **Optimization Settings**
  - Checking the Global checkbox enables global optimization and the Level dropdown list selects the global optimization level.
  - Checking the Assembler checkbox enables assembler optimization.
- Options allows you to enter any command line compiler options
- The Switch to Lite Mode button adds a command line option for the Pro compiler to compile using Lite mode. This button is applicable only to the Pro compiler and has no effect on the Lite compiler, which compiles in Lite mode regardless of this option setting.

Refer to the HI-TECH C(R) PRO for the PSoC(R) Mixed-Signal Array guide for more information.

5.3 Linker

The linking functions in the build process are transparent to the user. Building your project links all the programmed functionality of the source files (including device configuration) into a *.hex* file, which is the file used for downloading and debugging.

The linking process links intermediate object and library files generated during compilation and assembly, checks for unresolved labels, and then creates a *.hex* and a *.lst* file, as well as assorted *.o* and *.dbg* files. For descriptions of these files, refer to “Source Files Generated by Generate Project Operation” on page 57.

To set linker options in PSoC Designer, select Project > Settings... > Build > Linker. This screen configures the linker specific options based on the compiler selection made in the Compiler screen. The Selected C compiler box indicates which compiler (and linker) is currently selected.

5.3.1 ImageCraft Specific Linker Options

Configuration options of imagecraft specific linker are as follows:

- **Relocatable code start address** specifies the first Flash address for the linker to start placing relocatable code areas. The relocatable code start address can be entered in only in hexadecimal and is displayed in 0x hexadecimal.
- **Object/library modules** specifies a list of libraries to link in addition to the default library.
- **Additional library path** specifies a library path alternative to the default.

Refer to the ImageCraft C Compiler Guide for more information.

5.3.2 HI-TECH Specific Linker Options

Configuration options of Hi-tech specific linker are as follows:

- **Warning Level** specifies the minimum warning message level allowed for output.
- **Options** allows you to enter any command line linker options

Refer to the HI-TECH C(R) PRO for the PSoC(R) Mixed-Signal Array guide for more information.

5.3.3 Customizing Linker Actions

To customize the actions of the Linker, create a file called *custom.lkp* in the root folder of the project (see the *C Language Compiler User Guide - Command Line Overview*).

Be aware that in some cases, creating a text file and renaming it preserves the *.txt* file extension (e.g., *custom.lkp.txt*). If this occurs, you cannot use custom commands. The `make` reads the contents of *custom.lkp* and appends these commands to the Linker action.

A typical use for the *custom.lkp* capability is to define a custom relocatable code AREA. For example, to create code in a separate code AREA that should be located in the upper 2K of the Flash, use this feature. For this example, the custom code AREA is called ‘BootLoader’. If you were developing code in C for the BootLoader AREA, use this pragma in your C source file:

```
#pragma text:BootLoader// switch the code below from
// AREA text to BootLoader
// ... Add your Code ...
#pragma text:text // switch back to the text
// AREA
```

If you develop code in assembly, use the AREA directive in this manner:

```
AREA BootLoader(rom,rel)
; ... Add your Code ...
AREA text ; reset the code AREA
```

Now that you have code that should be located in the BootLoader AREA, you can add your custom Linker commands to *custom.lkp*. For this example, type this line in the *custom.lkp* file:

```
-bBootLoader:0x3800.0x3FFF
```

You can verify that your custom Linker settings were used by checking the Use verbose build messages field in the **Tools** > **Options**... > **Build** tab. Build the project, then view the Linker settings in the Build tab of the Output Status window (or check the location of the BootLoader AREA in the .mp file).

5.4 Librarian

The library and archiving features of PSoC Designer provide system storage and reference.

There are two types of Librarian files (located in the source tree): Library Source and Library Headers. Source file types include archived and assembly language such as *libpsoc.a* and *PSoCConfig.asm*. Header files are intermediate reference and include files created during application code generation and compilation. Both types are generated and used by PSoC Designer and are unique to each specific project.

6. Debugger

In this chapter you learn how to download your project to an in-circuit emulator (ICE), use debug strategies, and program the part.

PSoC Designer supports two different methods to provide in-circuit emulation. The first uses an external emulator called an ICE cube. The ICE cube has the ability to emulate different PSoC parts in circuit using different foot kits or adapters to emulate the device in circuit. Figure 6-1 shows the components in the ICE Cube Development Kit.

Some PSoC devices do not use an external emulator (ICE cube) for debugging. Instead, these devices support a subset of the debugging options on chip. These devices use I^{2}C to debug on-chip through an ISSP header and MiniProg3. This section documents both emulation methods and how to use them with the PSoC Designer debugger. Figure 6-2 on page 94 shows the MiniProg3 used with devices with on-chip emulators.

The following device familites have on chip emulators

- CY8CTMA3xx

Figure 6-1. Components of the ICE Cube Development Kit

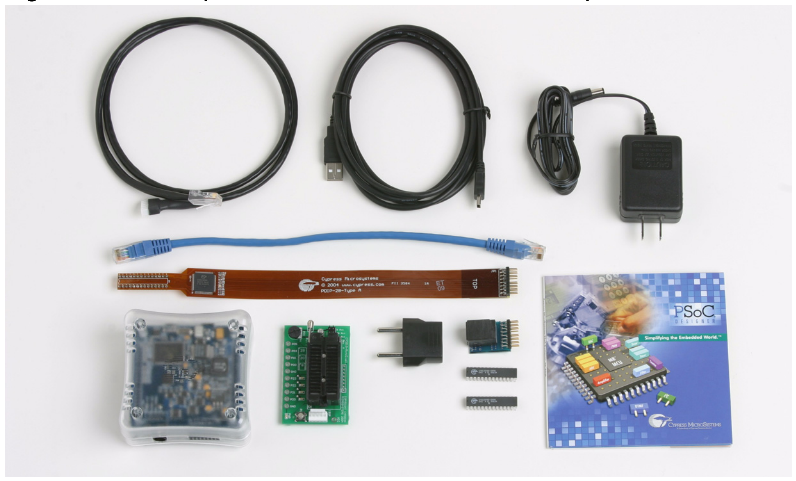
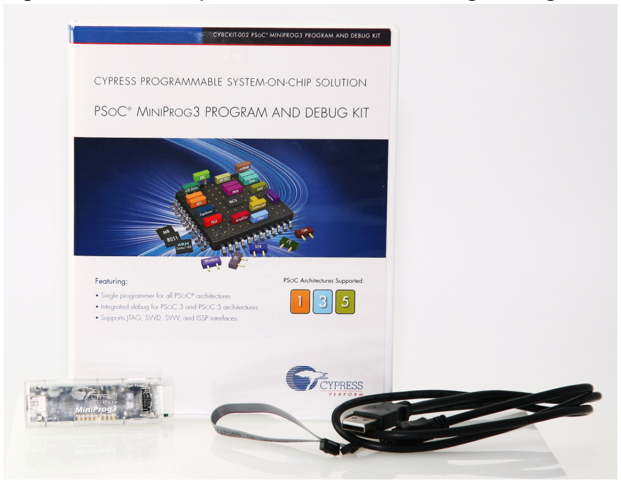


Figure 6-2. Components of the MiniProg3 Program and Debug Kit

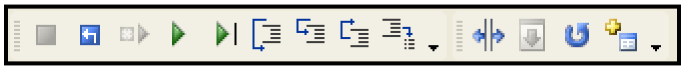

CY8CKIT-002 PSoC® MINIPROG3 PROGRAM AND DEBUG KIT
CYPRESS PROGRAMMABLE SYSTEM-ON-CHIP SOLUTION
PSoC® MINIPROG3 PROGRAM AND DEBUG KIT
Featuring:
PSoC Architectures Supported:
1
3
5
CYPRESS
MiniProg3

6.1 Online Training

For complete training on debugging and Dynamic Event Points, try PSoC Designer Module 3:Debugging with PSoC. Review and sign up under **Training** > **On-Demand** at http://www.cypress.com/.

6.2 Menu Options

The Debug and Ice tool bars incorporate the most important Debugger functions.



A listing of all Debugger menu options is available in Table 6-1.

The I^{2}C debugger does not use an external emulator and does not support the following:

- Events Window
- Trace Window
- Trace Mode

All other debug functions are fully supported.

Table 6-1. Debugging Menu Options

Icon	Menu/Tool Tip	Shortcut	Feature
⇹	Connect	[**F9**]	Connects PSoC Designer to ICE
⬇	Download to emulator	**[Shift + F9]**	Downloads project .hex file to hardware emulator (Pod). This file holds all device configurations and source-code functionality
▶	Execute Program	[**F4**]	Switches into Debugging subsystem, connects, downloads .hex, runs... all from one click
▶	Go	[**F5**]	Starts debugger
▶|	Run To Cursor	[**Ctrl**] [**F5**]	Creates a temporary (invisible) breakpoint at the current cursor location in the source code and runs the application to that point.

Table 6-1. Debugging Menu Options<Italic> (continued)

	Halt	[**Shift**][**F5**]	Stops debugger
	Reset	[**Ctrl**] [**Shift**] [**F5**]	Resets the device to a PC value of ‘0’ and restarts the debugger
	Step Into[a]	[**F11**]	Steps into next statement
	Step Out	[**Shift**] [**F11**]	Steps out of current function
	Step Over[a]	[**F8**]	Steps over next statement
	Step ASM	[**Shift**][**F10**]	If the current line of code is C code, the line is located in the.lst file and that line is executed.
	Refresh M8C Views		Refreshes the data in the Memory, CPU Registers, and Watch Variables debugger windows with current data from the emulator.
	Get Next Trace Data		When the debugger halts, the trace data window is loaded with the latest 64 lines of trace data. This button retrieves an additional 64 lines.
	Get all Trace Data		When the debugger halts, the trace data window is loaded with all the trace data. This button retrieves all the trace data.
	Configure USB Data Tracking		If the device supports USB user modules, this button will be used for configuring USB end points.

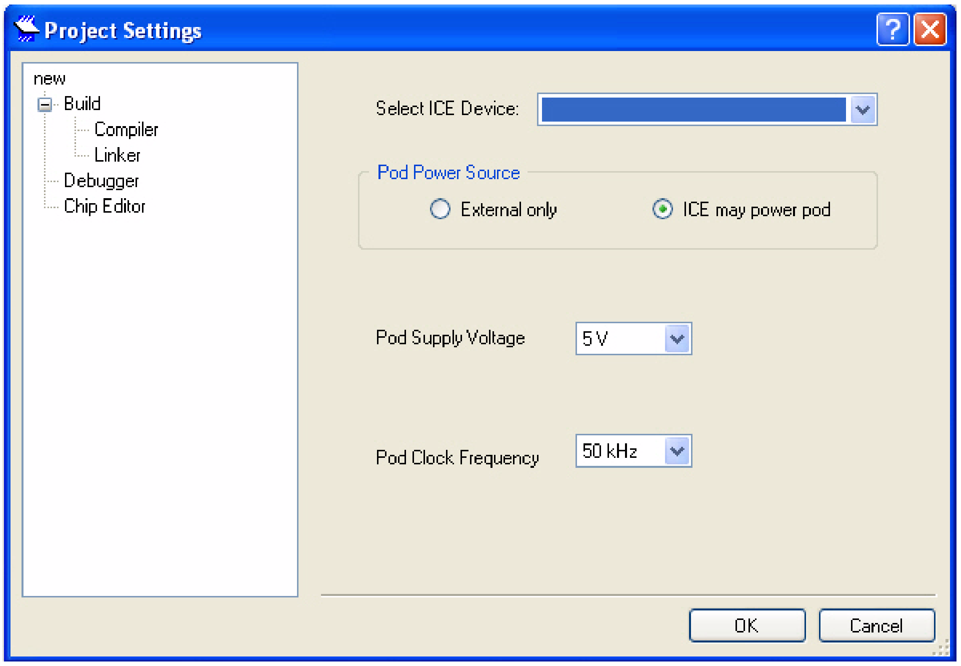
a. If C source lines are compiled into assembly code that branches the execution path (such as `lcall` or `call` instructions), the Debugger attempts to step into the source file of the destination address. For library code such as multiplication and division, a call to the library assembly code is made but the original source file is not available in the project. Step Into then gives the message “No source available for step operation.” Step Over can be used instead of step into for this situation.

6.3 Debugging With an External Emulator

6.3.1 Connecting to the ICE

You must establish a communication link between the PC and the ICE. This is done by choosing the appropriate ICE device. To make the Debugger port selection select the Project > Settings... > Debugger tab.

Figure 6-3. Debugger Project Settings for an ICE Cube



Select ICE Device:

Pod Power Source: External only [✓ ICE may power pod]

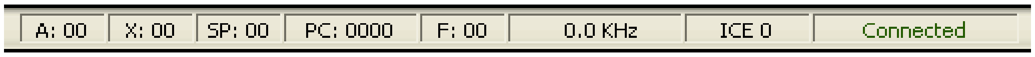
Pod Supply Voltage: 5 V

Pod Clock Frequency: 50 kHz

The Select ICE Device list shows USB connections supporting the ICE cube or MiniProg3. The ICE cube can support up to 100 mA at 5V, or 3.3V or you can choose to supply the power externally.

After you have set your device, check the PC to ICE communication link by using the Connect button or by selecting the Debug > Connect/Disconnect menu item. The results of the connection attempt are displayed in the status bar shown in Figure 6-4.

Figure 6-4. Debug Status Bar.



```
A: 00 | X: 00 | SP: 00 | PC: 0000 | F: 00 | 0.0 KHz | ICE 0 | Connected
```

6.3.2 Downloading to the Pod

Before you begin a debug session you need to download your project *.hex* file to the pod. By doing this, you load the ROM addressing data into the emulation bondout device (chip on the pod). A general rule to follow before downloading is to make sure there is not a part in the programming socket of the Pod. Otherwise, debug sessions may fail.

To download the .hex file to the Pod:

1. Click the **Download to Emulator** (Pod) icon .

The system downloads the project *.hex* file located in the …`\output` folder of your project directory. A progress indicator reports download status.

2. After the download is complete, the pod can be directly connected to and debugged on your specific circuit board.

If you cannot debug your project with the current pod, you receive this message:

```
"Attached pod is not compatible with the selected PSoC"
```

This appears in the Debug tab of the Output Status window:

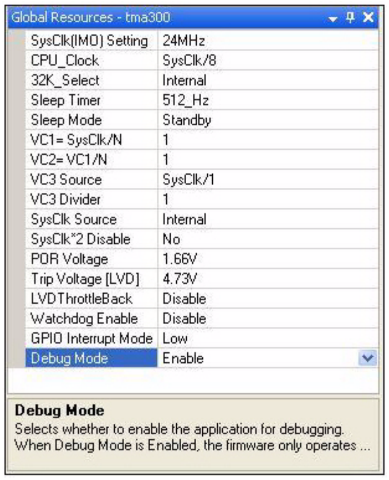
"Fail to Connect"

6.4 Debugging With an On-Chip Emulator

6.4.1 Enable Debug Mode

Before you generate your project, Debug Mode must be enabled in the Global Resources window. During code generation, extra code necessary for debugging will be added to the hex file automatically. A hex file generated with Debug Mode enabled will not run on the PSoC device unless it is connected to the Debugger. Before generating production code, disable debug mode.

Figure 6-5. Debug Mode Enabled



Global Resources - tma300

SysClk(IMO) Setting	24MHz
CPU_Clock	SysClk/8
32K_Select	Internal
Sleep Timer	512_Hz
Sleep Mode	Standby
VC1= SysClk/N	1
VC2= VC1/N	1
VC3 Source	SysClk/1
VC3 Divider	1
SysClk Source	Internal
SysClk*2 Disable	No
POR Voltage	1.66V
Trip Voltage [LVD]	4.73V
LVDThrottleBack	Disable
Watchdog Enable	Disable
GPIO Interrupt Mode	Low
Debug Mode	Enable

Debug Mode
Selects whether to enable the application for debugging.
When Debug Mode is Enabled, the firmware only operates ...

6.4.2 Connecting to the MiniProg

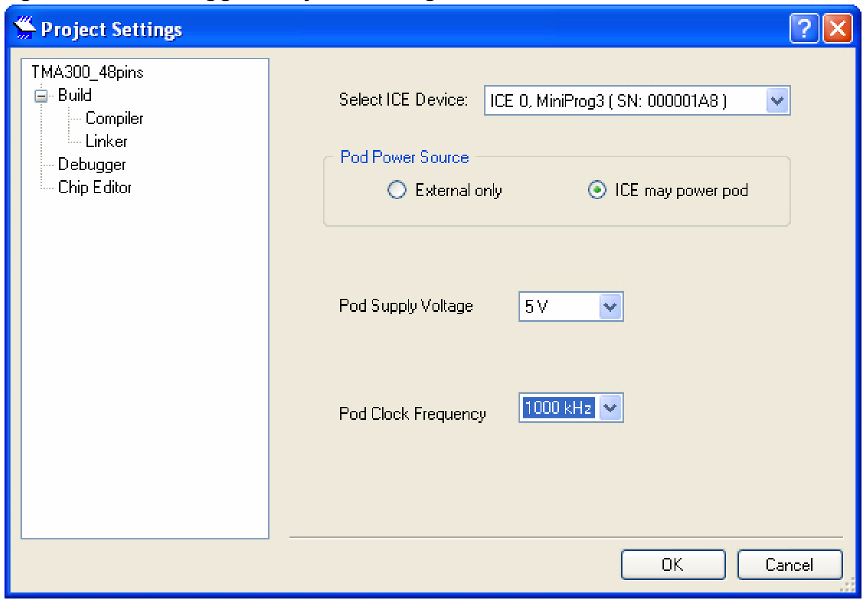
An important setting for each PSoC Designer project is the ICE Device that will be used for debugging. For I^{2}C debugging you will need to select the MiniProg3 from the list of ICE Devices in the **Project** > **Settings...** > **Debugger** tab. The ICE Device list shows only hardware that can be used for the project that is currently opened.

MiniProg3 can support up to 100 mA in 5 V, 3.3 V, 2.5 V, 1.8 V, or you can choose to supply the power externally.

Pod Clock Frequency is only effective in debug mode. It supports 50 kHz, 100 kHz, 400 kHz, 750 kHz, 1 MHz and 1.5 MHz. By default, it is set to 100 kHz. Default value for Pod Clock Frequency: 50 kHz. Default value for Pod Supply Voltage: 5 V

If the Auto option is selected for the Pod Clock Frequency, MiniProg3 will search for the fastest frequency that the target device can handle. This option is recommended. The fastest operable setting will depend on the project SysClk setting and on bus capacitance. Refer to the applicable PSoC device data sheet for guidance.

Figure 6-6. Debugger Project Settings



Select ICE Device: ICE 0, MiniProg3 (SN: 000001A8)

Pod Power Source: ( ) External only [✓ ICE may power pod]

Pod Supply Voltage: 5 V

Pod Clock Frequency: 1000 kHz

After you have set your ICE device, and are connected, check the communication link by using the Connect button or by selecting the Debug > Connect/Disconnect menu item. The results of the connection attempt are displayed in the Output window and the status bar.

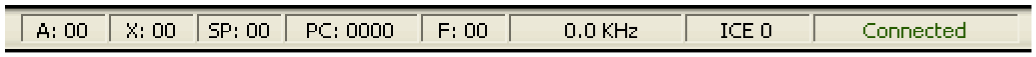
INFO - 2010-07-26 17:27:10,140

Connecting . . .

INFO - 2010-07-26 17:27:16,203

Connected."

Figure 6-7. Debug Status Bar



```
A: 00 | X: 00 | SP: 00 | PC: 0000 | F: 00 | 0.0 KHz | ICE 0 | Connected
```

6.4.3 Downloading to the Device

Before you begin a debug session you need to download your project *.hex* file to the device.

To download the .hex file to the device:

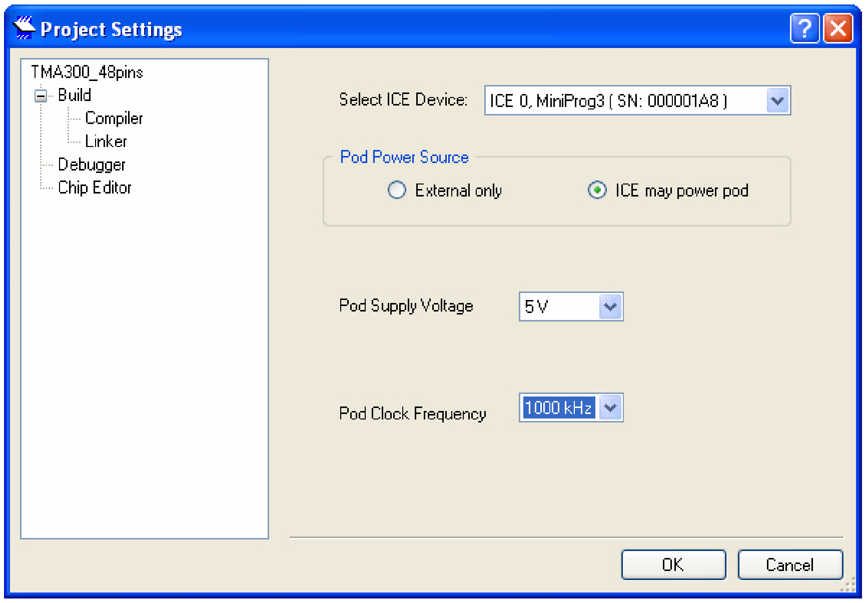
1. Click the **Download to emulator...** icon .

   The system downloads the project *.hex* file located in the ...`\output` folder of your project directory. A progress indicator reports download status.
2. After the download is complete, the device will run normally and report debug information over I^2C to the attached MiniProg3.

6.4.4 I2C Debugger

Connecting to the ICE : An important setting for each PSoC Designer project is the ICE Device is used for debugging. For I2C debugging you need to select the MiniProg3 from the list of ICE Devices in the **Project** > **Settings...** > **Debugger** tab. The ICE Device list shows only hardware that can be used for the project that is currently opened.

MiniProg3 can support up to 100mA in 5 V, 3.3 V, 2.5 V, 1.8 V, or you can choose to supply the power externally. Pod Clock Frequency is only effective in debug mode. It supports 50 kHz, 100 kHz, 400 kHz, 750 kHz, 1 MHz and 1.5 MHz. By default, it is set to 100 kHz. Choose the option that best fits the target hardware. The fastest operable setting depends on the project SysClk setting and on bus capacitance. Refer to the applicable PSoC device data sheet for guidance.



Select ICE Device: ICE 0, MiniProg3 (SN: 000001A8)

Pod Power Source: ( ) External only [✓ ICE may power pod]

Pod Supply Voltage: 5 V

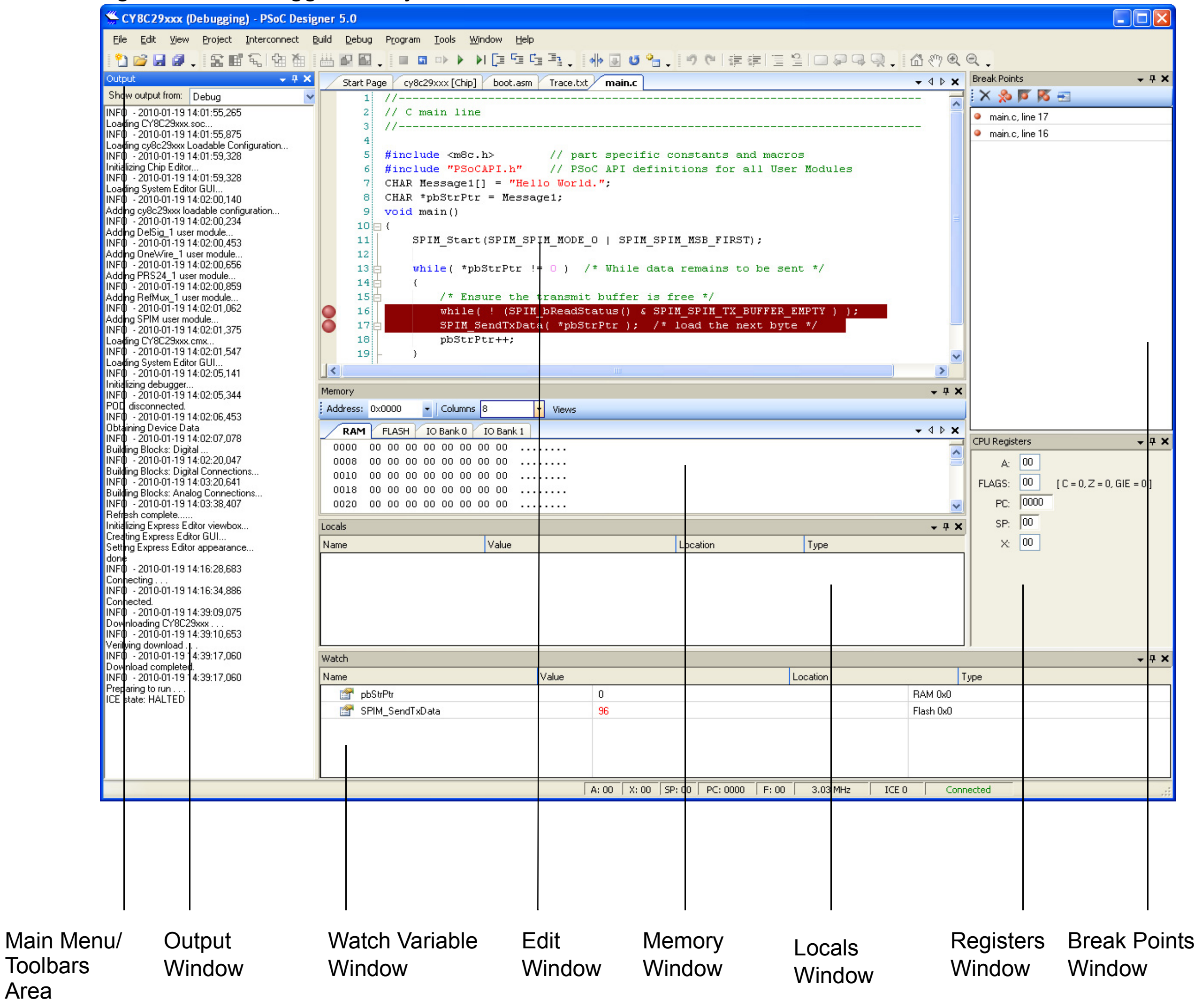
Pod Clock Frequency: 1000 kHz

After you have set your ICE device, and are connected, check the communication link by using the Connect button or by selecting the Debug > Connect/Disconnect menu item. The results of the connection attempt are displayed in the Output window and the status bar.

6.5 Debug Strategies

Debugger commands allow you to read and write program and data memory, read and write IO registers, read and write CPU registers and RAM, set and clear breakpoints, and provide program run, halt, and step control.

Figure 6-8. Debugger Subsystem View


CY8C29xxx (Debugging) - PSoC Designer 5.0
File Edit View Project Interconnect Build Debug Program Tools Window Help
Output
Show output from: Debug
Start Page cy8c29xxx [Chip] boot.asm Trace.txt main.c
Break Points
main.c, line 17
main.c, line 16
Memory
Address: 0x0000 Columns 8 Views
RAM FLASH IO Bank 0 IO Bank 1
CPU Registers
A: 00
FLAGS: 00 [C = 0, Z = 0, GIE = 0]
PC: 0000
SP: 00
X: 00
Locals
Name Value Location Type
Watch
Name Value Location Type
A: 00 X: 00 SP: 00 PC: 0000 F: 00 3.03 MHz ICE 0 Connected
Main Menu/ Toolbars Area
Output Window
Watch Variable Window
Edit Window
Memory Window
Locals Window
Registers Window
Break Points Window

```
INFO  - 2010-01-19 14:01:55,265
Loading CY8C29xxx.soc...
INFO  - 2010-01-19 14:01:55,875
Loading cy8c29xxx Loadable Configuration...
INFO  - 2010-01-19 14:01:59,328
Initializing Chip Editor...
INFO  - 2010-01-19 14:01:59,328
Loading System Editor GUI...
INFO  - 2010-01-19 14:02:00,140
Adding cy8c29xxx loadable configuration...
INFO  - 2010-01-19 14:02:00,234
Adding DelSig_1 user module...
INFO  - 2010-01-19 14:02:00,453
Adding OneWire_1 user module...
INFO  - 2010-01-19 14:02:00,656
Adding PRS24_1 user module...
INFO  - 2010-01-19 14:02:00,859
Adding RefMux_1 user module...
INFO  - 2010-01-19 14:02:01,062
Adding SPIM user module...
INFO  - 2010-01-19 14:02:01,375
Loading CY8C29xxx.cmx...
INFO  - 2010-01-19 14:02:01,547
Loading System Editor GUI...
INFO  - 2010-01-19 14:02:05,141
Initializing debugger...
INFO  - 2010-01-19 14:02:05,344
POD disconnected.
INFO  - 2010-01-19 14:02:06,453
Obtaining Device Data
INFO  - 2010-01-19 14:02:07,078
Building Blocks: Digital ...
INFO  - 2010-01-19 14:02:20,047
Building Blocks: Digital Connections...
INFO  - 2010-01-19 14:03:20,641
Building Blocks: Analog Connections...
INFO  - 2010-01-19 14:03:38,407
Refresh complete......
Initializing Express Editor viewbox...
Creating Express Editor GUI...
Setting Express Editor appearance...
done
INFO  - 2010-01-19 14:16:28,683
Connecting . . .
INFO  - 2010-01-19 14:16:34,886
Connected.
INFO  - 2010-01-19 14:39:09,075
Downloading CY8C29xxx . . .
INFO  - 2010-01-19 14:39:10,653
Verifying download . . .
INFO  - 2010-01-19 14:39:17,060
Download completed.
INFO  - 2010-01-19 14:39:17,060
Preparing to run . . .
ICE state: HALTED
```

```c
//----------------------------------------------------------------------------
// C main line
//----------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
CHAR Message1[] = "Hello World.";
CHAR *pbStrPtr = Message1;
void main()
{
    SPIM_Start(SPIM_SPIM_MODE_0 | SPIM_SPIM_MSB_FIRST);

    while( *pbStrPtr != 0 )  /* While data remains to be sent */
    {
        /* Ensure the transmit buffer is free */
        while( ! (SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY ) );
        SPIM_SendTxData( *pbStrPtr );  /* load the next byte */
        pbStrPtr++;
    }
```

```
0000  00 00 00 00 00 00 00 00  ........
0008  00 00 00 00 00 00 00 00  ........
0010  00 00 00 00 00 00 00 00  ........
0018  00 00 00 00 00 00 00 00  ........
0020  00 00 00 00 00 00 00 00  ........
```

Name	Value	Location	Type
pbStrPtr	0	RAM 0x0	
SPIM_SendTxData	96	Flash 0x0	

In the status bar of the Debugger subsystem you find ICE connection indication, debugger target state information, and **A**ccumulator, **X**, **S**tack **P**ointer, **P**rogram **C**ounter, and **F**lag register values.

To help with troubleshooting, you can view your application source files inside the Debugger subsystem. If the project source tree is not showing, click View > Workspace Explorer.

The project files that are viewed in the debugger are not read-only while the debugger is halted at a breakpoint. The source files are editable when the debugger is reset.

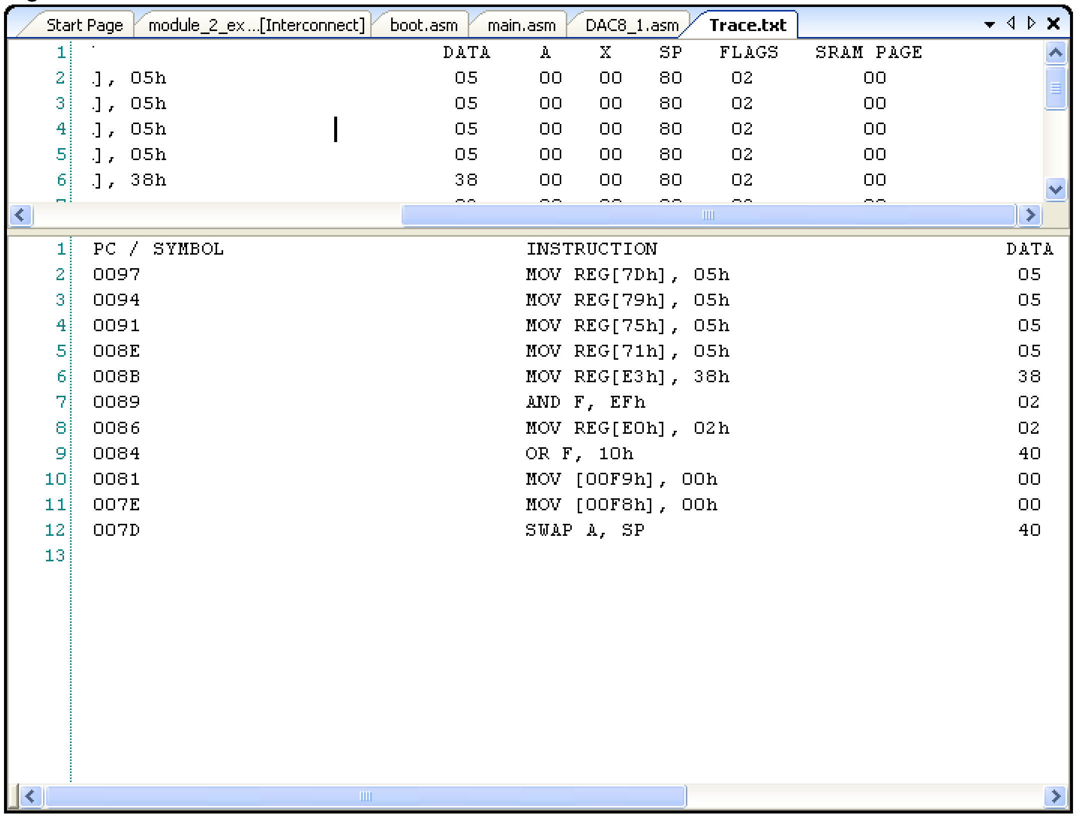
6.5.1 Trace

The trace feature is not available with the I^{2}C debugger. The trace feature enables you to track and log device activity at either a high or detailed level. Such activity includes register values, data memory, and time stamps.

The Trace window is displayed when **Debug** > **Windows** > **Trace** is chosen.

The Trace window displays a continuous, configurable listing of project symbols and operations from the last breakpoint. (The trace shows symbolic, rather than address data, to enhance readability.) Each time program execution starts, the trace buffer is cleared. When the trace buffer becomes full, it continues to operate and overwrite old data.

Figure 6-9. Trace Window



```
`                                 DATA    A     X     SP    FLAGS    SRAM PAGE
], 05h                            05      00    00    80     02          00
], 05h                            05      00    00    80     02          00
], 05h                            05      00    00    80     02          00
], 05h                            05      00    00    80     02          00
], 38h                            38      00    00    80     02          00
```

```
PC / SYMBOL                          INSTRUCTION                      DATA
0097                                 MOV REG[7Dh], 05h                 05
0094                                 MOV REG[79h], 05h                 05
0091                                 MOV REG[75h], 05h                 05
008E                                 MOV REG[71h], 05h                 05
008B                                 MOV REG[E3h], 38h                 38
0089                                 AND F, EFh                        02
0086                                 MOV REG[E0h], 02h                 02
0084                                 OR F, 10h                         40
0081                                 MOV [00F9h], 00h                  00
007E                                 MOV [00F8h], 00h                  00
007D                                 SWAP A, SP                        40
```

Configure the Trace window by selecting either **Debug** > **Trace Mode** from the menu. Configuration options include:

PC Only – Lists the PC value and instruction only.

PC/Registers – Lists the PC, instruction, data, A register, X register, SP register, F register, and ICE external input.

PC/Timestamp – Lists the PC, instruction, A register, ICE external input, and time stamp.

The Trace.txt file is automatically updated and saved in the “...\output” folder of your project directory by PSoC Designer IDE regardless of user actions **File** > **Save Trace.txt As**...

Trace.txt file is updated and saved automatically by PSoC Designer IDE regardless of user actions.

The trace log entries are logged after the instruction is executed. The contents of those entries are:

- PC Register

- A Register
- Data Bus
- External Signals

The default size of trace is 256 kilobytes. This provides 128K trace instructions in trace mode 1 and 32K trace instructions in trace modes 2 and 3.

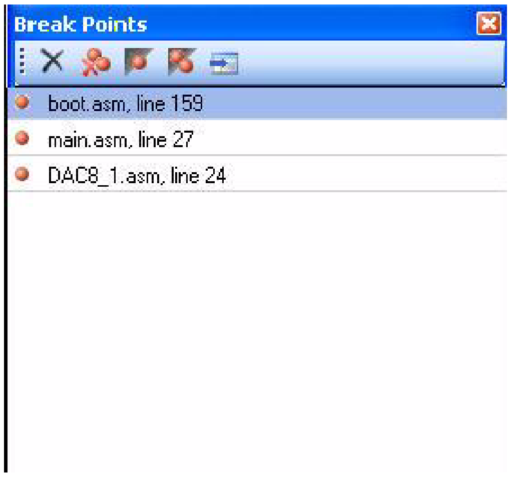
6.5.2 Break Points

The Break Point feature allows you to stop program execution at predetermined address locations. When a break point is encountered, the program stops at the address of the break point, without executing the address code. After halted, the program is restarted using the available menu or icon options.

To set break points, first open the file to debug. Do this from the Workspace Explorer. (If your project file Workspace Explorer is currently not showing, click View > Workspace Explorer.) Break points are created by right clicking your mouse at targeted points and selecting Insert Break Point. You can view and remove active break points in the Break Points window. To open the Break Points window, select Debug > Windows > Break Points.

The I^{2}C Debugger doesn't use the external emulator and has a limited number of break points. Active Break points will be shown with solid icon as shown in Figure 6-10:

Figure 6-10. Breakpoints Window



boot.asm, line 159
main.asm, line 27
DAC8_1.asm, line 24

You can also view the exact line and column for each break point (or wherever you click your cursor in the file) across the bottom of PSoC Designer.

6.5.3 CPU and Register Views

There are five areas that are readable and writable during debugging: CPU Registers, Bank Registers 0, Bank Registers 1, RAM, and Flash. The CPU Registers are shown in their own window (Debug > Windows > Registers) and in the notification area at the bottom of PSoC Designer. The other four areas can be viewed in the Memory Window (Debug > Windows > Memory). Select one of the four memory areas from the **Address Space** box. Each is described below.

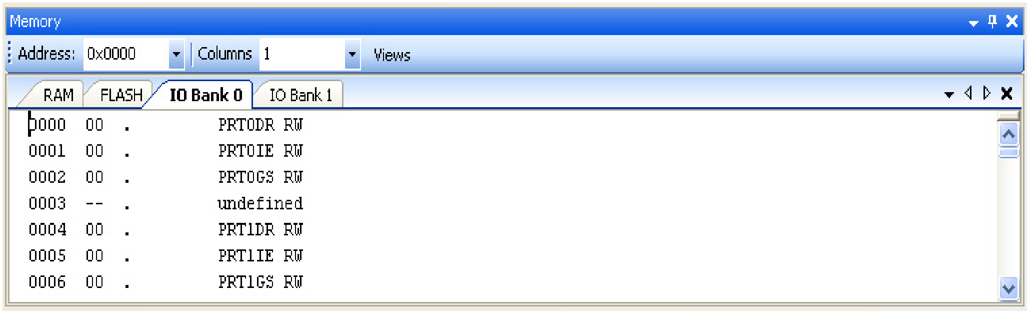
NOTE: Use caution when changing register values; they can alter hardware functions.

CPU Registers – This window allows you to examine and change the contents of the CPU registers. Data is entered in hexadecimal notation. CPU register values can be viewed across the bottom of PSoC Designer.

Bank Registers 0 and 1 – You can scroll through the register bank to view the values in the register bank. Type a new value into the **Offset** to scroll directly to that offset. Click next to a value and type a new value for the register. All values must be entered in hexadecimal notation.

Note that you cannot change some registers because they are read only. Some registers are write only and cannot be read.

Figure 6-11. Bank 0 Registers in the Memory Window



```
0000  00  .        PRT0DR RW
0001  00  .        PRT0IE RW
0002  00  .        PRT0GS RW
0003  --  .        undefined
0004  00  .        PRT1DR RW
0005  00  .        PRT1IE RW
0006  00  .        PRT1GS RW
```

RAM – View a RAM memory page. RAM locations can be modified by clicking the data at the specific location and typing in the new value. Data is entered in hexadecimal notation.

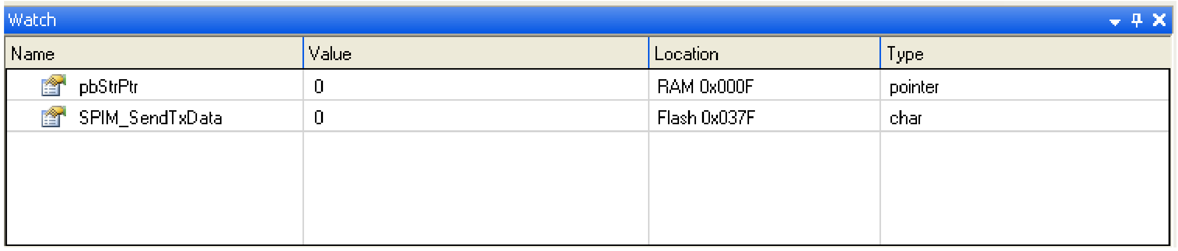
Flash – The Flash window displays the data stored in Flash. This is the program memory; it is read only.

6.5.4 Watch Variables

Watch Variables can be set by right clicking a variable in a source file and selecting Add Watch. You can also select Global Variables.

Right-click inside the Watch Variables, the context menu appears as Define arbitrary watch, Delete, Delete all watch variables, Edit, Hexadecimal. The properties in the Watch window to add, delete, or modify values. Note that if you change a variable type (or other settings in the window) and close the project, the next time you access that project the variable types and settings are the same.

Figure 6-12. Watch Variables Window

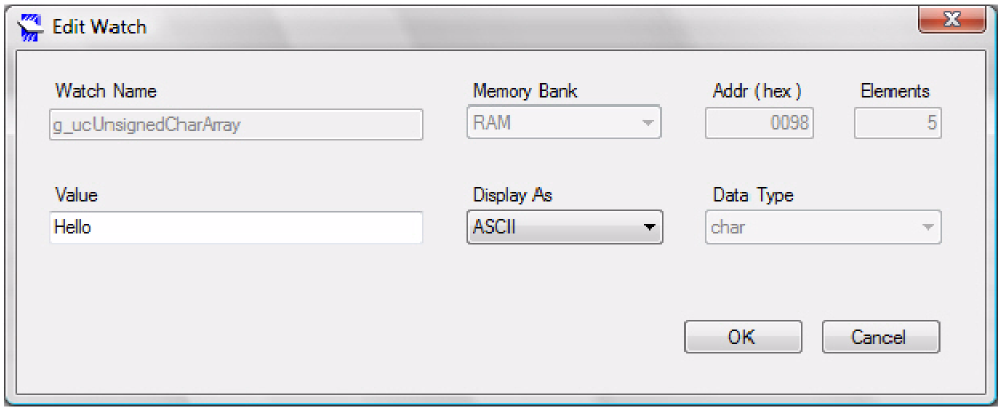


Name	Value	Location	Type
pbStrPtr	0	RAM 0x000F	pointer
SPIM_SendTxData	0	Flash 0x037F	char

The WatchWindow subsystem distinguishes between watch variables that were declared in the project source code and watch variables that were created through the Watch Window pop-up menu, **Define arbitrary watch**.

The dialog box for editing watch variables (and creating new arbitrary ones) is shown in Figure 6-13

Figure 6-13. Edit Watch Dialog



Watch Name: g_ucUnsignedCharArray
Memory Bank: RAM
Addr (hex): 0098
Elements: 5
Value: Hello
Display As: ASCII
Data Type: char

6.5.4.1 *Display Format*

The 'Display As' property controls how the value will be displayed when the Watch Window is not set to display its variables in hexadecimal format. There are 5 possible settings for the display format.

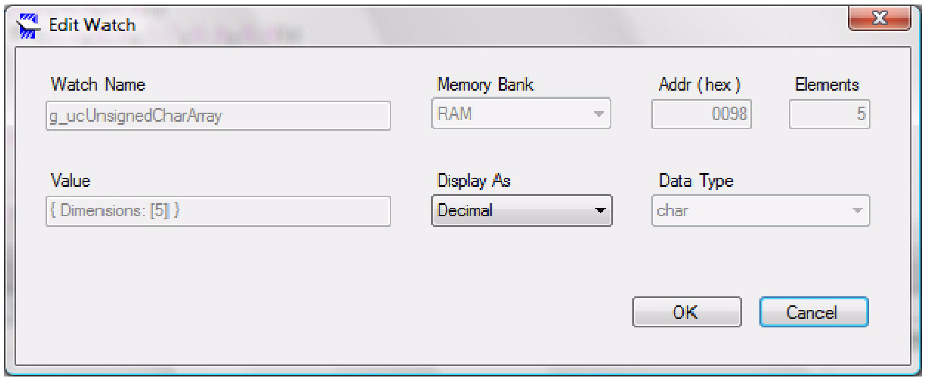
Table 6-2. Data Display versus Format

Format	Types of Data that Can be Displayed
Decimal	All Data Types
Hexadecimal	All Data Types
Binary	All Data Types
ASCII	One-byte data types - char and unsigned char
Unicode	Two-byte data types - int, unsigned int, short, unsigned short, pointer

Usually, the contents of the watch variable's memory location are displayed and modifiable in the **Value** field. There are two conditions when the data is not displayed in the **Value** field:

1. When the Data Type of the watch variable is 'struct' the memory contents will not be displayed in the 'Value' field.
2. When the 'Elements' field is greater than 1 and the 'Display As' value is not ASCII (non-character array) the memory contents will not be displayed in the 'Value' field.

Figure 6-14. A char Array Not Displayed as ASCII



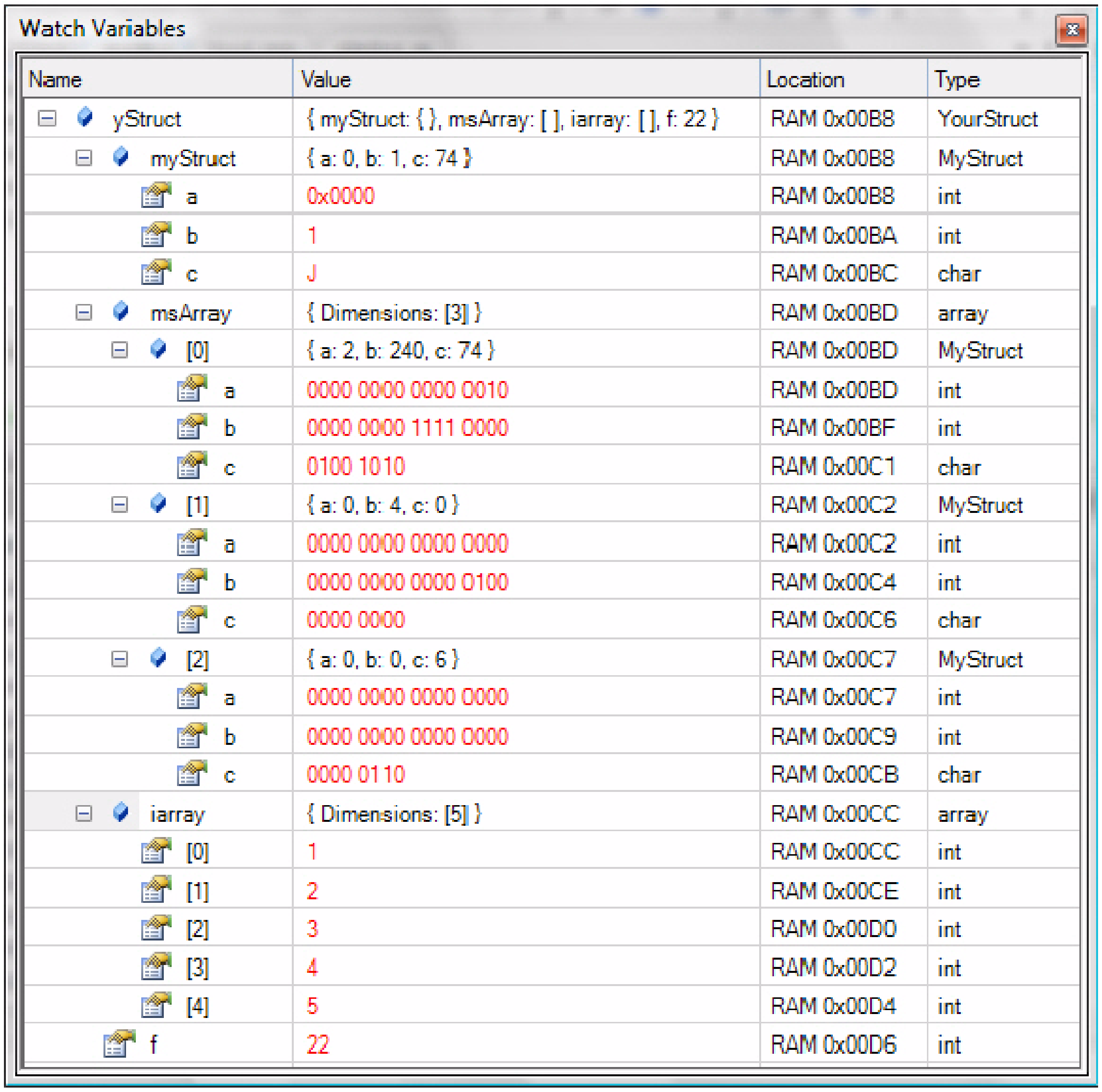
Watch Name: g_ucUnsignedCharArray
Memory Bank: RAM
Addr (hex): 0098
Elements: 5
Value: { Dimensions: [5] }
Display As: Decimal
Data Type: char

When the **Display As** property is changed, the value from memory is displayed in the **Value** field. If you changed the Value and did not click OK before changing the Display As value, the data change is lost.

When a display format is applied to an array or a struct watch variable, all the elements (or fields) of the variable are displayed in the selected format.

To display a single element or field of an array or struct, select just that element from the Watch Window and then edit the format of just that element.

Figure 6-15. Items Displayed in their Configured Data Formats



Watch Variables

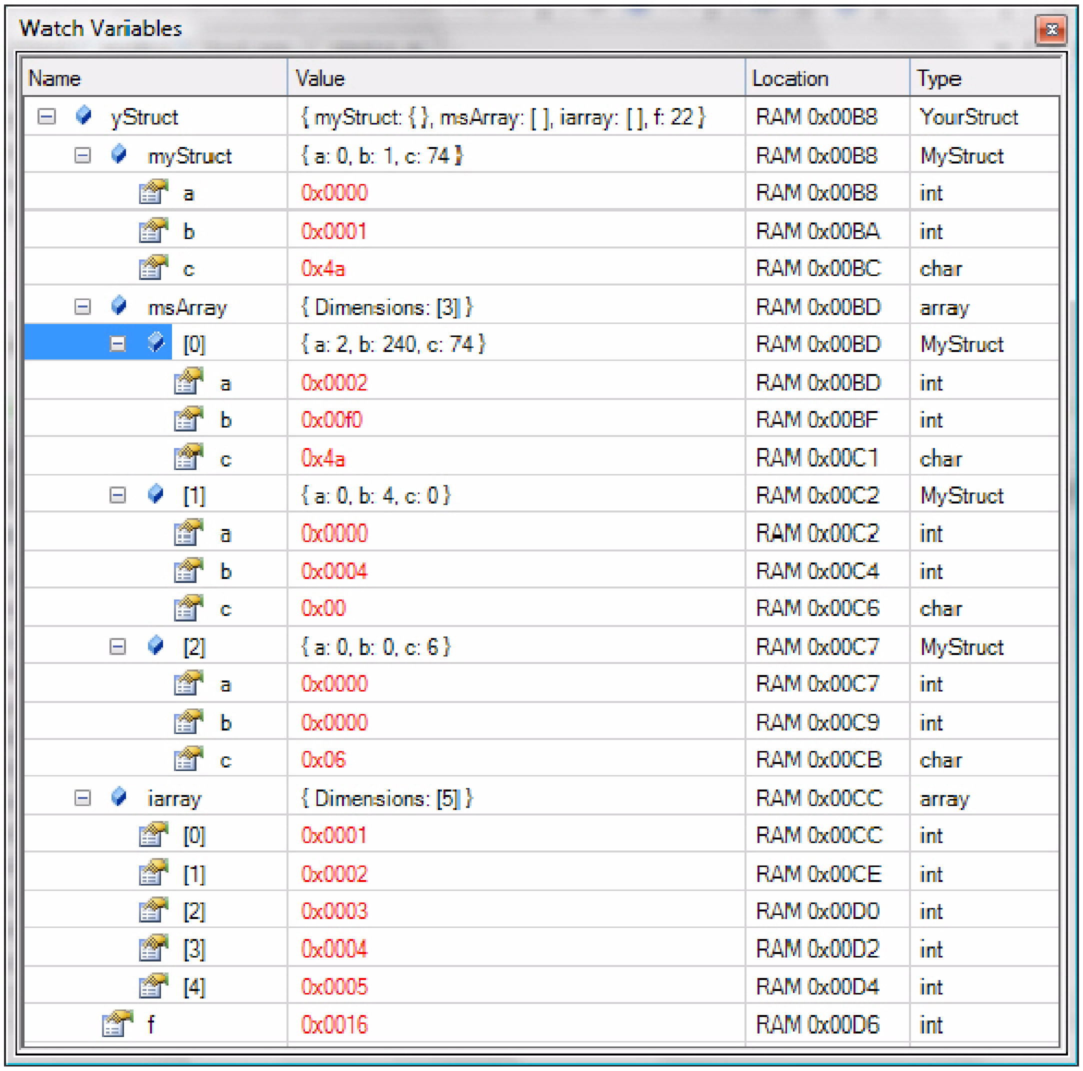
Name	Value	Location	Type
yStruct	{ myStruct: { }, msArray: [ ], iarray: [ ], f: 22 }	RAM 0x00B8	YourStruct
myStruct	{ a: 0, b: 1, c: 74 }	RAM 0x00B8	MyStruct
a	0x0000	RAM 0x00B8	int
b	1	RAM 0x00BA	int
c	J	RAM 0x00BC	char
msArray	{ Dimensions: [3] }	RAM 0x00BD	array
[0]	{ a: 2, b: 240, c: 74 }	RAM 0x00BD	MyStruct
a	0000 0000 0000 0010	RAM 0x00BD	int
b	0000 0000 1111 0000	RAM 0x00BF	int
c	0100 1010	RAM 0x00C1	char
[1]	{ a: 0, b: 4, c: 0 }	RAM 0x00C2	MyStruct
a	0000 0000 0000 0000	RAM 0x00C2	int
b	0000 0000 0000 0100	RAM 0x00C4	int
c	0000 0000	RAM 0x00C6	char
[2]	{ a: 0, b: 0, c: 6 }	RAM 0x00C7	MyStruct
a	0000 0000 0000 0000	RAM 0x00C7	int
b	0000 0000 0000 0000	RAM 0x00C9	int
c	0000 0110	RAM 0x00CB	char
iarray	{ Dimensions: [5] }	RAM 0x00CC	array
[0]	1	RAM 0x00CC	int
[1]	2	RAM 0x00CE	int
[2]	3	RAM 0x00D0	int
[3]	4	RAM 0x00D2	int
[4]	5	RAM 0x00D4	int
f	22	RAM 0x00D6	int

The image of the Watch Window shown in Figure 6-16 shows a single watch variable named `yStruct`, of type `YourStruct`. It contains 4 fields named `myStruct`, `msArray`, `iArray` and `f`. The display format of the `msArray` field is set to Binary. Every data element contained in `msArray` is displayed in binary format. The format of `iArray` and `f` are the default of Decimal.

The display format of each field of `yStruct.myStruct` was set individually. The display format of `yStruct.myStruct.a` is Hexadecimal. The format of `yStruct.myStruct.b` is Decimal, and the format of `yStruct.myStruct.c` is set to ASCII.

Checking the **Hexadecimal** item in the Watch Window's pop-up menu toggles all data in the window to hexadecimal format. When **Hexadecimal** is un-checked, the data formats return to their original display format.

Figure 6-16. Items Displayed in Hexadecimal Format



Watch Variables

Name	Value	Location	Type
yStruct	{ myStruct: { }, msArray: [ ], iarray: [ ], f: 22 }	RAM 0x00B8	YourStruct
myStruct	{ a: 0, b: 1, c: 74 }	RAM 0x00B8	MyStruct
a	0x0000	RAM 0x00B8	int
b	0x0001	RAM 0x00BA	int
c	0x4a	RAM 0x00BC	char
msArray	{ Dimensions: [3] }	RAM 0x00BD	array
[0]	{ a: 2, b: 240, c: 74 }	RAM 0x00BD	MyStruct
a	0x0002	RAM 0x00BD	int
b	0x00f0	RAM 0x00BF	int
c	0x4a	RAM 0x00C1	char
[1]	{ a: 0, b: 4, c: 0 }	RAM 0x00C2	MyStruct
a	0x0000	RAM 0x00C2	int
b	0x0004	RAM 0x00C4	int
c	0x00	RAM 0x00C6	char
[2]	{ a: 0, b: 0, c: 6 }	RAM 0x00C7	MyStruct
a	0x0000	RAM 0x00C7	int
b	0x0000	RAM 0x00C9	int
c	0x06	RAM 0x00CB	char
iarray	{ Dimensions: [5] }	RAM 0x00CC	array
[0]	0x0001	RAM 0x00CC	int
[1]	0x0002	RAM 0x00CE	int
[2]	0x0003	RAM 0x00D0	int
[3]	0x0004	RAM 0x00D2	int
[4]	0x0005	RAM 0x00D4	int
f	0x0016	RAM 0x00D6	int

Note that when the native format of an item has been set to **Hexadecimal**, toggling the Watch Window **Hexadecimal** setting will appear to have no effect on that item.

6.5.4.2 *Arbitrary vs. Project-Defined Watch Variables*

For watch variables that were declared in the project source code, only the **Value** and **Display As** properties can be changed. The **Watch Name**, **Memory Bank**, **Addr**, **Elements** and **Data Type** cannot be changed for these variables.

All the parameters can be configured on arbitrary watch variables.

Watch Variables must have names. If you do not specify the name of an arbitrary watch variable before you click **OK**, the watch variable will not be added to the watch variables shown in the Watch Window.

6.5.4.3 *Locals Watch Window*

A separate watch window is available for local variables. Whenever execution halts, the content of the Locals watch window is automatically updated to contain the current scope's local symbols.

6.5.5 Dynamic Event Points

Dynamic Event Points are only available with the ICE cube. The Events window is selectable by clicking **Debug** > **Windows** > **Events**. It allows you to perform complex debugging by configuring conditional breaks and traces.

While breakpoints allow you to select a program location and halt, Dynamic Event Points provide multiple sequences of logical combinations and have multiple potential actions. Breakpoints allow you to select locations within a program to stop, look around, and determine, “How did I get here?” However, debugging is enhanced by the ability to stop and collect information about the target program based upon specified conditions. An example scenario is “when variable OutputV gets set to zero, turn trace buffer on.”

Dynamic Event Points help simplify the debugging process by providing this capability. They monitor the processor to determine a match with logical operations of Program Counter (PC), data bus, data address, instruction type, external logic signals, X Register, Accumulator, Stack Pointer, and Flags.

Breakpoints have one logical input (PC) and one action (Break). Dynamic Event Points, on the other hand, trigger actions when the specified logical condition occurs. An event point can trigger the following actions: break, turn trace on or off, decrement the input counter, trigger the trace buffer, and enable an event sequence.

In summary, Dynamic Event Points provide you with the ability to:

- Define complex breakpoints.
- Characterize multiple test cases to be monitored and logically sequenced.
- Perform any of the following actions: break, turn the trace on, turn the trace off.

6.5.5.1 *Configuring Events*

Use the Events Window to enable or disable event settings anytime during a debug session. To configure events:

1. Click **Debug > Windows > Events** to access debugger events.
2. Click your cursor in the first row, labeled ‘0’.
3. Below the rows are options for 8 and 16 bit threads. Check one or both depending on the needs of your project. Enabling both thread options activates the Combinatorial Operator field.
4. Fill in the applicable thread fields (i.e., Low Compare, Input Select, High Compare, Input Mask), as well as state logic fields (i.e., Next State, Match Count).

   As you make your selection in the Input Select drop-down, you see details in the grayed-out, scrollable box below. Also, use Match Count to specify the number of times an event task occurs before it performs the selected action.

   The input mask for 8-bit threads is applied to the high and low range comparison values, as well as to the input select value. This is done to support range comparisons on subsets of the bits in the input select value. All comparisons take place within the bits specified by the input mask. Other bits are ignored.

   The range values are masked during event editing when the thread states are saved by the Apply button or by switching to a thread state. For example, if the entered low compare value is 06 HEX and the input mask value is 05 HEX, the low compare value after the mask is applied is 04 HEX. The input select value is masked at run time.

In the State Logic section you can choose to start or stop the trace and you can also choose to transition to another logic state once the current condition is met.

5. When finished, click **Apply**. The individual event is now configured and its information appears at row 0.

   If you forget to apply your entries, you are prompted to save. Click **Yes** or **No**.

   To clear all events in the dialog box, click **Clear All**. To disable all events in the dialog box, click **Disable All**.
6. Click row 1 and repeat steps 3-5 to configure another event. Repeat this process for each additional event. 64 Even Points entries are available in Event window.
7. Click **Close** to exit the dialog box. All entries are saved.

As you run events, you can view messages regarding the status in the Debug tab of the Output window. For instance, if you check Break as part of an event, "Hit Event state break" appears in the Output window as the debugger hits the event.

6.5.5.2 *Typical Event Uses*

The many potential uses for events include:

- Find a stack overflow. See Stack Overflow Errors under Invalid Memory Reference in PSoC Designer Online Help System at Help > Help Topics.
- Detect `jmp` or `call` out of program. See Code that will Corrupt Stack under Invalid Memory Reference in PSoC Designer Online Help System at Help > Help Topics.
- Trace a specific range of code.
- Find when a register is written (with optional matching data value).
- Measure interrupt latency.
- Break the 'n'th time a line of code is executed (match count).
- Break on Carry Flag status.
- Break on signals from customer target board.
- Wait for certain number of instructions.
- Count sleep periods.
- Break on specific data in Accumulator on certain instructions (PC).
- Collect trace data reads or writes to specified register.
- Find memory write.

6.5.5.3 *Event Examples*

Here are a few examples for common events.

Find Memory Write

To break on a memory write to address 20h, execute the following example steps.

1. Access the Debugger Events dialog box by clicking **Debug > Windows > Events**.
2. Choose **BITFIELD** in the Parameter box.

   The small help box in the lower left describes the BITFIELD input. Bit 1 is the RAM write flag.
3. Bit 1 is the focus bit of this example; therefore, you need to mask the other bits. The Input Mask field allows you to hide bits that you do not care about. It is an 8-bit number. Set bits in the input mask to mark the bits that you care about (e.g., 0F to get the low four bits, FF to get all eight bits, 01 to get the low bit, 80 to get the high bit, C0 to get the top two bits, etc.). Pick 02 for the input mask to get the RAM write bit.

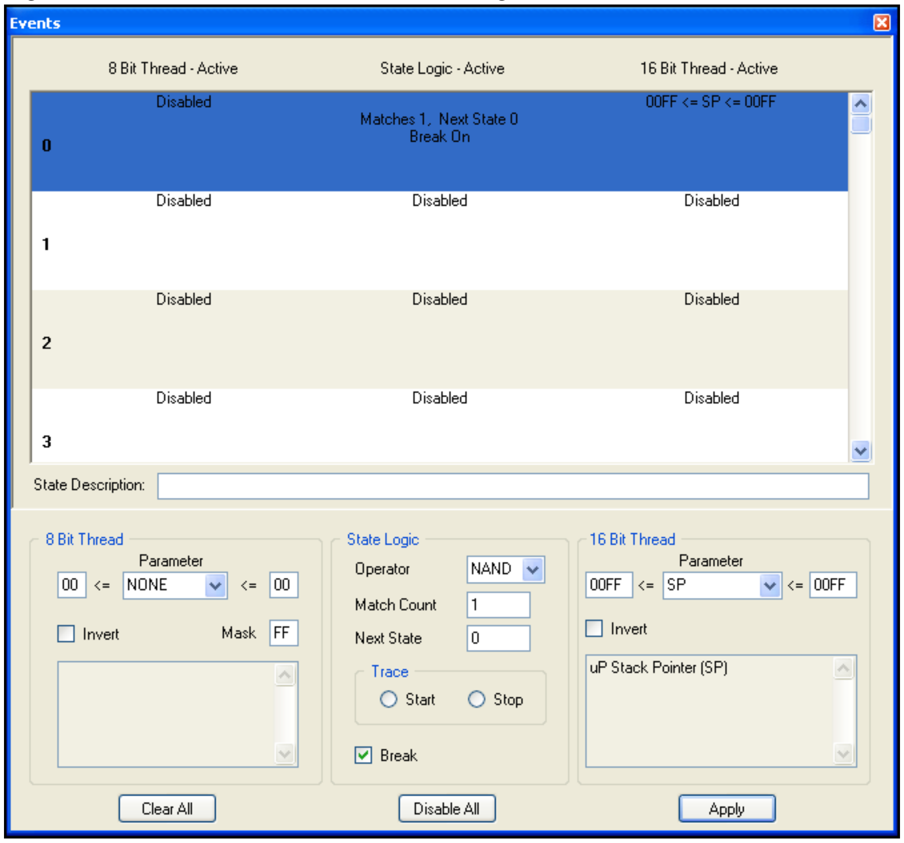
4. Any masked bits need to be zeros in the compare fields. You can see in the help box that the focus bit is active low. It is a ‘0’ when the RAM write is happening. This means you can just set both compare values to ‘0’.
5. Pick **MEM_DA** for the input select. (If you picked **MEM_DA_DB**, you could check the address and the data value.) Note that this selection is different for the ICE Cube support of the CY8C29x66 devices.
6. Set both compare values to ‘20’, your desired address.
7. Pick **AND** for your Combinatorial Operator.
8. Press the **Break** check box. This makes the debugger halt when it sees the RAM write.
9. Click **Apply**.

Stack Overflow

To create an event to break when the Stack Pointer reaches FF:

1. Access the Debugger Events dialog box by clicking **Debug > Events**.

Figure 6-17. Stack Overflow Events Dialog Box



Events

	8 Bit Thread - Active	State Logic - Active	16 Bit Thread - Active
0	Disabled	Matches 1, Next State 0 Break On	00FF <= SP <= 00FF
1	Disabled	Disabled	Disabled
2	Disabled	Disabled	Disabled
3	Disabled	Disabled	Disabled

State Description:

8 Bit Thread: Parameter 00 <= NONE <= 00; Invert ☐; Mask FF

State Logic: Operator NAND; Match Count 1; Next State 0; Trace: Start ○ Stop ○; Break ✓

16 Bit Thread: Parameter 00FF <= SP <= 00FF; Invert ☐; uP Stack Pointer (SP)

2. Set the Input select drop-down to **SP**.
3. Set both the Low compare and High compare values to **00FF**.
4. Check the **Break** check box in the State Logic fields.
5. Click **Apply**, then close the Events window.

The Register A Value, Trace On and Off, and Match Count

To create an event to turn the Trace Off at PC16=0000, turn the Trace On when Register A gets the value 0x32, and then turn the Trace Off and Break after Register A gets the value 0x32 ten times.

1. Access the Debugger Events dialog box (click on **Debug > Windows > Events**).
2. Click on the **State 0** event.
3. Set Input select to PC16, Low compare to 0000, and High compare to 0000.
4. Under State Logic set **Next state to 1** and check **Trace Off.**
5. Click **Apply** to save.
6. Click on the **State 1** event.
7. Turn on the 8-bit thread by checking **Enable 8-Bit Thread**.
8. Set Input Select to A, Low compare and High compare to 32, and leave the Input Mask at FF.
9. Under State Logic set **Next state to 2** and check **Trace On and Break**.
10. Click **Apply** to save.
11. Click on the **State 2** event.
12. Set Input select to A, Low compare and High compare to 32, and leave the Input Mask at FF.
13. Under State Logic set **Next state to 3**, the **Match Count to 10**, and check **Trace Off and Break**.
14. Click **Apply** to save, then close the Events window.

6.5.6 End Point Data

If your project includes one of the USB user modules, you can display USB endpoint data captured from the emulator in a debug window. To do so, select Debug > Windows > USB Endpoint Data. This creates a file named USB Data.txt that captures and displays endpoint data every time the debugger is halted.

To define Areas, use Debug > Configure USB Data Tracking.

For example:
```
PMA0_WA: 00    PMA0_RA: 00
PMA1_WA: 00    PMA1_RA: 00
PMA2_WA: 00    PMA2_RA: 00
PMA3_WA: 00    PMA3_RA: 00
PMA4_WA: 00    PMA4_RA: 00
PMA5_WA: 00    PMA5_RA: 00
PMA6_WA: 00    PMA6_RA: 00
PMA7_WA: 00    PMA7_RA: 00

Area 1 Data...
0x00: 8A 47 6F 84 35 2A 01 28   14 46 00 93 F3 B3 5D 60
0x10: B8 6C C4 A8 31 71 19 41   99 4A 89 AC CE 11 8D 28
0x20: 06 9D 43 8D 22 EE 6C 92   88 3B 33 61 93 A0 16 27
0x30: 8C 0C 9A F0 AA 21 E4 62   80 FA 48 C6 9E E3 66 22
0x40: C8 31 02 8E 14 50 5A 74   AB 38 6F 15 6C 21 E4 DA
0x50: E5 A0 00 1A 0E EA C1 6E   59 46 9A 50 33 06 1F 98
0x60: 8E C3 F4 30 FD 12 01 43   38 52 95 20 D1 EF 71 94
0x70: 1C F8 7B 52 97 E6 12 3C   0C 56 92 2C 89 05 B5 D6
0x80: 1A 1A 80 0B 1E 09 48 3D   4E AB 38 2C CA 42 B4 3A
0x90: 33 DB 13 A3 44 D9 8B 3B   51 DC 00 2D 1D A3 BD 32
0xA0: 01 E7 A1 24 4A 99 04 D8   83 63 C4 EF 88 4A 28 18
0xB0: 48 81 96 16 51 1C F3 9D   1C 2F 92 36 2A AE 5F 6D
```

```
0xC0: 9C BC 0A FC D5 17 96 CF   61 20 FD 14 C4 09 E4 57
0xD0: D2 6C 51 36 3E AE 19 E2   99 5E 04 A1 C6 A9 D1 D0
0xE0: 5E 2D C1 70 97 F5 73 61   57 00 3D CB 77 E5 97 33
0xF0: 33 34 84 C7 2E 82 07 EA   76 9C 24 63 2B 7D 41 20

Area 2 Data...
0x03: 84 35 2A 01

Area 3 has zero length

Area 4 has zero length
```

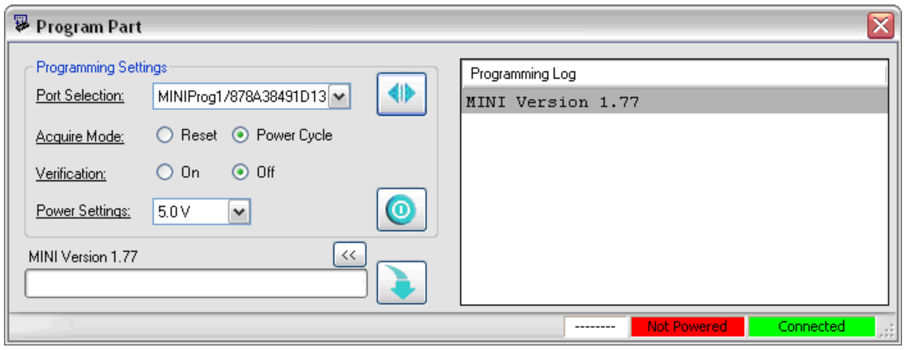
6.6 Programming the Part

Programming a PSoC device is done once debugging is complete. By doing this, you store the ROM data directly in the flash memory of the part. The PSoC device can be reprogrammed many times. Programming is performed by using the Embedded programmer, which is launched by selecting **Program Part**... from the **Program** menu.

This menu selection will launch the embedded programmer. Figure 6-18 shows the embedded programmer. The embedded programmer enables the following actions:

- Select from the available programmers
- Connect or Disconnect to the selected port
- Set Acquire Modes
  - Reset
  - Power Cycle
- Set Verification procedures
  - On
  - Off
- Select Power Settings (if applicable)
- Toggle power
- Program
- Log all processes and error messages

Figure 6-18. PSoC Designer Embedded Programmer



Port Selection: MINIProg1/878A38491D13
Acquire Mode: Reset [✓ Power Cycle]
Verification: On [✓ Off]
Power Settings: 5.0 V
MINI Version 1.77

```
MINI Version 1.77
```

Set the acquire mode appropriately for the device you are programming. If the MiniProg is supplying power to the device it can acquire the device by cycling power. If the device is externally powered the MiniProg can only acquire the device by resetting it. Click the connect button to connect to the device. Click the Power button to power the device, if appropriate. Click the download button to program the device. The status bar and the status window will update as programming progresses. Any error messages will be reported in the status window. Click the details button to see the status window.

Additional information regarding programming the PSoC devices and kits can be found in either the specific kit documentation or the PSoC Programmer User Guide, which can be found in the root PSoC Programmer installation directory.

You can also use the full interface of PSoC Programmer, **Start** > **All Programs** > **Cypress** > **PSoC Programmer**. The full interface provides access to more advanced programming and debugging features.

Caution: There is a limit to the amount of current that can be supplied to the Vdd pin from the programmer. If the target application requires more than 100 mA, it is safer to supply external power to the device. Attempting to draw more than 100 mA from the programmer may cause programming failures or damage the programmer.

7. Flash Protection

Flash Program Memory Protection (FPMP) allows you to select one of four protection (or security) modes for each 64-byte block within the Flash, based upon the particular application.

7.1 FPMP and PSoC Designer

PSoC Designer has a rudimentary mechanism that enables you to set security modes in your Flash program memory. The security (or protection) is set from within PSoC Designer on a per project basis. The protection mechanism is implemented using the System Supervisor Call instruction (SSC). When this command is executed, two bits of data programmed into the Flash selects the protection mode. Table 7-1 lists the available protection options.

Table 7-1. Flash Program Memory Protection Options

Mode Bits	Mode Name	External Read	External Write	Internal Write	Mode Code
00	Unprotected	Enabled	Enabled	Enabled	U
01	Factory Upgrade	Disabled	Enabled	Enabled	F
10	Field Upgrade	Disabled	Disabled	Enabled	R
11	Full Protection	Disabled	Disabled	Disabled	W

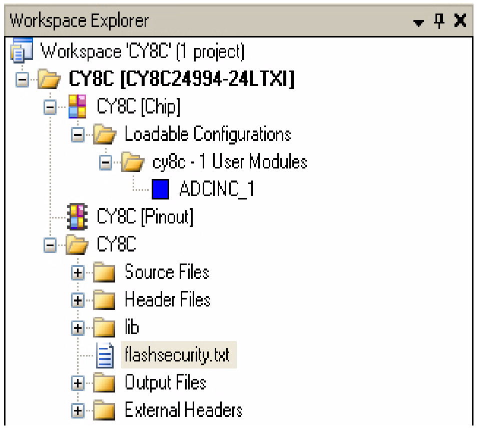
A simple text file called *flashsecurity.txt* is used as the medium for the Flash security. This text file contains comments describing how to alter the Flash security. PSoC Designer validates the correctness of the Flash security data before it is used.

NOTE: The *flashsecurity.txt* file for PSoC Designer v. 2.xx and higher projects that use Flash writes must be set to the correct protection modes. The defaults are set to full-protect mode. To change the protection mode, the part must be bulk erased and re-programmed using the flash security settings.

7.2 About flashsecurity.txt

The FPMP file, *flashsecurity.txt*, is added to each new project and appears in the Workspace Explorer.

Figure 7-1. f*lashsecurity.txt* in Source Tree



PSoC Designer also adds the FPMP file to a cloned project. This is especially useful when cloning projects created with earlier versions of PSoC Designer because earlier versions did not carry this feature. Note that if you do clone a project created in an earlier version of PSoC Designer, you are prompted to update your project, (see "Updating Existing Projects" on page 23).

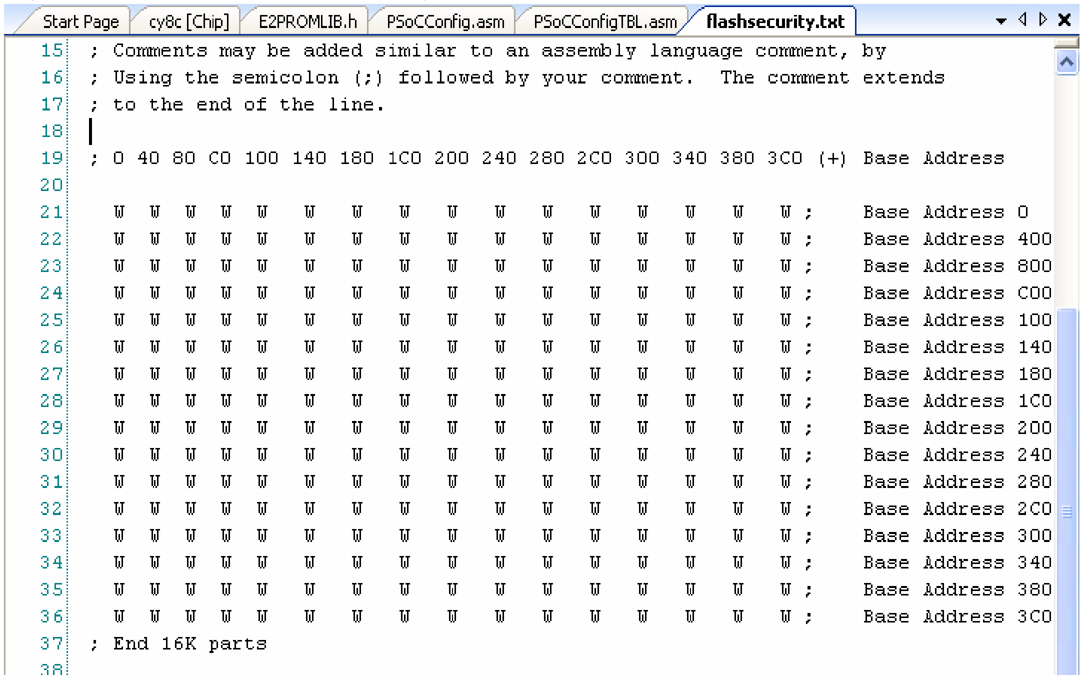
NOTE: If, for some reason, *flashsecurity.txt* is missing or was deleted from the project, the default behavior is to apply Mode Bit 11 Full Protection to the entire program memory.

This information contains instructions on modifying *flashsecurity.txt* and appears at the beginning of this file in PSoC Designer.

```
; Edit this file to adjust the Flash security for this project.
; Flash security is provided by marking a 64-byte block with a
; character that corresponds to the type of security for that
; block, given:
;
; W: Full (Write protected)
; R: Field Upgrade (Read protected)
; U: Unprotected
; F: Factory

; Note #1: Protection characters can be entered in upper or
; lower case.
; Note #2: Refer to the Flash Program Memory Protection section
; in the Data Sheet.

; Comments may be added similar to an assembly language
; comment, by using the semicolon (;) followed by your comment.
; The comment extends to the end of the line.
```

Figure 7-2 is an example of a *flashsecuriy.txt* file.

Figure 7-2. Example of *flashsecurity.txt*

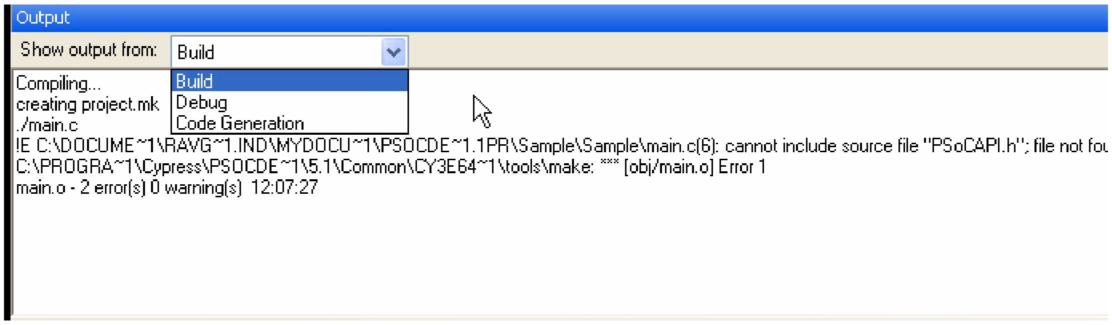


```
; Comments may be added similar to an assembly language comment, by
; Using the semicolon (;) followed by your comment.  The comment extends
; to the end of the line.

; 0 40 80 C0 100 140 180 1C0 200 240 280 2C0 300 340 380 3C0 (+) Base Address

  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 0
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address C00
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 100
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 140
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 180
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 1C0
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 200
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 240
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 280
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 2C0
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 300
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 340
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 380
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;     Base Address 3C0
; End 16K parts
```

7.3 FPMP File Errors

PSoC Designer lets you know when FPMP file errors occur. A dialog box appears, if you edit or enter the wrong information in the *flashsecurity.txt* file, when you are downloading to the ICE or programming a part. A message also appears in the Build tab of the Output Status window.

Figure 7-3. FPMP Error in Output Status Window



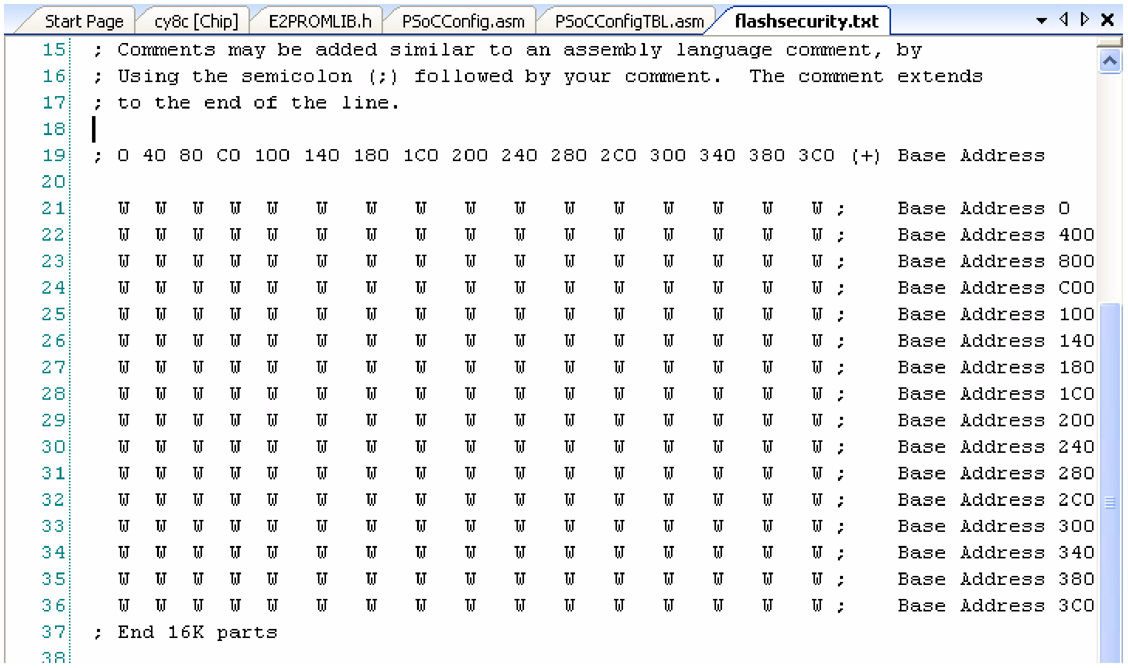
```
Compiling...
creating project.mk
./main.c
!E C:\DOCUME~1\RAVG~1.IND\MYDOCU~1\PSOCDE~1.1PR\Sample\Sample\main.c(6): cannot include source file "PSoCAPI.h"; file not fou
C:\PROGRA~1\Cypress\PSOCDE~1\5.1\Common\CY3E64~1\tools\make: *** [obj/main.o] Error 1
main.o - 2 error(s) 0 warning(s)  12:07:27
```

For example, if you have a Flash data table that can be changed using a Flash write routine, you might have assembly code that looks like this:

```
area Table (ROM, ABS)
org 3C80h
widgetTable:
export WidgetTable
db 57h ; W
db 49h ; I
db 44h ; D
db 47h ; G
db 45h ; E
db 54h ; T
; …. More table entries continue
```

You then unprotect the Flash block associated with this table at address 3C80h and make your change in the *flashsecurity.txt* file as shown in Figure 7-4.

Figure 7-4. Unprotected Flash at 3C80h



```
; Comments may be added similar to an assembly language comment, by
; Using the semicolon (;) followed by your comment.  The comment extends
; to the end of the line.

; 0 40 80 C0 100 140 180 1C0 200 240 280 2C0 300 340 380 3C0 (+) Base Address

  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 0
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 400
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 800
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address C00
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 100
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 140
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 180
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 1C0
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 200
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 240
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 280
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 2C0
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 300
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 340
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 380
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    Base Address 3C0
; End 16K parts
```

Appendix A. Troubleshooting

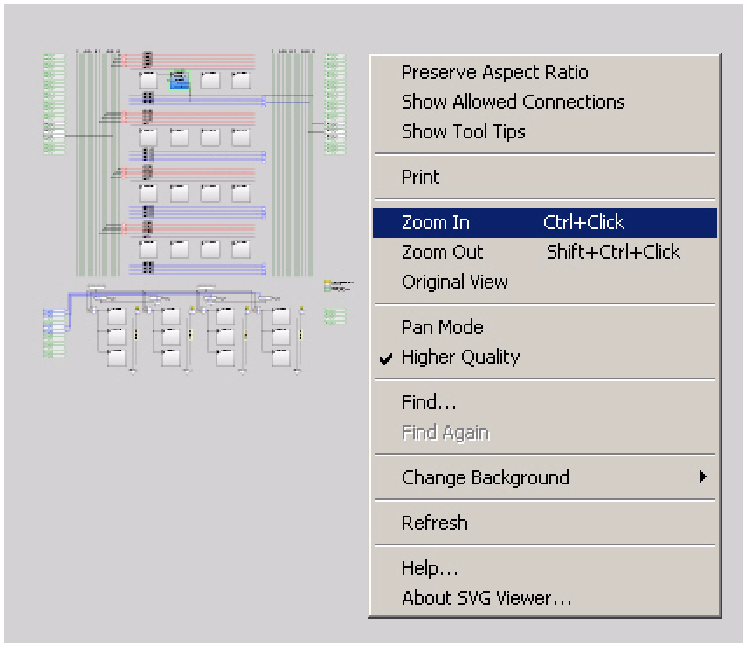
This appendix presents solutions for some potential system problems.

A.1 Troubleshooting the Chip-Level Editor

Problem: You cannot read the fine print in the User Interface that shows the Chip Editor view.

Solution: Place the cursor in the center of the diagram. Right-click the mouse and select Zoom In from the menu. You can also hold the CTRL key and click the image itself.

Figure A-1. Reading the Fine Print



To move the image to a different location on the screen, hold down the ALT key, click in the image with the mouse and drag the image to a place where you want it on the screen.

To zoom out, hold down the CTRL+SHIFT keys and click.

A.2 Troubleshooting the Code Editor

Problem:	You cannot see the Output tab in the Project window.
Solution:	From the View menu select Output.
Problem:	When building the project, you see this error message: “process_begin: CreateProcess((null), `C:\DOCUME~1\bok\LOCALS~1\Temp\make81222.bat`, ...) failed make (e=2): The system cannot find the file specified.”.
Solution:	Rebuild the project, making certain that no other instances of PSoC Designer are building at the same time.
Problem:	When compiling, you see this warning: “!W warning: area ‘<project_name>’ not defined in startup file './obj/boot.o' and does not have an link time address.”
Solution:	This is a bug in PD4.2, see the release notes for more details to "Rebuild the project, making certain that no other instances of PSoC Designer are building at the same time."

A.3 Troubleshooting the Debugger

Problem:	User cannot connect to a pod.
Solution:	There are a few possible issues that may cause this problem. One way to begin the solution process is to create a decision tree that lists the issue and presents potential causes.
Problem:	While debugging, you see the message “Invalid Memory Reference.”
Solution:	Similar to previous solution.
Problem:	User cannot connect to ICE.
Solution:	Similar to the previous solution.

A.4 ICE Configuration

Problem:	It is possible to incorrectly configure the ICE for use with a POD in PSoC Designer, even though the settings appear to be correct in a certain location.
Solution:	There are two places in PSoC designer to adjust Programmer/Emulator settings: Go into Tools > Options > Debugger tab and uncheck Use default ICE connection for all projects. Go into Project > Settings... > Debugger tab, select the ICE from the pull down menu, Select OK for ICE to power pod and in Power supply voltage, select **3.3 V**

A.5 Incorrect Code Compilation

Problem:	Code does not compile correctly after “*** Warning: File has modification time in the future” is received. The application is built and a warning: “*** Warning: File has modification time in the future” is generated at compile time. Any code changes are not updated in the final build and not executed at run time.
Cause	The compiler checks to see if the source file is newer than the output file before regenerating the output file. If the output file is sent to you from a time zone that is

several hours ahead, then the output file is newer than the source file that you just modified. Therefore, the source file is not compiled.

Solution: Use the “rebuild all” function to delete all of the output files and regenerate them using the existing code.

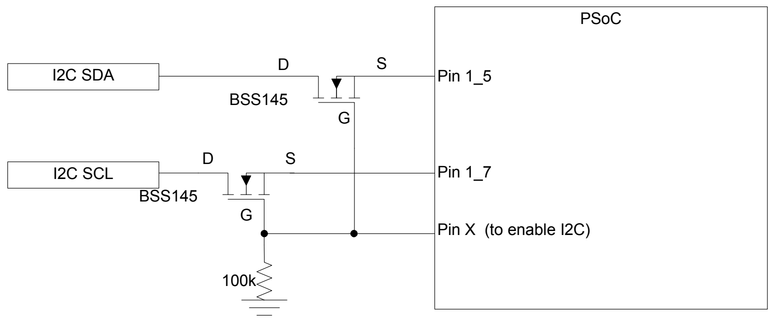
A.6 I2C Hot Swapping

Problem: The PSoC does not function independently as a hot swap controller with I2C.

The protection diodes on the GPIO pins attached to the I2C bus load the bus if the PSoC Vdd is separate from the I2C power supply and is powered down.

Solution: We have a workaround fix. Please find attached block in PDF format.

Figure A-2. I2C Hot Swapping



Notes: Place the N channel FETs used in this block so that the source of the FETs attaches to the PSoC GPIO pin (otherwise the body diodes in the FETs will conduct in the same way as the protection diodes).

This drawing illustrates the BSS145 from Infineon, but you may select something less expensive as long as they are N ch FETs with an RdsON rating for Vgs <= PSoC Vdd. For example, if PSoC Vdd = 3.5V you need an NCH FET with a specified Rds ON for a Vgs <= 3.5V (such as the BSS145).

Setting “Pin X” drive mode to “strong” and writing a 1 to the pin Drive Register enables the PSoC on the I2C bus. Write a 0 to the pin drive register to disable the I2C.

A.7 Manually Turning off the Analog Section

Problem: For situations where low quiescent current is required, such as sleep, you need to to shutoff the entire analog section. To completely shut off the analog section of the PSoC, manually write to the ARF_CR register. You should read about the ARF_CR register, in the Technical Reference Manual (TRM), before changing this register.

Solution: To turn off the Analog Section use the following code snippet.

```
ARF_CR &= ~0x03;
```

A.8 Trace Issues

Problem:	Options under **Debug** > **Trace Mode** are grayed out in the menu of PSoC Designer when in Debug mode if user code is running.
Solution:	Set a breakpoint or click **Halt** on the toolbar. This activates the selections in the menu.

A.9 Using an External USB Hub

Problem: Use an external USB hub to program your PSoC TWICE as fast.

The time it takes to program a PSoC is often reduced when the PSoC is connected to an external USB hub. This is because of the Intel chipset based USB found on many computers.

There are three common USB hub systems: UHCI, OHCI, and EHCI. UHCI and OHCI were developed for the original USB 1.1 spec. UHCI was designed by Intel and is a bare-bones implementation. OHCI is more aggressive and is more widely used. OHCI is much more efficient and attempts to make better use of the bandwidth. UHCI merely adheres to the basic specification. EHCI is designed for the 2.0 (high speed specification) used to communicate to all high-speed devices.

Solution: External 2.0 Hubs operate very aggressively, like the old OHCI interface. Most external Hubs are much more complex than internal Intel chipset based hubs, and make much better use of the available bandwidth. If you have a computer with the Intel USB chipset, connecting a 2.0 hub between your PC and your device increases the data transfer rate. Programming the PSoC is usually TWICE as fast!

This speed increase carries over to other applications as well. The amount of speed increase depends upon the driver used by the Host PC and the type of USB transmission (bulk, isochronous, etc.).

A.10 POD Detection Problem

Problem: While in PSoC Designer you see this error message: **Cannot detect a pod**.

Solution: 1. Check all connections. Make certain that the ICE is powered and is connected to the computer.

2. Make sure the ICE driver is correctly installed. Try uninstalling it and reinstalling it.

3. Make sure you are using the correct POD for the part selected in PSoC Designer. A CY8C29000 POD only emulates CY8C29x parts.

4. Check your PSoC Designer options. There are two places in PSoC Designer to adjust Programmer/Emulator settings:

Select Project > Settings...> Debugger, select the ICE from the drop down menu, click OK for ICE to power pod radio button and under Power supply voltage, click 3.3 V.

5. Start a TightLink Case about the issue. Include as many details about the project and hardware setup as possible. You may have defective hardware.

A.11 Project Cloning Warnings

Problem: You clone a project and receive what seems to be strange errors or warnings.

Solution: To convert your project between device types, select **Start New Project**. See 2.2.1 Clone a Project for details. Type a name for your new project in the Clone dialog box. In the existing configuration window, select the project to clone and the new device type into which to migrate your project. Select **OK**. The new version of your project may require modification if you migrated to a part with fewer resources.

Unfortunately, this migration tool does not always migrate the *boot.tpl* file correctly.

To fix this issue delete the *boot.tpl* and *boot.asm* file from the project directory and restart PSoC Designer. This forces PSoC Designer to regenerate the *boot.tpl* file.

A.12 AreaName Not Defined

Problem: Warning: area 'AreaName' not defined in startup file `./obj/boot.o` and does not have a link time address." applies to assembly code as well.

C compiler generates extraneous warning message relating to RAM AREAS.

In projects that make use of RAM AREAs that are not explicitly defined in the *boot.asm*, the linker may generate the warning message "Warning: area 'AreaN-ame? not defined in startup file '*./obj/boot.o*' and does not have an link time address."

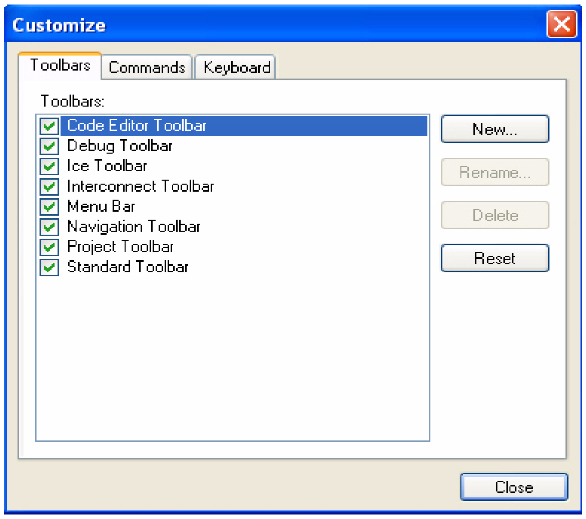
Solution: You may ignore this message. The warning my also safely be ignored in projects that only use assembler.

A.13 General Troubleshooting Issues

Problem: You undocked a minibar, and then closed it by clicking the 'x' on the upper right corner. Now, you cannot get the minibar back.

Solution: From the menu select Tools > Customize > Toolbars tab. Select the minibars to display. This restores the closed minibar.

Figure A-3. Toolbar Customize Menu



Toolbars:

- ✓ Code Editor Toolbar
- ✓ Debug Toolbar
- ✓ Ice Toolbar
- ✓ Interconnect Toolbar
- ✓ Menu Bar
- ✓ Navigation Toolbar
- ✓ Project Toolbar
- ✓ Standard Toolbar

Problem:	You have minibars stacked on the left side of the screen pushing down the viewable screen area.
Solution:	Increase size of the window and move minibars to the desired location. Then switch to another PSoC Designer state to memorize the layout of the current state. Repeat this process for the Chip-Level Editor, the Code Editor, and the Debugger because PSoC Designer memorizes their layouts independently.

Appendix B. Build Process

B.1 Build Utilities

The build utilities are single-purpose applications that contribute to the build process. Table B-1 presents a list of these utilities. These utilities/programs are installed in the Tools folder beneath the root installation of PSoC Designer.

Table B-1. Utilities and Programs

Utility/Program	Developed By	Description
iasm8c.exe	ImageCraft	Assembler
iccomm8c.exe	ImageCraft	Compiler/Assembler/Linker “Engine”
iccm8c.exe	ImageCraft	Compiler
ilinkm8c.exe	ImageCraft	Linker
ilstm8c.exe	ImageCraft	Produces a complete program listing (LST)
icppw.exe	ImageCraft	Preprocessor for compiler
ilibw.exe	ImageCraft	Librarian works on an archive file (.a)
MakeHex.exe	Cypress	Creates an Intel HEX file from a ROM file and applies security records.
psocmakemake.exe	Cypress	Creates a project-specific make file (Project.mk)
VerLst.exe	Cypress	Adds PSoC and ImageCraft version information to the program listing file.
mkdepend.exe	Cypress	Creates a project-specific dependency file (Project.dep)
make.exe	GNU	Make “Engine” version 3.79
sed.exe	GNU	Formatting utility used within the Makefile version 3.02

Make and sed are standard tools with documentation in the `Documentation/Supporting Documents` folder of the PSoC Designer installation directory.

B.2 Make Process

B.2.1 Environment Variables

PATH: Change the path to add the PSoC designer tools directory.

```
Example:
PATH=C:\Program Files\Cypress\PSoC Designer\X.Y\Common\CypressSemiBuild-
Mgr\tools
```

DEVICE: The name of the device.

```
Example: DEVICE=CYC24994
```

BASEDEVICE: The `tools/include` subdirectory of the PSoC Designer install directory that contains the include files for the DEVICE.

```
Example: BASEDEVICE=CY8C24090
```

LASTROM: The last location in ROM.

```
Example: LASTROM=0x3fff
```

B.2.2 MAKE Invocations

PSoC Designer sets the environment variables as indicated above and calls MAKE as follows:

```
make -f <installdir>/tools/makefile clean
make PROJNAME=<projectname> -f <installdir>/tools/makefile makemake
make -f <installdir>/tools/makefile depend
make -f <installdir>/tools/makefile
```

Make invocations are executed from the project root folder.

B.2.3 Build Files

This section describes the files used by the build process. Except for 'Makefile', these files reside in the local project folder.

B.2.3.1 Makefile

This file is a general-purpose MAKE file for all PSoC projects. Therefore, use care when changing the actions in this file, because they apply to all PSoC builds. This file is located in the tools folder, off the main PSoC Designer installation path.

The relevant makefile targets are:

Makemake target

When you use the *psocmakemake.exe* utility, this target produces the *project.mk* from the project.SOC file (see Section B.2.3.2 project.mk).

Depend target

This runs *mkdepend.exe* with the appropriate arguments to generate include file dependencies.

All target

This is the default target that compiles, links, and builds.

Clean target

This removes all object files.

B.2.3.2 project.mk

Project.mk is generated by *psocmakemake.exe* from environment variables and the project's *.soc* file. It is rewritten by the PSoC Designer build process.

This section describes what these symbolic variables are and gives examples of what values they can have. Figure B-1 shows an example of a *project.mk* file for the c24 project using a CY8C24994 device.

Figure B-1. Example Project.mk file

```
PROJNAME=CY8C24x94_SLCD
DEVICE=CY8C24994-24BVXI
BASEDEVICE=CY8C24090
PROJPATH=C:/DOCUME~1/FSU~1.MIC/MYDOCU~1/PSOCDE~1.1PR/CY8C24~1/
CY8C24~1
PSOCDIR=C:/PROGRA~1/Cypress/PSOCDE~1/5.1/Common/CY3E64~1
INCLUDE_PATH=C:/PROGRA~1/Cypress/PSOCDE~1/5.1/Common/CY3E64~1/tools/
include/CY8C24~1
CSRCS= main.c
LIBCSRCS=
ASMSRCS=
LIBASMSRCS= psocconfig.asm psocconfigtbl.asm slcd_1.asm slcd_1int.asm
OBJECT_SOURCES= main.c
FILLVALUE=0x30
LASTROM=0x3fff
LASTRAM=0x3ff
LAST_DATARAM=0x2ff
CODECOMPRESSOR=
MORE_CFLAGS=-Wf-Osize -Wf-LMM4 -D_LMM
RELSTART=0x1a0
CDEFINES=
LIBS=
UMLIBS=
LIB_PATH=
ENABLE_ALIGN_SHIFT=0
LMM=1
SYS_INC_CONTENTS:=SYSTEM_STACK_PAGE:_equ_...etc
SYSTEM_TOOLS=1
CSFLOW=
CONFIG_NAMES=cy8c24x94_slcd
```

The section provides a description of "UMLIBS" and "CSFLOW" of the symbolic names found in a *project.mk* file. You can look at the master make file in an editor and use a text search utility to see how these symbolic names are used.

PROJNAME – Helps the master make file create the file name for the *.hex*, *.lst*, and *.map* files.

DEVICE – Not actually used by the master make file, but used by the *psocmakemake.exe* utility to get things out of the device description XML files. This is a shell environment variable that is set before running the 'makemake' target.

BASEDEVICE – Used to help create the proper path to a device family specific library. This is a shell environment variable that is set before running the 'makemake' target.

PROJPATH – Helps create a path to link in the project's user module library (*libpsoc.a*)

PSOCDIR – Helps form a path to invoke a sub-make target called ‘expanded_lib_prereq’. This is also used in formulating other paths.

INCLUDE_PATH – PSoC Designer version 4.2 with Service Pack 1 has a limitation to the use of this make variable in the master make file. The makemake target (*psocmakemake.exe* process) essentially creates only one path for this variable using the PSOCDIR and BASEDEVICE values. The master make file also statically adds an include paths to ./lib and %PSOCDIR%/tools/include. However, assume a project had a *local.mk* file that wanted to add additional include paths to common folders that where located away from the project folder using a statement like:

```
INCLUDE_PATH:=$(INCLUDE_PATH);../common
```

The assignment, in the master make file, to CY_INCLUDE_PATH receives the potential list of include folders. A list of include paths breaks the path creation that forms variables that produce the *memory.inc* file. The solution is to modify the master make file assignment to:

```
CXY_INCLUDE_PATH=$(subst ;, ,$(INCLUDE_PATH))
CY_INCLUDE_PATH=$(firstword $(CXY_INCLUDE_PATH))
```

Notice that in the INCLUDE_PATH example for the *local.mk* file you must use a colon/equals to break any recursion. Also, notice that the include paths are separated by a colon. This colon is replaced in the master make file by a –I.

CSRCS – This is a (white space separated) list of C files in the root folder of the project. This list is provided as an argument to the ‘depend’ target (*mkdepends.exe* process). The master make file was not designed to add source files outside of the project directory structure.

LIBCSRCS – This is not used since the master make file is not yet made to ‘compile’ C library files.

ASMSRCS – This is a (white space separated) list of assembly files in the root folder of the project.

LIBASMSRCS – This is a list of project assembly files in the ‘lib’ folder of the project. These files are added to the project by the Chip-Level Editor and are stored in the project SOC file, which the ‘makemake’ target (*psocmakemake.exe* process) extracts.

OBJECT_SOURCES – This variable basically governs the link order. The ‘makemake’ target puts the files in alphabetical order. However, if you wish to change the link order you can revise the order of the file list assigned to this variable. See Section B.5.4 Change Link Order on page 137 for an example.

FILLVALUE – This variable is used as a linker argument (e.g. –F0x30) to fill empty areas between code and by the *MakeHex.exe* utility to fill HEX records from the last code record to the end of FLASH with the fill character. A halt (e.g. 0x30) opcode is used.

LASTROM – This is the FLASH size of the device. It is a shell environment variable that is set before running the ‘makemake’ target. It is used in the link process of the master make file.

LASTRAM – This variable comes from the *opts.txt* file and gives the last RAM address. The PSoC Designer build system “asks” the Chip-Level Editor for this value. This value is not really used in the master make file because most devices that do not have paged RAM are 256 bytes. For example this value is 0x7FF for a 2K RAM device and 0xFF for a 256 byte device.

LAST_DATARAM – This variable only shows up in the *project.mk* file when the project has multiple pages of RAM and paging is enabled. The value is equal to the address of the last RAM byte that could be used for data, just before the stack. This value relates to the Stack Page Offset GUI in the Compiler settings of PSoC Designer. For example, if you have a stack page offset of zero (0), with the stack on Page 7, your LAST_DATARAM value is 0x6FF. For example, with a stack page offset of 0x20 (stack on page 7) you get a value, for this variable, of 0x71F.

CODECOMPRESSOR – This is used by the linker in the master make file. PSoC Designer's Project > Settings Compiler GUI shows two options for the Code Compression Technologies: (1) Condensation and (2) Sublimation. These GUI options are only shown for non-RAM-paged device projects. Using the shell environment does not limit you to use these options:

-O: Condensation (or duplicate code removal)

-elim: UserModules: Sublimation (unused User Module API elimination)

For example:

```
CODECOMPRESSOR=-O -elim:UserModules
```

MORE_CFLAGS – This variable adds ImageCraft commands when compiling C files. The PSoC Designer ImageCraft compiler settings will hide or allow certain settings based on the project device. For example, for a device that does not have a hardware Multiplier/Accumulator (MAC) the value in MORE_CFLAGS is set to '-Wf-nomac'.

-Wf-nomac:	Does not generate code to use the MAC.
-Wf-Osize:	Uses calls to math library functions instead of inlining the code.
-Wf-LMM8:	Tells the compiler to generate paged RAM code for eight pages.
-D_LMM:	Needed for C code using >1 page of RAM.
-g -e -c:	These are always used by the master make file. They are, respectively, add debug information, accept C++ comments, and compile file only.

RELSTART – This is the relocatable start address, for example 0x140. This is the starting address for the text area (above the TOP area).

CDEFINES – This is added to a command line for compiling C source files. 'Defines' are prefixed with a '-D' and un-defines are prefixed with a '-U'. For example:

```
CDEFINES=-DSET_SPI –DMAX=2 –UADD_DBG
```

LIBS – This is a list of object (*.o*) and library (*.a*) files that you wish to link in from outside the project. The elements must be separated by white space.

LIB_PATH – These are folder locations that should hold the LIBS. Elements in this list should be separated by semicolons (;). Short (e.g., 8.3) path names should be used.

ENABLE_ALIGN_SHIFT – This is not used.

LMM – This helps the master make file when a project wishes to use paged RAM. The values are a 1 or nothing.

SYS_INC_CONTENTS – The value (e.g., list) for this variable is quite long. This value is a mechanism used by PSoC Designer to push information about memory settings into an include file that the master make file creates. Possibly, the additional elements in this list could be added so that you can put your 'equates' in the memory include file. The master make file effectively creates the memory include file by redirecting *memhead.tpl* from the proper device include folder to the top of the memory include file. Then each element in this list is placed in the memory include file after replacing '_equ_' with ' equ ' (spaces added where underscores were). The remainder of the memory include file comes from the redirection of the *memfoot.tpl* file (in the proper devices include folder). Again, reading the master make file and understanding this will help a lot.

SYSTEM_TOOLS – This value is 1 for ImageCraft.

CONFIG_NAMES – This lists (white space separated) the overlay (or configuration) names created in the device editor. This helps the master make file create linker switches to ensure that the RAM data declared in User Modules gets located on the same RAM page for each overlay.

B.2.3.3 *Local.mk*

This file contains additional make instructions. It is not touched by the make process. Here you can add and/or modify the build variables set in *project.mk*.

B.2.3.4 *project.dep*

Created by the ‘mkdepend’ via the make depend command. This file contains the include dependencies for the project. It is rewritten by the build process.

B.2.3.5 *local.dep*

This file contains additional file dependencies and make commands. It is not touched by the make process.

Since this is the last thing in the make file, assignments here override those in the main make file. See B.5.2 Boot Loader Example on page 136.

B.2.3.6 *custom.lkp*

This is a ‘legacy’ file inserted via sed into the linker arguments. This is typically used to locate ‘areas’ with the -b switch (for example: -bfoo:0x1C00.0x1FFF)

B.2.3.7 *opts.txt*

This is a file created by PSoC Designer to take the GUI settings for the compiler and linker and translate them to ImageCraft arguments and other MAKE variables used by the master make file. The contents of this file gets pulled into the *project.mk* file

NOTE : The following section (appendix B.3) is applicable only to PSoC Designer 5.0 SP4 or earlier versions.

B.3 Moving the Build System to Another PC

Typically when there is an interest in using the shell build environment it is because you wish to create a PSoC HEX file with a minimum build environment on another workstation.

Archive or move the tools folder and all its subfolders. This ensures the same compiler version/release and the same include and library files.

B.3.1 ImageCraft License key

If you are using C source files then you want to make sure that the workstation you intend to compile files on has a valid compiler license. The utility program *icc.exe* is a wrapper for the ImageCraft compiler. This wrapper validates the compiler license stored in the registry. The wrapper looks for the compiler license in this registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Cypress MicroSystems\PSoC Designer\AddIn\Compilers\IMAGECRAFT
```

There should be a String value (REG_SZ) in this key named COMPILER_LICENSE. Its values are your license (case sensitive). Create this key by exporting the key and then importing it onto another PC or add it manually. This registry key location changed in PSoC Designer version 4.2 (*icc.exe*). Earlier versions of *icc.exe* look for the license in the ‘AddIn’ key.

You then need to put your PSoC project files in a folder on the other PC. When you build your project on the other machine make sure that the PSOCDIR variable, as well as other path-related variables (e.g. INCLUDE_PATH), in a *project.mk* or *local.mk* file, point to the appropriate paths.

B.4 Building Project Through Command Line

B.4.1 Command Line Instructions

PSoC Designer 5.1 provides executable file, PDCLI.exe, clean, generate, and build the project through command line. The location of the PDCLI.exe file is under installation folder = \Cypress\PSoC Designer\5.1\ PSoC Designer 5\PDCLI.exe.

PDCLI provide four types of operation:

1. Cleaning the project files

This operation takes two arguments, command and project path. The command to invoke this operation is CP. The project path should be .cmx file.

2. Generate project files

This operation takes two arguments, command and project path. The command to invoke this operation is GP. The project path should be .cmx file.

3. Clean, generate, and build project

This operation takes three arguments, command, project path, and compiler. The command to invoke this operation is CGBP. The project path should be cmx file. The compiler should be IMAGECRAFT, ImageCraftPro, or HI-TECH. When the build fails, the console should show the error message.

4. Help

When there is no argument, PDCLI shows help function

Examples

1. Cleaning the project files

PDCLI "**CP**" "C:\MyProject\MyProject.cmx"

2. Generate project files

PDCLI "**GP**" "C:\MyProject\MyProject.cmx"

3. Clean, generate, and build project

PDCLI "**CGBP**" "C:\MyProject\MyProject.cmx" "IMAGECRAFT

PDCLI "**CGBP**" "C:\MyProject\MyProject.cmx" "HI-TECH"

PDCLI "**CGBP**" "C:\MyProject\MyProject.cmx" "ImageCraftPro"

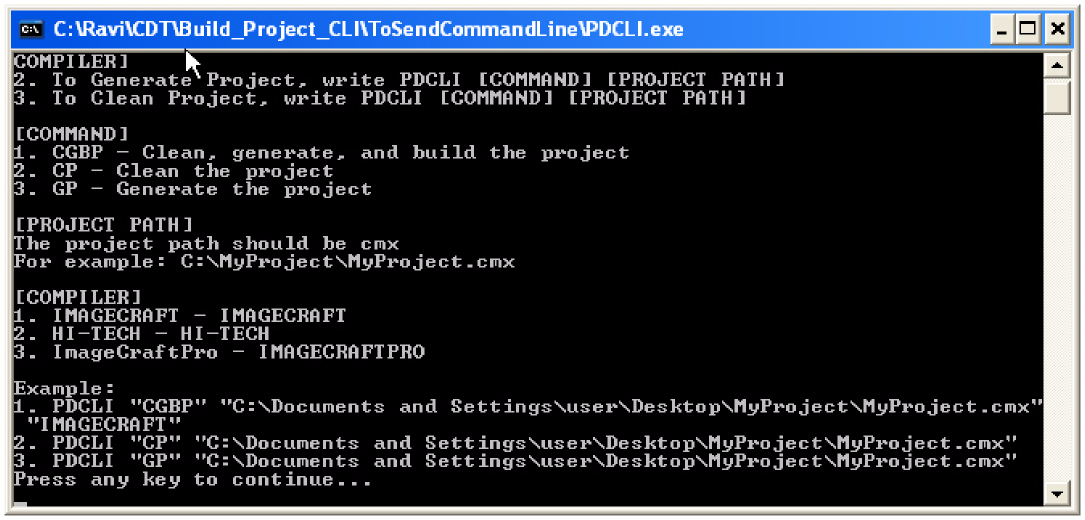
4. Help

PDCLI

B.4.2 Executable File (PDCLI.exe)

The following figure helps you to perform build, clean, and generate project using command line interface

Figure B-2. PDCLI.exe



```
COMPILER]
2. To Generate Project, write PDCLI [COMMAND] [PROJECT PATH]
3. To Clean Project, write PDCLI [COMMAND] [PROJECT PATH]

[COMMAND]
1. CGBP - Clean, generate, and build the project
2. CP - Clean the project
3. GP - Generate the project

[PROJECT PATH]
The project path should be cmx
For example: C:\MyProject\MyProject.cmx

[COMPILER]
1. IMAGECRAFT - IMAGECRAFT
2. HI-TECH - HI-TECH
3. ImageCraftPro - IMAGECRAFTPRO

Example:
1. PDCLI "CGBP" "C:\Documents and Settings\user\Desktop\MyProject\MyProject.cmx"
 "IMAGECRAFT"
2. PDCLI "CP" "C:\Documents and Settings\user\Desktop\MyProject\MyProject.cmx"
3. PDCLI "GP" "C:\Documents and Settings\user\Desktop\MyProject\MyProject.cmx"
Press any key to continue...
```

B.5 Examples

B.5.1 Batch Build File

This is a .bat file that builds a project; here PSoC Designer is installed in `c:\a\pd`

```
set PSOCTOOLS=c:\a\pd\tools
set PATH=%PSOCTOOLS%;c:\winnt\system32;c:\winnt
set DEVICE=CY8C24423B
set BASEDEVICE=CY8C24000B
set LASTROM=0xFFF
make -f %PSOCTOOLS%\Makefile clean
make PROJNAME=aa -f %PSOCTOOLS%\Makefile makemake
make -f %PSOCTOOLS%\Makefile depend
make -f %PSOCTOOLS%\Makefile
```

B.5.2 Boot Loader Example

This ‘make’ trick allows a project to have a block of RAM that is shared between a boot-loader and the boot-loaded code. This trick tells the linker to begin RAM allocation at some point other than location 0. In this example we will have this RAM scratch pad area from RAM address 0 to 4.

In the project's *local.mk* file add a ‘new’ MAKE variable as follows:

```
STARTRAM:=0x5
```

In the project’s *local.dep* file add this line:

```
DATARAM:=-bdata:$(STARTRAM).0xFF
```

Now in your boot loader code use an absolute RAM area to declare variables in this ‘hidden/shared’ area like:

```
AREA Foo(RAM,ABS)
ORG 0
export _Z1, _Z2, _Z3, _Z4
_Z1: blk 1
_Z2: blk 1
_Z3: blk 1
_Z4: blk 2
```

B.5.3 Add External Files to the Project

This is a trick for *local.mk* to have a ‘custom’ rule to build/add files outside of the project folder.

```
# CSRCS is used by makedepends to get the external headers/dependencies
CSRCS:=$(CSRCS) ../common/foo.c
OBJECT_SOURCES:=$(OBJECT_SOURCES) foo.c
# new rule to tell MAKE to get C files from ../common
# DON'T have files in COMMON with the same name as this project's Source
# Files because we won't know which file MAKE will build last.

obj/%.o : ../common/%.c
ifeq ($(ECHO_COMMANDS),novice)
   echo $(call correct_path,$<)
endif

   $(CCMD) $(CFLAGS) $(CDEFINES) $(INCLUDEFLAGS) $(DEFAULTCFLAGS) -o $@
$(call correct_path,$<)
```

B.5.4 Change Link Order

```
# The CSRCS and ASMSRCS lists are found in project.mk.
# project.mk is generated by psocmakemake.exe and reflects
# the project settings (e.g. files, linker options, part family)

# Alter the filter patterns to change the
# link order.

      C_MY_SORT=$(filter s%.c, $(CSRCS))
      C_MY_SORT+=$(filter m%.c, $(CSRCS))
      C_THE_REST= $(filter-out $(C_MY_SORT), $(CSRCS))
      C_ORDER =$(C_MY_SORT) $(C_THE_REST)

      ASM_MY_SORT=$(filter j%.asm, $(ASMSRCS))
      ASM_MY_SORT+=$(filter h%.asm, $(ASMSRCS))
      ASM_THE_REST=$(filter-out $(ASM_MY_SORT), $(ASMSRCS))
      ASM_ORDER=$(ASM_MY_SORT) $(ASM_THE_REST)

      OBJECT_SOURCES=$(C_ORDER) $(ASM_ORDER)

# Note: Changes/edits to this file will not re-link
# the project since this file is not a prerequisite
# condition for the Link 'target'.
```

Glossary

A

active windows	Subsystem-related windows that are open and workable.
analog PSoC blocks	Basic programmable opamp circuits. There are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more.
Application Programming Interface (API)	A series of software routines that comprise an interface between a computer application and lower-level services and functions (for example, user modules and libraries). APIs serve as building blocks for programmers that create software applications.
Code Editor	PSoC Designer subsystem where users edit and program C Compiler and assembly language source files.
assemble (combined with compiling)	Assembling, in PSoC Designer, translates all relative-addressed code into a single .rom file with absolute addressing.

B

build/link	Building your project in PSoC Designer links all the programmed functionality of the source files and loads it into a .rom file, which is the file you download for debugging and programming.

C

compile (combined with assembling)	Compiling, in PSoC Designer, takes the most prominent, open file and translates the code into object source code with relative addresses.

D

debugger	A hardware and software system that allows the user to analyze the operation of the system under development. A debugger usually allows the developer to step through the firmware one step at a time, set break points, and analyze memory.

design (export/ import)	One or more loadable configurations that can be exported from a project then imported and used in a new or existing project. A loadable configuration consists of one or more "placed" user modules with module parameters, Global Resources, set pinouts, and generated application files.
design browser	Venue to identify reusable designs for import to PSoC Designer projects.
Chip-Level Editor	PSoC Designer subsystem where you choose/configure your device.
digital PSoC blocks	The 8-bit logic blocks that can act as a counter, timer, serial receiver, serial transmitter, CRC generator, pseudo-random number generator, or SPI.
dynamic reconfiguration	Dynamic Reconfiguration allows for applications to dynamically load and unload configurations. With this feature, your single PSoC MCU can have multiple functions.

F

family of devices	A family of PSoC devices is a group of devices with similar characteristics but different part numbers. For example, the CY8C20396A and CY8C20496A parts are very similar. Both devices are members of the CY9C20x96 device family. A family is frequently listed using Xs in place of the numbers that differ within the family.

I

ice-Cube	In-Circuit Emulator (ICE) and USB adapter for seamless USB connection, debugging, and programming.
interrupt service routine (ISR)	A block of code that normal code execution is diverted to when the M8C receives a hardware interrupt. Many interrupt sources may each exist with its own priority and individual ISR code block. Each ISR code block ends with the RETI instruction, returning the device to the point in the program where it left normal program execution.

L

link/build	Linking your project in PSoC Designer links all programmed functionality of the source files (with absolute addressing) and loads it into a .rom file, which is the file you download for debugging and programming.

M

M8C	An 8-bit Harvard Architecture microprocessor. The microprocessor coordinates all activity inside a PSoC by interfacing to the Flash, SRAM, and register space.

miniProg1	Developmental programmer that provides a low-cost solution for learning about, programming and evaluating the PSoC.

P

pod	Part of the ICE that emulates functionality, in which debugging occurs.
PSoC	PSoC® is a registered trademark and Programmable System-on-Chip™ is a trademark of Cypress Semiconductors.
PSoCEval1	Evaluation board that provides a low-cost solution for learning about, programming, and evaluating the PSoC.
PSoC blocks	*See analog blocks* and *digital blocks*.
PSoC Designer	The software for Cypress MicroSystems' Programmable System-on-Chip technology.
PSoC Programmer	New, multi-functional programming software accessible from within PSoC Designer.

S

source tree	Project file system displayed by default in left frame of Workspace Explorer.
subsystem	PSoC Designer has three subsystems: Chip-Level Editor, Code Editor, and Debugger.

U

USB adapter	Port connection to work the ICE in PSoC Designer v. 4.1 or later.
user modules	Accessible, preconfigured function that, once placed and programmed, will work as a peripheral in the target device.

Index