

PSoC™ Designer:
Integrated Development Environment

User Guide
Revision 1.09

CMS10005A
Last Revised: May 30, 2001
Cypress MicroSystems, Inc.

Copyright Information

Copyright © 2000-2001 Cypress Microsystems, Inc. All rights reserved.

PSoC™ (Programmable System on Chip) is a trademark of Cypress Microsystems, Inc.

Athlon is a trademark of Advanced Micro Devices, Inc.

Copyright © 1999-2000 ImageCraft Creations Inc. All rights reserved.

InstallShield® is a registered trademark and service mark of InstallShield Software Corporation in the United States and/or other countries.

All Microsoft products referenced herein are either trademarks or registered trademarks of Microsoft Corp.

All Intel products referenced herein are either trademarks or registered trademarks of Intel Corporation.

The information contained herein is subject to change without notice.

Two-Minute Overview

This two-minute overview of *PSoC Designer: Integrated Development Environment User Guide* was purposefully placed up front for you advanced engineers who are ready to configure and program the chip but need a *quick* point in the right direction. **Make sure you have the latest version of the software.** (Now we only have a minute and-a-half left.)

Overview 35 seconds You have the M8C, PSoC Designer, and the vision... This guide provides:

- installation procedures
- interface overview
- instructions for creating a project
- instructions for configuring the device
- instructions for editing assembly-source files
- instructions for compiling files
- instructions for building the project
- instructions for debugging the project
- project tutorial
- troubleshooting tips.

Basics 30 seconds PSoC Designer contains three subsystems; Device Editor, Application Editor, and Debugger. Start by creating a project, and go from there...



1. Create a project.



2. Configure device in Device Editor.



3. Edit source files in Application Editor.



4. Debug project in Debugger.

Quick Reference 15 seconds Click a hyperlink to reference key material:

[Section 2. Installation](#)

[Section 10. Project Tutorial](#)

[Troubleshooting](#)

Bottom Line 10 seconds Programmable System on Chip PSoC™ Designer empowers you to customize the functionality you desire into the M8C microprocessor.



Time's up... Now get to work.

Documentation Conventions

Following, are easily identifiable conventions used throughout the PSoC Designer suite of product documentation.

Convention	Usage
Times New Roman Size 10	Displays an input command: <code>cmasm sourcefile.asm -b -t nn</code>
Courier Size 12	Displays output: <code>>cmasm testfile -t 4</code> CMASM Version 2.20 For C series Microcontrollers I 2000 Cypress MicroSystems Inc. Complete! >
Courier Size 12	Displays file locations: <code>c:\ ...cd\icc\</code>
<i>Italics</i>	Displays file names: <code>sourcefile.rom</code>
[Ctrl] [C]	Displays keyboard commands: [Enter]
File >> Open	Displays menu paths: Edit >> Cut

Notation Standards

Following, are notation standards used throughout the PSoC Designer suite of product documentation.

Internal Registers:

Notation	Description
A	Primary Accumulator
CF	Carry Flag
expr	Expression
F	Flags (ZF, CF, and Others)
I	Operand 1 Value
K	Operand 2 Value
PC	(PCH,PCL)
SP	Stack Pointer
X	X Register
ZF	Zero Flag

Assembler Directives:

Symbol	Assembler Directive
AREA	Area
BLK	RAM Block (in Bytes)
BLKW	RAM Block in Words (16 Bits)
DB	Define Byte
DS	Define ASCII String
DSU	Define UNICODE String
DW	Define Word (2 Bytes)
DWL	Define Word with Little Endian Ordering
ELSE	Alternative Result of IF...ELSE...ENDIF
ENDIF	End of IF...ELSE...ENDIF
EQU	Equate Label to Variable Value
EXPORT	Export
IF	Conditional Assembly
INCLUDE	Include Source File
MACRO/ENDM	Macro Definition Start/End
ORG	Area Origin

Table of Contents

Two-Minute Overview	1
<i>Quick-start summary for advanced users who are ready to dive in.</i>	
Documentation Conventions	2
<i>Lists conventions used in this guide and throughout the PSoC Designer suite.</i>	
Notation Standards	3
<i>Lists notation for quick-reference used in this guide and throughout the PSoC Designer suite.</i>	
Section 1. Introduction	7
<i>Describes purpose of this guide and overviews section and product information.</i>	
1.1. Purpose.....	
1.2. Section Overview.....	
1.3. Product Updates.....	
1.4. Support.....	
Section 2. Installation	9
<i>Describes how to install PSoC Designer.</i>	
2.1. Hardware Requirement Checklist.....	
2.2. Software Requirement Checklist.....	
2.3. Installing the System.....	
Section 3. Using the IDE	15
<i>Discusses high-level functionality within PSoC Designer IDE.</i>	
3.1. System Diagram.....	
3.2. File Types and Extensions.....	
3.3. Project Manager.....	
3.4. Edit Windows.....	
3.5. Status Window.....	
Section 4. Creating a Project	23
<i>Describes selecting a device and creating a project.</i>	
4.1. Creating a Project.....	
4.2. Configuration Method.....	
Section 5. Device Editor	29
<i>Describes comprehensive use of Device Editor.</i>	
5.1. Selecting User Modules.....	
5.2. Placing User Modules.....	
5.3. Deploying Interconnectivity.....	
5.4. Specifying Pin-out.....	
5.5. Tracking Device Space.....	
5.6. Generating Application Files.....	

Section 6. Application Editor 49

Describes all source-editing options.

- 6.1. File Definitions and Recommendations.....
- 6.2. Modifying Files.....
- 6.3. Adding Files.....
- 6.4. Removing Files.....

Section 7. Assembler 55

Describes complete functionality and use, including compiling/assembling files.

- 7.1. Accessing the Assembler.....
- 7.2. The Microprocessor.....
- 7.3. Assembly File Syntax.....
- 7.4. List File Format.....
- 7.5. Assembler Directives.....
- 7.6. Instruction Set.....
- 7.7. Compiling/Assembling Files.....

Section 8. Builder 61

Describes building a project and transparent system functions.

- 8.1. Building a Project.....
- 8.2. C Compiler.....
- 8.3. Linker/Loader.....
- 8.4. Librarian.....

Section 9. Debugger 65

Describes connecting to the ICE and debugging the project.

- 9.1. Connecting the ICE.....
- 9.2. Downloading to Pod.....
- 9.3. Debug Strategies.....
- 9.4. Menu Options.....
- 9.5. Programming the Part.....

Section 10. Project Tutorial 77

Guides you step by step through creating and debugging a project.

- 10.1. Create.....
- 10.2. Configure.....
- 10.3. Compile.....
- 10.4. Build.....
- 10.5. Debug.....
- 10.6. Results.....

Troubleshooting..... 85

Data Dictionary 89

Index..... 90

Figure 1: Welcome10

Figure 2: PSoC Designer Setup Wizard.....10

Figure 3: License Agreement.....11

Figure 4: Choose Destination Location11

Figure 5: Select Program Folder.....11

Figure 6: Start Copying Files12

Figure 7: Setup Status12

Figure 8: PSoC Designer Installation Wizard Completed13

Figure 9: Activate Emulator Driver13

Figure 10: Source Tree.....17

Figure 11: Device Editor Subsystem18

Figure 12: Application Editor Subsystem19

Figure 13: Debugger Subsystem20

Figure 14: Cascaded Windows21

Figure 15: Status Window.....21

Figure 16: Start Dialog Box.....23

Figure 17: New Project Dialog Box24

Figure 18: New Configuration Dialog Box25

Figure 19: Parts Catalog Dialog Box.....25

Figure 20: Existing Configuration Dialog Box.....27

Figure 21: Select User Module in Device Editor Toolbar29

Figure 22: User Module Options29

Figure 23: User Module Data.....30

Figure 24: User Module Selections31

Figure 25: Place User Module in Device Editor Toolbar.....32

Figure 26: User Module on PSoC Block32

Figure 27: Right-Click Parameters.....34

Figure 28: User Module Parameters35

Figure 29: Global Resources36

Figure 30: Interconnectivity Parameters.....38

Figure 31: Specify Pin-Out in Device Editor Toolbar39

Figure 32: Specify Pin-Out.....40

Figure 33: PSoC Block Resources41

Figure 34: Application Generation Status.....42

Figure 35: *PWM16_1.h* File43

Figure 36: Timer32 on Four Digital PSoC Blocks.....44

Figure 37: 32-Bit Timer Interrupt Hook.....45

Figure 38: Add New File53

Figure 39: Status Window.....61

Figure 40: Debugger Hardware Components65

Figure 41: Hardware Components Connected66

Figure 42: Pod (Top)67

Figure 43: Pod (Bottom)67

Figure 44: Debug Breakpoints71

Figure 45: Debug ASM Watch Properties72

Figure 46: Pod Programming Socket74

Figure 47: DAC6SC and Timer16 User Module/PSoC Block Resources.....78

Figure 48: DAC6SC User Module Parameters.....78

Figure 49: Timer16 User Module Parameters79

Figure 50: DAC6SC and Timer16 Global Resources79

Figure 51: AnalogOutBuffer_3 Port_0_280

Figure 52: PortPin P0[2]81

Figure 53: *main.asm* Source Code for Tutorial.....82

Figure 54: *Timer16_1INT.asm* Source Code for Tutorial.....82

Section 1. Introduction

1.1 Purpose

The *PSoC Designer: Integrated Development Environment User Guide* will guide you from start to finish on utilizing PSoC Designer to configure, program, compile, build, emulate, and debug your customized system that runs from the M8C microprocessor.

For comprehensive details on compiling and assembling, see:

- *PSoC Designer: C Language Compiler User Guide*
- *PSoC Designer: Assembly Language User Guide*

Together, these three user guides complete the PSoC Designer documentation suite.

1.2. Section Overview

<u>Section 1. Introduction</u>	Describes the purpose of this guide, overviews each section, and gives product upgrade and support information.
<u>Section 2. Installation</u>	Lists system hardware and software requirements and runs through the installation procedure.
<u>Section 3. Using the IDE</u>	Discusses the functional format of the system interface.
<u>Section 4. Creating a Project</u>	Describes how to create a project.
<u>Section 5. Device Editor</u>	Details how to select and place User Modules, implement interconnectivity, specify pin-out, track device space, and generate application files.
<u>Section 6. Application Editor</u>	Describes all source-editing options.

<u>Section 7. Assembler</u>	Details assembly-language source and compiling/assembling project files.
<u>Section 8. Builder</u>	Describes how to build a project and details transparent linker/loader and librarian functionality.
<u>Section 9. Debugger</u>	Describes connecting to the In-Circuit Emulator (ICE) and debugging the project.
<u>Section 10. Project Tutorial</u>	Guides you step by step through creating, configuring, compiling, building, and debugging a project.

1.3. Product Upgrades

Cypress MicroSystems provides scheduled upgrades and version enhancements for PSoC Designer *free of charge*. You can order the upgrades from your distributor on CD-ROM or, better yet, download them directly from the Cypress MicroSystems web site at <http://www.cypressmicro.com/>.

Also provided at the web site are critical updates to system documentation. To stay current with system functionality you can find documentation updates under the Documentation hyperlink, again, at <http://www.cypressmicro.com/>.

Check the Cypress MicroSystems web site frequently for both product and documentation updates. As the M8C and PSoC Designer evolve, you can be sure that new features and enhancements will be added. To register and receive product update notification go to <http://www.cypressmicro.com/registerme/>.

1.4. Support

Free support for PSoC Designer is now available online at <http://www.cypressmicro.com/support/>. Resources include FAQs, Discussion Forums, Application Notes, and Support Technicians.

Section 2. Installation

In this section you will learn recommended hardware and software requirements to optimally run PSoC Designer as well as how to install the system.

2.1 Hardware Requirement Checklist

The following hardware specifications have been deemed best for running PSoC Designer:

- 166 MHz Pentium® or better (AMD Athlon™ or better)
- SVGA Monitor Graphics (High Color 16 Bit, 1024x768 Resolution)
- CD-ROM Drive
- 64 MB RAM
- 100 MB of Free Hard Drive Space
- Available (Un-used) EPP Parallel Port for In-Circuit Emulator (ICE)
- ICE, Pod, CAT5 Patch Cable, Parallel Port Cable, and Power Adapter **Supplied in Kit**
- PSoC Pup™ Board for Project Tutorial [Section 10](#) **Supplied in Kit**

2.2. Software Requirement Checklist

The following software specifications are required to run PSoC Designer:

- Windows® 95, 98, NT 4.x, 2000, or ME
 - Microsoft Internet Explorer 5.x
-

2.3. Installing the System

To install PSoC Designer, execute the following procedure (estimated elapsed time is 2-4 minutes):

1. Place Cypress MicroSystems PSoC Designer CD-ROM in drive.
2. At the Welcome screen, single-click **Next**. See Figure 1. (If the Welcome screen does not automatically appear, click **Start >> Run** and **Browse** your CD drive for PSoC.exe. Once located, click **OK**. This action will trigger the Welcome screen.)

The system will extract files and then run InstallShield® Wizard.

3. At the PSoC Designer Setup Wizard, click **Next**. See Figure 2.



Figure 1: Welcome

Click **Back** at any time during installation if you need to view or modify the previous screen. Click **Cancel** at any time to halt installation.



Figure 2: PSoC Designer Setup Wizard

4. At the License Agreement screen, use [**Page Down**] to view the terms of the agreement. When satisfied, single-click **Yes**. See Figure 3.
5. At the Choose Destination Location screen, click **Next** to install the system to the default directory path of C:\Cypress MicroSystems\PSoC Designer. If you wish to choose an alternative location, click **Browse** and select a different directory path. See Figure 4.

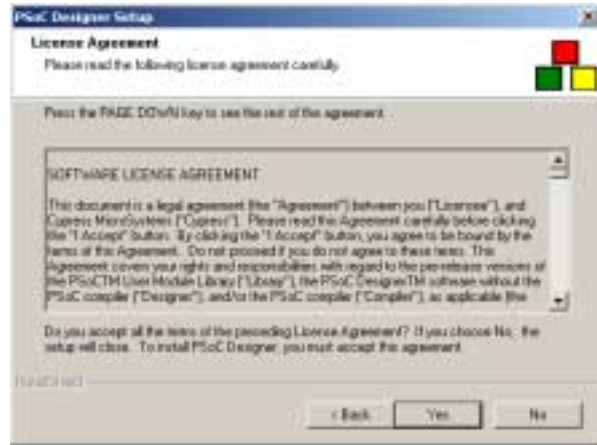


Figure 3: License Agreement

6. At the Select Program Folder screen, click **Next** to accept the default folder of Cypress MicroSystems in which to add system icons. If you prefer, you can type a new folder name in the Program Folders field or select an existing folder from the Existing Folders field. See Figure 5.



Figure 4: Choose Destination Location



Figure 5: Select Program Folder

- At the Start Copying Files screen, scroll to review your current settings. If you are satisfied, click **Next**. If not, click **Back** to return to view and/or modify settings in a previous screen. See Figure 6.

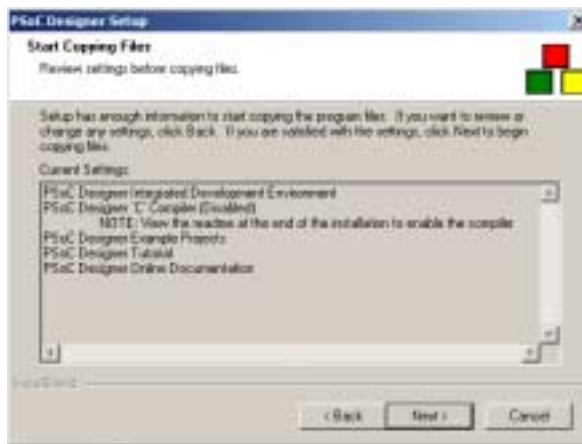


Figure 6: Start Copying Files

- At the Setup Status screen, you will see a status, in percentage complete, of system installation. Click **Cancel** if you wish to cancel installation at this time. See Figure 7.

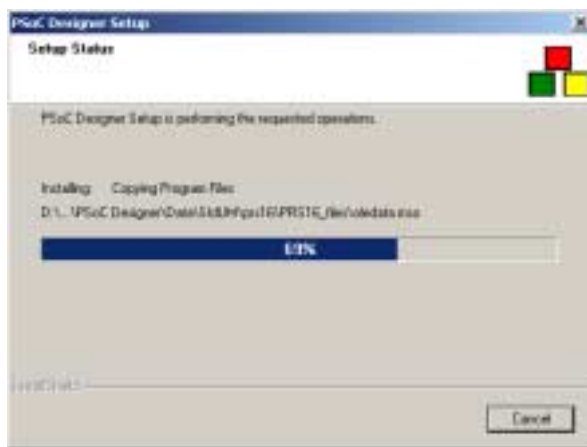


Figure 7: Setup Status

- At the PSoC Designer Installation Wizard Completed screen, click a check in the applicable box if you wish to view release notes or instructions on enabling the C Compiler. When finished, click **Finish**. See Figure 8.



Figure 8: PSoC Designer Installation Wizard Completed

- At the Activate Emulator Driver screen, click Yes to restart your computer now or No to restart later.

To gracefully complete the installation process, it is recommended that you restart now. Click **Finish**. Clicking **Finish** will initiate the restart if you have selected Yes. See Figure 9.

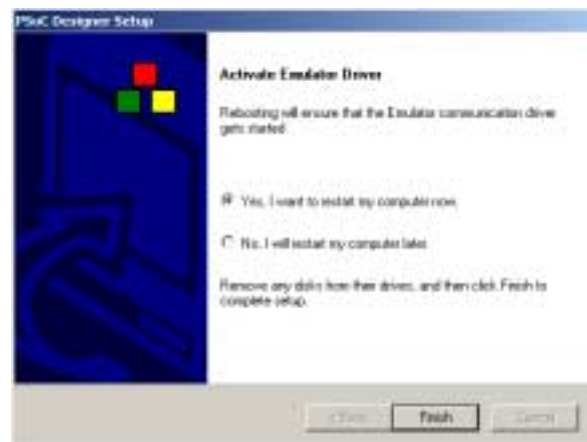


Figure 9: Activate Emulator Driver

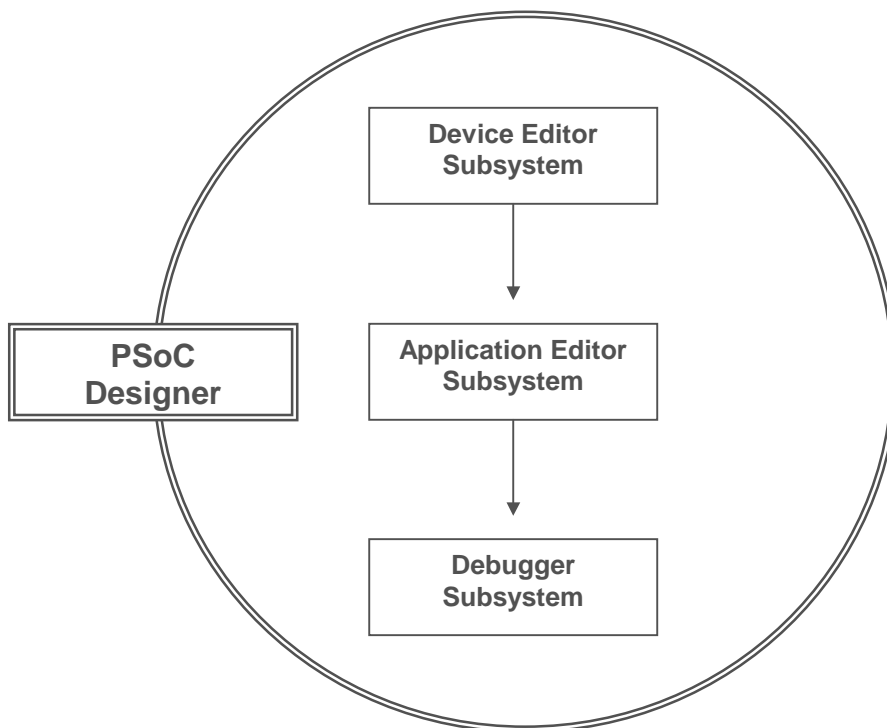
This page has intentionally been left blank.



Section 3. Using the IDE

In this section you will learn the fundamentals of the system interface.

3.1 System Diagram



3.2. File Types and Extensions

When you create a project (see **Section 4. Creating a Project**), a root directory with three folders will be generated at the location specified by you. The name of the root directory will be the project name and the names of the three folders are lib (Librarian), obj (Objects), and output (for files generated by a build).


The lib folder contains system Library Source files.

The obj folder contains intermediate files generated during the compiling/assembling of .c and assembly-source files.

The output folder contains the project .rom file (used for debugging), the listing file, and other files that contain debug information.

Upon installation and subsequent system use you will have access to the following files (contained in the folders previously described):

Type	Extension	Location	Description
A File	.a	...\lib folder under project directory	Generated when a file is archived/stored in the Library Source
AAA File	.aaa		Generated from errors during the build process
ASM File✓	.asm	Source Files\ Library Source in source tree	Editable assembly-language source file (created initially, added, or generated for APIs)
C File✓	.c	Source Files in source tree	C compiler-language file that can be added to the project
DBG File	.dbg	...\output folder under project directory	Generated during the build process. Used by Technical Support Technicians for troubleshooting
HEX File	.hex	...\output folder under project directory	Output file in Intel HEX format generated during the build process
INC C File	.h	Library Headers in source tree	Editable assembly-language include file (generated for APIs)
INC ASM File✓	.inc	Library Headers in source tree	Editable c-language include file (generated for APIs)
List File	.lis	...\obj folder under project directory	Even though this is a listing file generated by the assembler, it does not contain true address information. Check the .lst file
List File	.lst	...\output folder under project directory	Generated/updated each time you build/link a project. Includes how project is mapped to memory values as well as a list of errors/warnings and labels. This file is solely for reference
MP File	.mp	...\output folder under project directory	Generated during the build process. Identifies global symbol addresses and other attributes of output
O File	.o	...\obj folder under project directory	Intermediate, relocatable object file generated during compilation
TPL File✓	.tpl	Installation directory under ...\ Templates then copied to project directory	Template files used to generate project files (<i>boot.tpl</i> >> <i>boot.asm</i> and <i>ProjectInc.tpl</i> >> ...\ lib\ <i>projectName_GlobalParams.inc</i>)
ROM File	.rom	...\output folder under project directory	Output file in raw binary image generated by device configuration, placed in <i>PSocConfig.asm</i> , and updated during the build process. This file alone will be downloaded to the ICE for project debugging
SOC File✓	.soc	Project directory	Project file accessed under File >> Open Project
Text Document	.txt	Project directory	Text document that contains system information
XML Document✓	.xml	Project directory	Device resource file

✓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 .

Most of these files are editable and appear in the left frame of the system interface inside the folder bearing the project name. See Figure 10.

The project file system (source tree) is set up identical to the standard Windows file system.

The Source Files folder contains assembly-language code and C Compiler files generated by the system and you.

The Headers and Library Headers folders contain intermediate files added by device configurations and you.

Library Source contains the project configuration .asm as well other project-specific reference files generated by device configuration.

To access and edit files simply double-click target file.

Open files appear in the main window (to the right of the source tree).

For more details regarding files and recommended usage see **Section 6. Application Editor**.

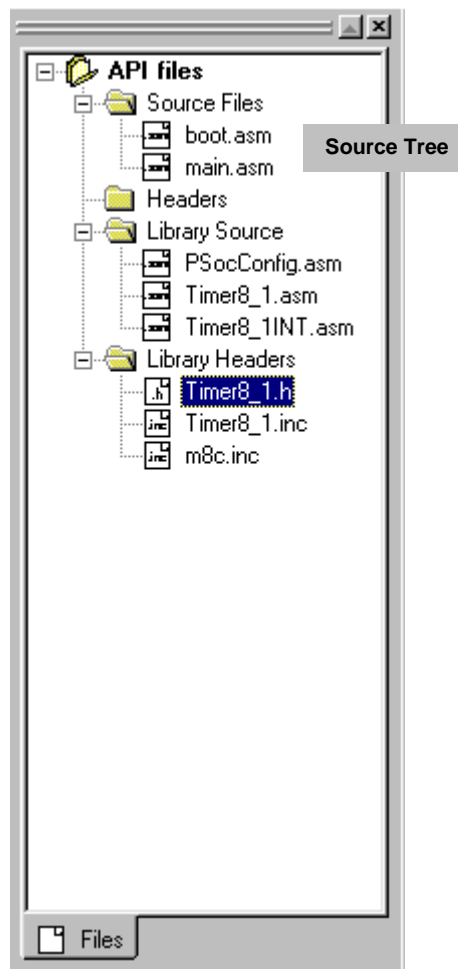


Figure 10: Source Tree



To access the project source tree any time while in any subsystem, click the **Project View** icon.

If you are viewing the source tree in the Debugger subsystem, you will see an Output tab. In the Output tab you can access the project .lst and .mp files. Because these files are generated output from your assembled and linked source, they are Read Only.

3.3. Project Manager

PSoC Designer contains three subsystems; Device Editor, Application Editor, and Debugger. The interface is split into several active windows that differ depending on which subsystem you are in.



If you are in the Device Editor subsystem, by default you will see a User Module window, a User Module placement window, a resource manager window that appears once modules have been placed, and two User Module information windows, which include data about the chosen modules. See Figure 11.

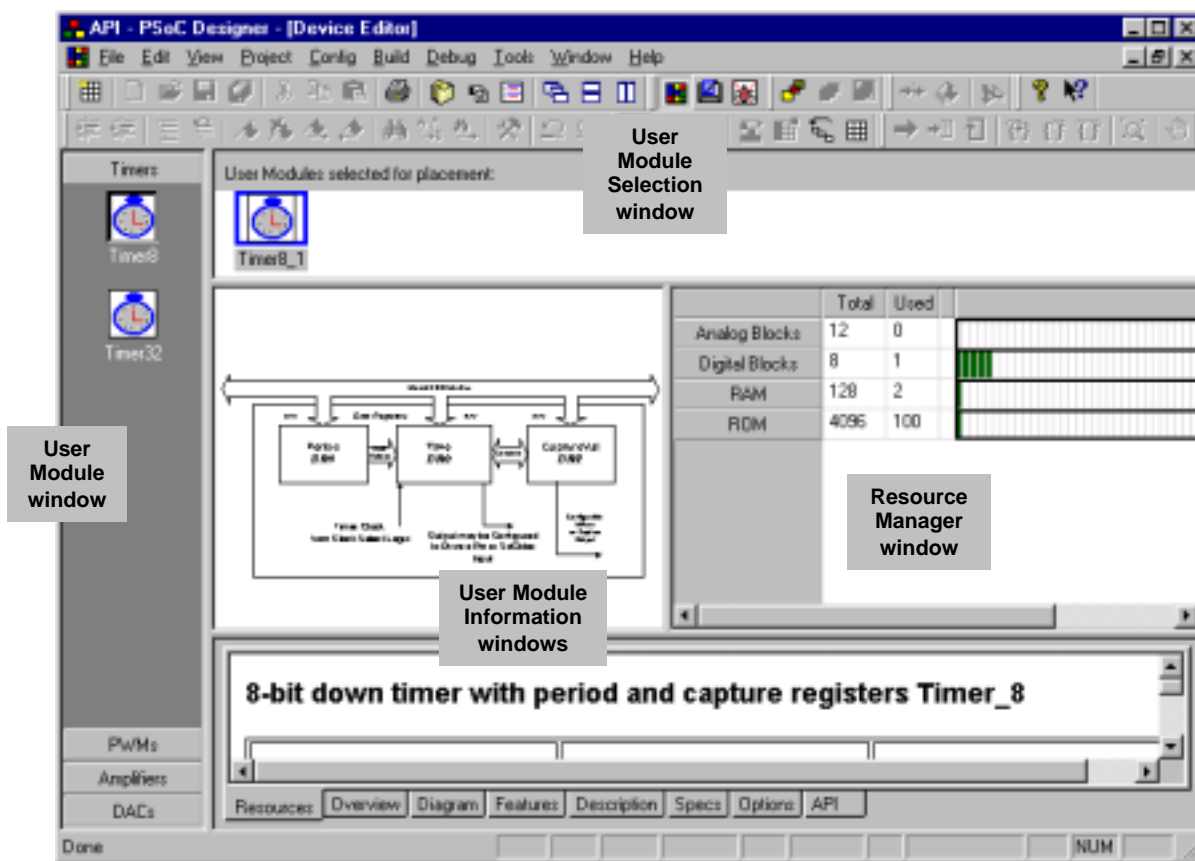


Figure 11: Device Editor Subsystem

To resize any of the windows, just hover your mouse over the dividing line until your pointer becomes a two-sided arrow then drag up or down, left or right.

The amount of information and functionality in the Device Editor subsystem is quite extensive. For further details see [Section 5. Device Editor](#).



Application Editor

If you are in the Application Editor subsystem, you will see the project files (source tree) window, the open source-file editing window, and the status window, where error messages appear if there are code problems when files are compiled and built. See Figure 12.

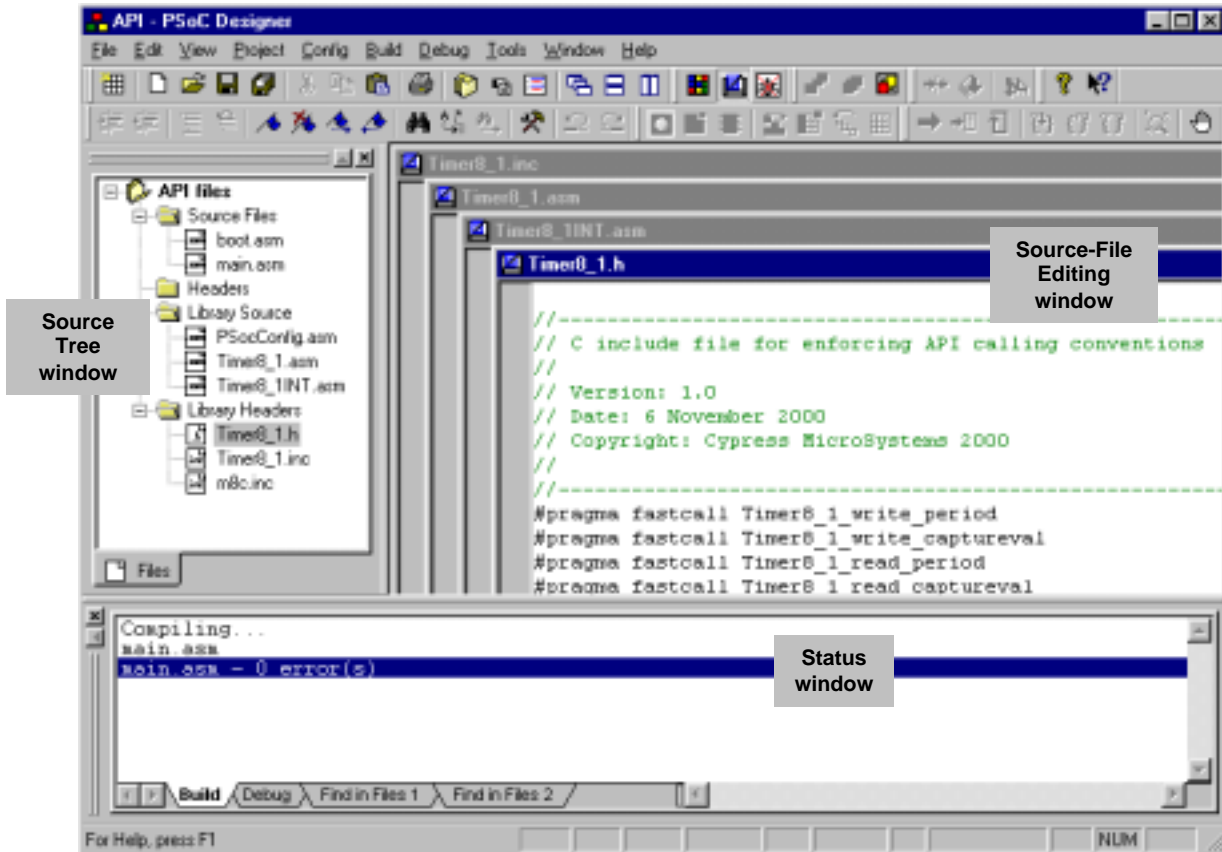


Figure 12: Application Editor Subsystem



Output View

To access the status window any time while in any subsystem, click the **Output View** icon.



Debugger

Lastly, if you are in the Debugger subsystem, you will see the same active windows as in Application Editor plus CPU register and RAM/Bank/Flash data register windows. See Figure 13.

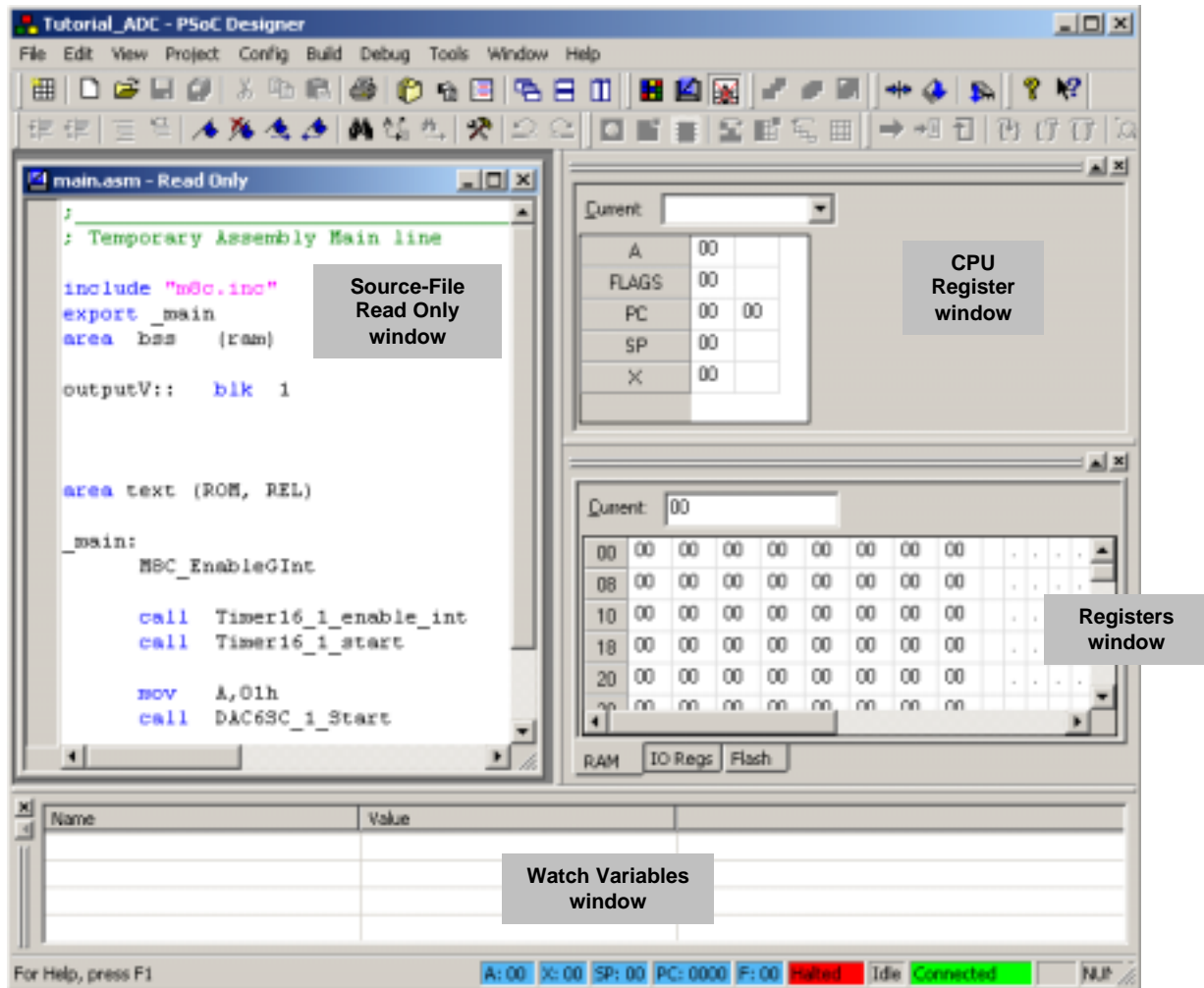


Figure 13: Debugger Subsystem

Click the “arrow” in the upper-right corner of an active window to expand or collapse the placement. Click the “x” to close. (Use the **V**iew menu to re-open closed windows.)

As you move between subsystems, you will notice different options being enabled or disabled in the toolbar and menus as applicable to functionality.

3.4. Edit Windows

As mentioned earlier, open assembly language and C Compiler source files reside in the main frame of Application Editor (see Figure 12). Each file can be opened as a separate window and tiled, cascaded, maximized and so on. See the cascaded assembly-language source files in Figure 14.

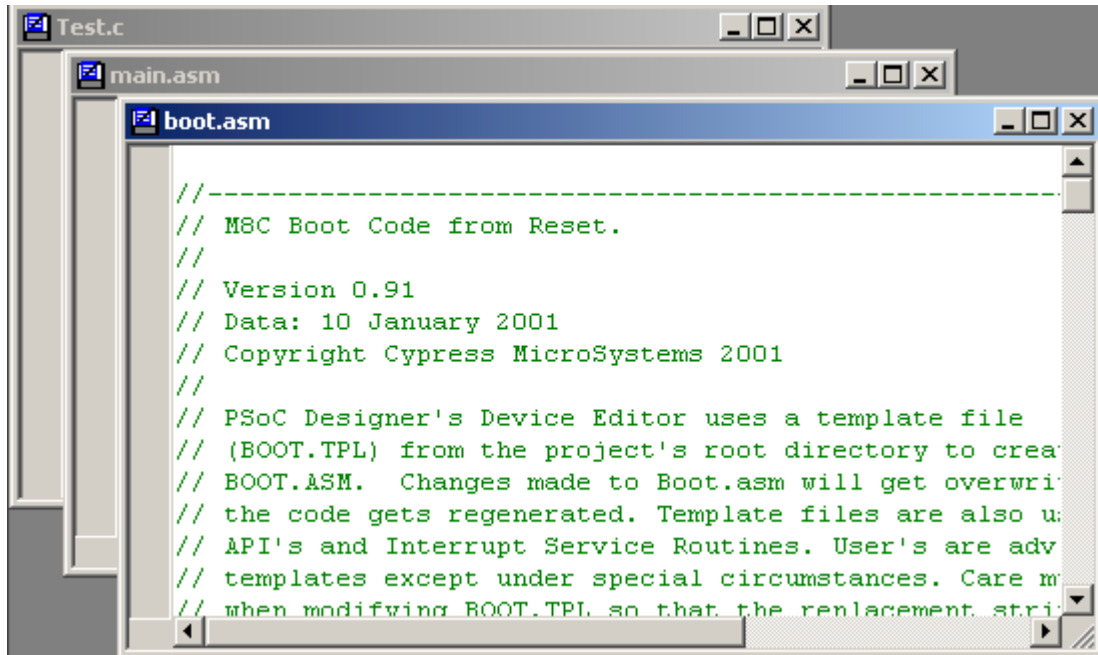
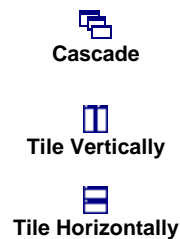


Figure 14: Cascaded Windows



You can also use icons to cascade or tile your window display.

Open files are always accessible from within other subsystems under the **Window** menu but are displayed by default in Application Editor. This is where you add and edit the assembly language and C Compiler source files of your project. **See Section 6. Application Editor** for further details on this subsystem.

3.5. Status Window

The status (or error-tracking) window of Application Editor is where the status of file compiling/assembling and project building resides. See Figure 15.

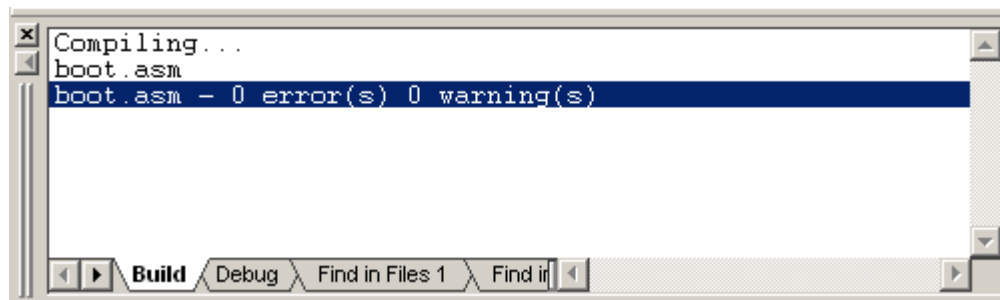


Figure 15: Status Window

Each time you compile/assemble files or build the project, the status window is cleared and the current status entered as the process occurs.

When compiling or building is complete, you will see the number of errors. Zero errors signify that the compilation/assembly or build was successful. One or more errors indicate problems with one or more files. Such errors include *missing input data* and *undeclared identifier*. For a list of all identified compile and build errors with solutions see **Section 8. Compile/Assemble Error Messages in *PSoC Designer: Assembly Language User Guide***. For further details on compiling and building see [Section 8. Builder](#) in this user guide.

Section 4. Creating a Project

In this section you will learn how to create a project.

4.1. Create a Project

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



1. To access the New Project dialog box you can either click the **New Project** icon or select Start new project from the Start dialog box upon system entry. See Figure 16.

2. Once inside the New Project dialog box, click once on a Configuration method, type a Project name, and either type or **Browse** to designate a project directory. See Figure 17.

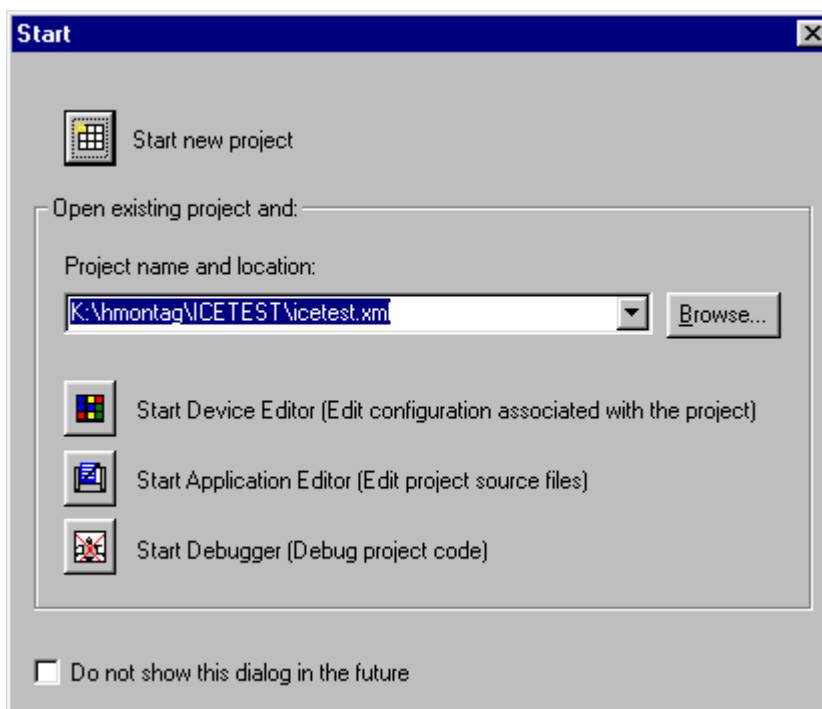


Figure 16: Start Dialog Box

For definitions of each Configuration method option (Create a New Configuration, Clone a Configuration, Select from Configuration Catalog, Start Configuration Wizard), jump ahead in this section to **Configuration Methods**.

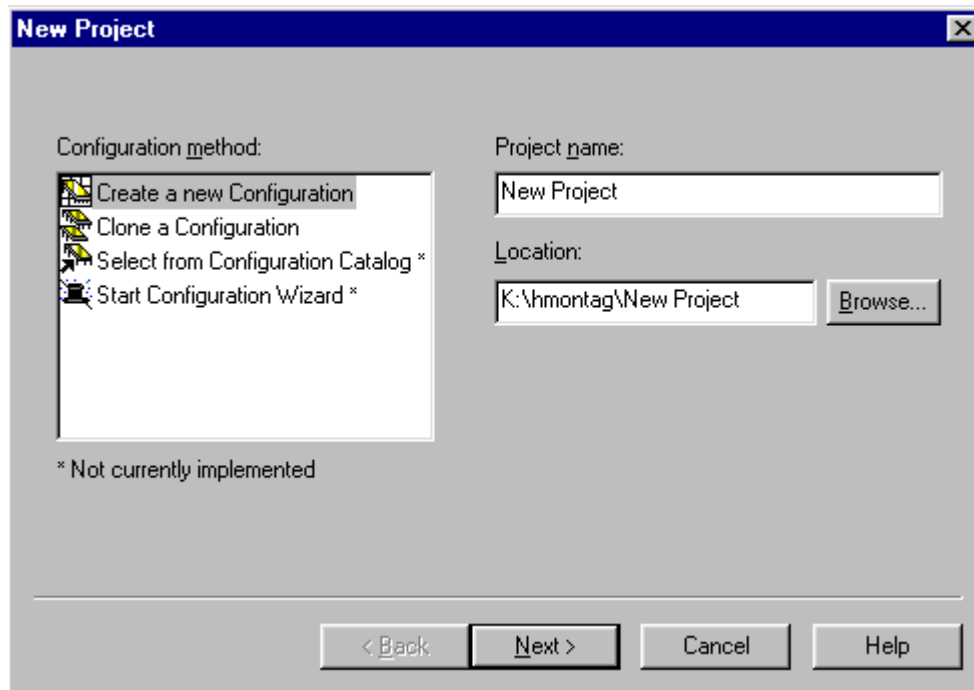


Figure 17: New Project Dialog Box

3. When finished, click **Next**.

At any time you can click **Back** to return to the previous dialog box, **Cancel** to cancel operation, or **Help** to view context-sensitive help.

Once you click **Next**, you will see the New Configuration dialog box. See Figure 18.

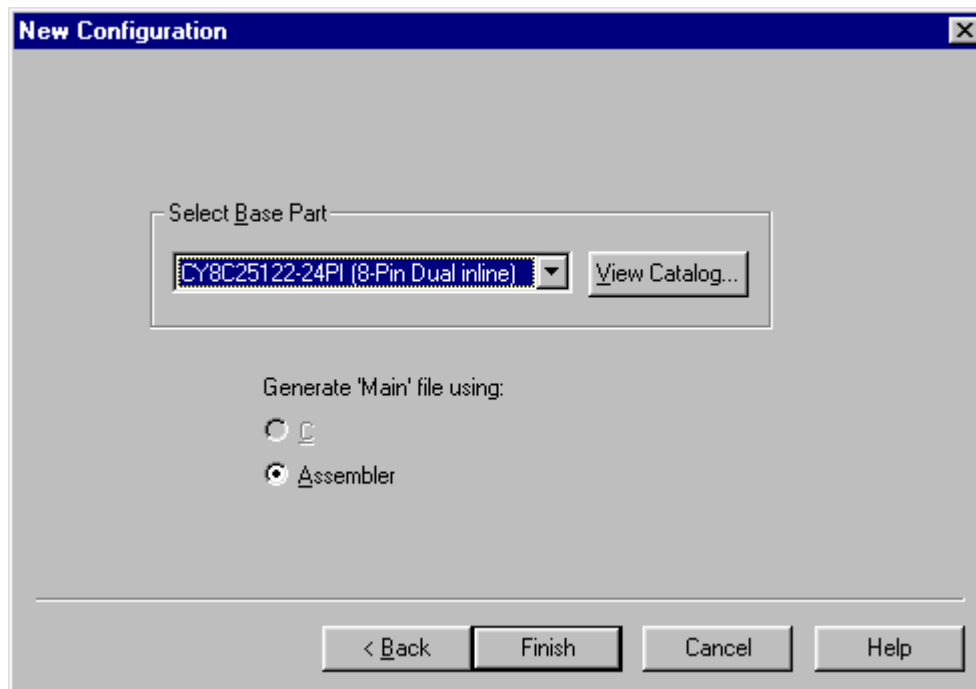


Figure 18: New Configuration Dialog Box

- Here, click the drop-arrow in the Select Base Part field and select a part. Click **View Catalog** to access a detailed list of available parts. Highlight your part of choice and click **Select** to save your selection and exit the dialog box. See Figure 19.

	Analog Blocks	Digital Blocks	IO Pin Count	Package Type	RAM	ROM
CY8C25122-24PI	12	8	6	8-Pin Dual inline	128	4096
CY8C26233-24PI	12	8	16	20-Pin Dual inline	256	8192
CY8C26443-24PI	12	8	24	28-Pin Dual inline	256	16384
CY8C26643-24AI	12	8	40	44-TQFP	256	16384
CY8C26643-24PI	12	8	44	48-Pin Dual inline	256	16384

Figure 19: Parts Catalog Dialog Box

- Once you have selected a part, click C or Assembler to designate the source in which you want the system to generate the “main” file. (Note that C will only be an option if the C Compiler has been enabled in your version of PSoC Designer. See *PSoC Designer: C Language Compiler User Guide* for enabling instructions.)
- Click **Finish**.

After clicking **Finish**, your project directory with folders will be created and can be seen in the source tree (left frame) of the Application Editor subsystem.

4.2. Configuration Methods

Following, is a definition for each configuration method to help you decide the best option for your project:

4.2.1. Create a New Configuration

Creating a new configuration is described in [4.1 Create a Project](#) earlier in this section. PSoC Designer will provide a “blank” configuration for which you select a part to be configured. Once you create your project, select your part, and click **Finish**, you are taken directly to the Device Editor subsystem where you choose and configure User Modules then generate the “new” device configuration. See [Section 5. Device Editor](#) for details on device configuration.

4.2.2. Clone a Configuration

You can clone an existing project/configuration at any point of its existence; upon creation, or before, during, or after device configuration, assembly-source programming, or project debugging.

To clone an existing project, execute the following steps:

1. Start as if you are creating a new project but in the New Project dialog box, click Clone a Configuration in the Configuration method field (refer back to Figure 17). Type a Project name and either type or **Browse** to designate a project directory.
2. Click **Next**.
3. Once you click **Next**, you will be asked if you wish to create a new directory for the cloned project with its new name. Click **Yes**.
4. In the Existing Configuration dialog box, **Browse** (or type) to identify the existing directory of the project you wish to clone.

If you wish to specify an alternative part (device), do so in the Select Base Part drop-down.

Finally, select the subsystem in which you would like to begin; Device Editor, Application Editor, or Debugger. See Figure 20.

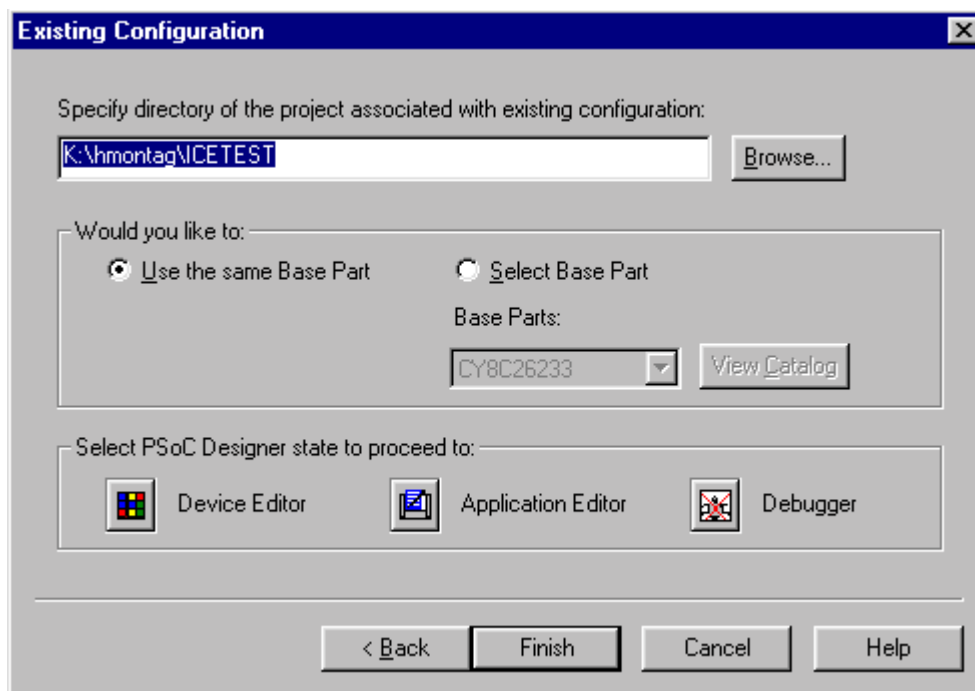


Figure 20: Existing Configuration Dialog Box

5. When finished, click **Finish**.

After clicking **Finish**, your new project directory will be created (from the existing project you chose) and can be seen in the source tree (left frame) of the Application Editor subsystem.

If you wish to move an existing project from one directory to another, use the cloning method to create a new “cloned” project in the new directory (rather than employing a physical move). This is to ensure that all project source links remain in tact.

4.2.3. Select from Configuration Catalog (Not Yet Implemented)

4.2.4. Start Configuration Wizard (Not Yet Implemented)

This page has intentionally been left blank.

Section 5. Device Editor

In this section you will learn how to select applicable User Modules, configure and place User Modules on PSoC blocks, make interconnections, set pin-outs, track usage of resources, and generate application files.

5.1. Selecting User Modules

Selecting applicable User Modules is the first step (after creating a project) to configuring your target 8C2xxxx device. A User Module, as defined in PSoC Designer, is an accessible, pre-configured function that once placed and programmed will work as a peripheral on the target device.



To access Device Editor, click the **Device Editor** icon.

By default, you will be in the Select User Module mode of the subsystem (see Figure 11). See the Device Editor toolbar, Figure 21.



Figure 21: Select User Module in Device Editor Toolbar

In the left frame you will see options of User Modules. See Figure 22.

To view the individual User Modules, click one of the set titles (i.e., Timers, Counters, PWMs, etc.) and scroll to see pre-configured options.

In the other active windows of Device Editor you can view configuration data related to an individual User Module. To view device/module data, single-click a User Module from within a set title. See Figure 23 of data related to a 16-bit PWM (Pulse Width Modulator).



Figure 22: User Module Options

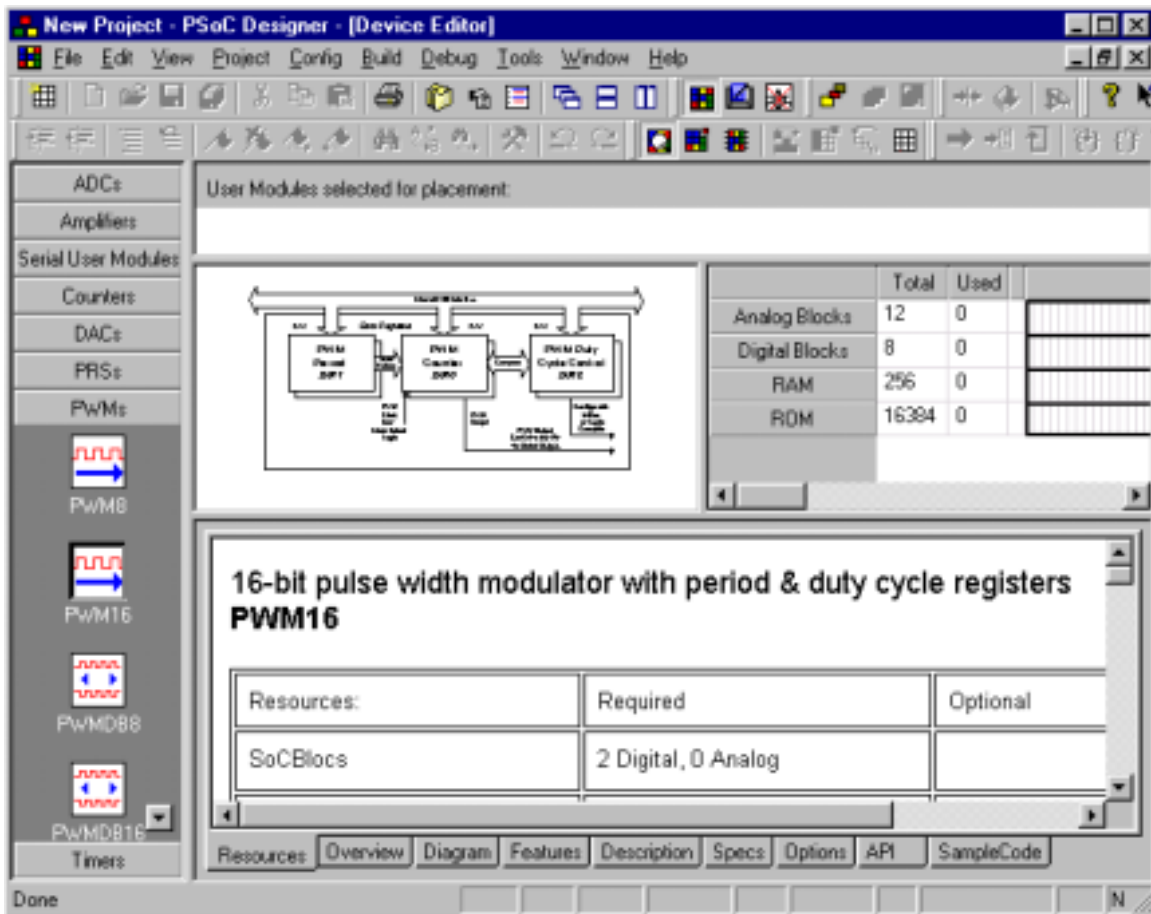


Figure 23: User Module Data

In the lower active window, click the different tab options (Resources, Overview, Diagram, Features, etc.) to view additional information regarding a chosen User Module.

Once you have viewed and decided upon User Modules, you are ready to officially select them. To select a User Module, execute the following steps:

1. Choose a User Module from the left frame.
2. Double-click it. It will then appear in the upper active window.
3. Repeat the process for each individual User Module you wish to select.

For each User Module you add, the system updates the data in the Resource Manager window with the number of occupied PSoC blocks, along with RAM and ROM usage used by the current set of “selected” User Modules. If you attempt to select a User Module that requires more resources than are currently available, PSoC Designer will not allow the selection.

- View your selections in the upper active window. See Figure 24.

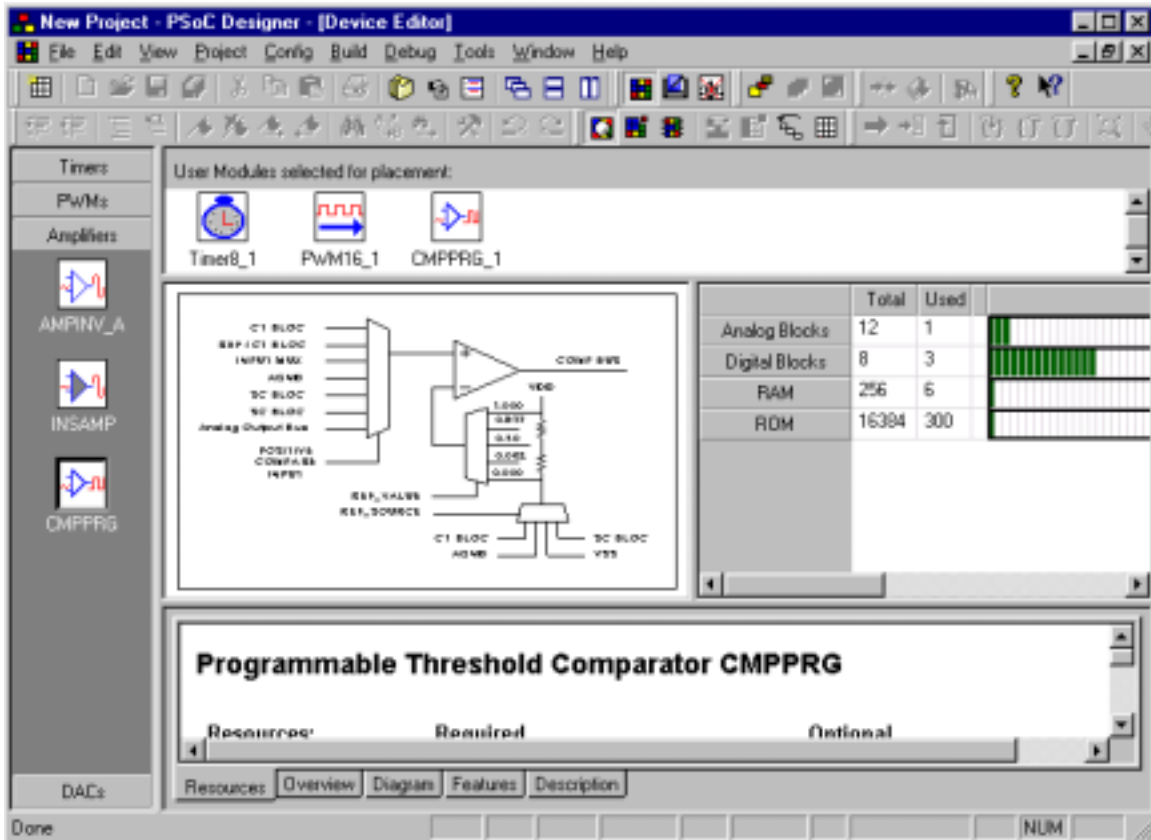
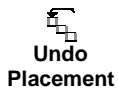


Figure 24: User Module Selections

At any time during device configuration you can add and remove User Modules to and from your device.



To remove User Modules from your collection (undo placement), click on the User Module that you wish to remove and click the **Undo Placement** icon. This will not remove User Modules from PSoC Designer, just from your collection.

5.2. Placing User Modules

Placing selected User Modules on PSoC blocks is the second step to configuring your target 8C2xxx device. PSoC blocks, as defined in PSoC Designer, are the analog and digital peripheral blocks of a device that are customized by the placement and configuration of User Modules.

To access Place User Module mode, click the **Place User Module** icon in the Device Editor toolbar. See Figure 25.



Figure 25: Place User Module in Device Editor Toolbar

In the left frame you will see User Module Parameters and Global Resources. In the upper window see your selected User Modules. In the main window see the analog and digital PSoC blocks. See Figure 26.

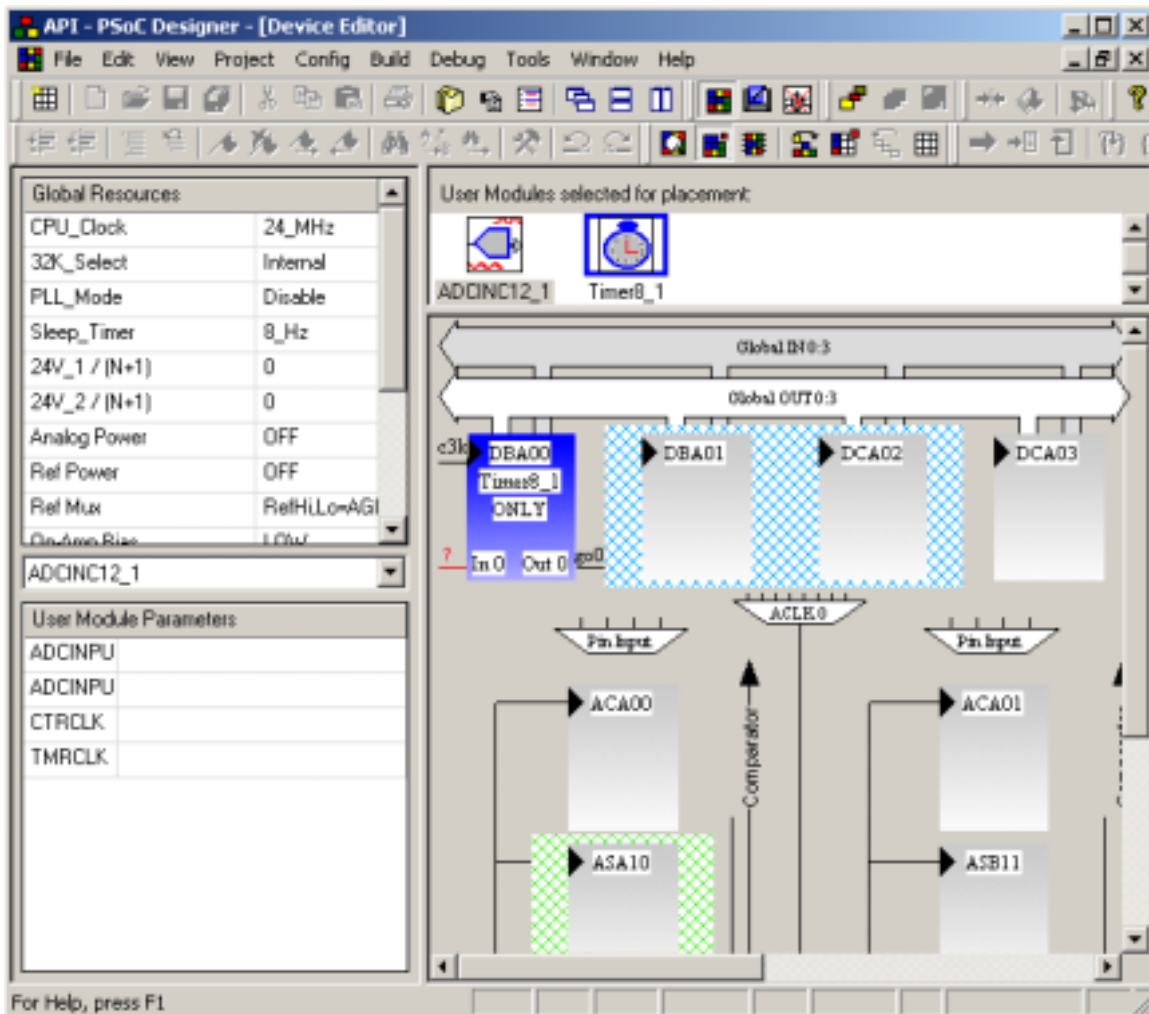


Figure 26: User Module on PSoC Block

5.2.1. Placing a User Module

To place a User Module execute the following steps:

1. Single-click on a selected User Module.

When you click the module, the first available location on the device is highlighted. If the User Module consists of more than one group of PSoC blocks, then the groups will be highlighted in green (active) or blue (inactive).



Click the **Next Position** icon to advance the highlights to the next available location (identified with green cross-hatch background). Do this until you have identified the exact location for the User Module.

Note that if your User Module occupies a combination of blocks (both digital and analog), the active blocks (green cross-hatch) will advance as you click the **Next Position** icon and the inactive blocks (blue cross-hatch) will remain static. Currently, the ADCINC12_1 User Module is the only compound module in PSoC Designer.

2. When you have identified the location, click the Place User Module icon or right-click and select Place.

Once you have placed the module, it will appear on the device, color-coded, bearing the designated name on the chosen PSoC block.

Input connection parameters associated with the block will appear in the lower-left corner of the block, Output connection parameters will appear in the lower-right corner, and Clock connection parameters will appear (once set) by the triangle in the upper-left corner. See Figure 27.

If, at any time, you would like to name or rename User Modules, right-click on the module, select Rename, and type a new name.

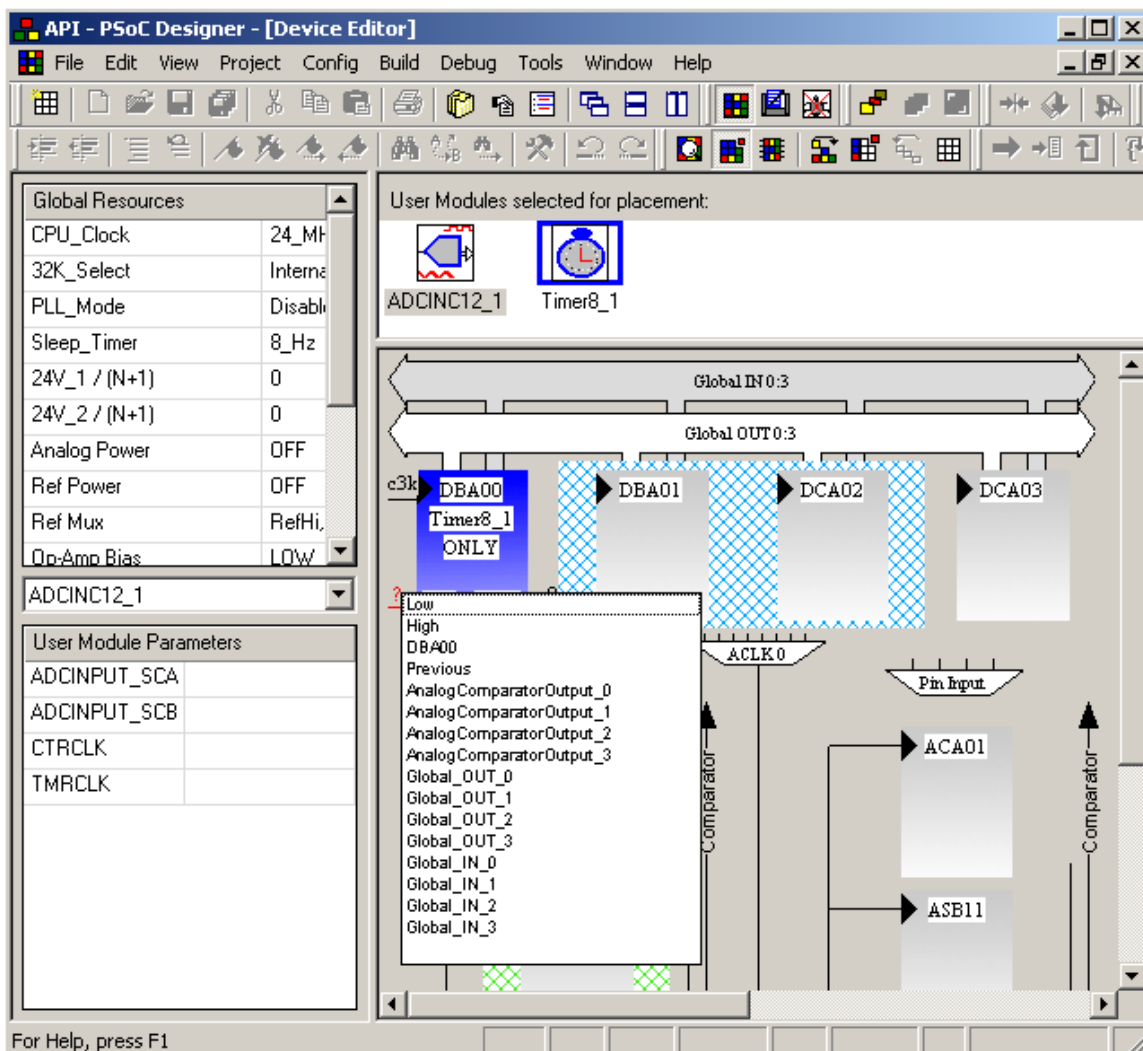


Figure 27: Right-Click Parameters

3. Repeat this process for all selected User Modules.



Undo Placement

To remove User Modules from your collection (undo placement), click on the User Module that you wish to remove and click the **Undo Placement** icon. This will not remove User Modules from PSoC Designer, just from your collection.



Clear All

To clear all User Modules from your collection, click the **Clear All Placements** icon.

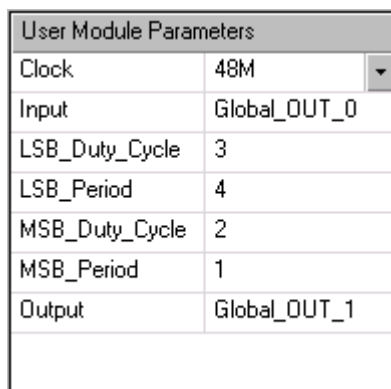
If you add or remove User Modules after you have generated application files, you will need to re-generate the application files (as well as reconfigure required settings). For further details, see **Generating Application Files** later in this section.

5.2.2. User Module Parameters

As you single-click a selected User Module you can view its parameters under User Module Parameters.

To view all settings for a selected User Module, either single-click on the module itself or use the numbered drop-down list in the upper-left corner of the left frame.

Once you place that User Module, the parameters will be updated with applicable module names. You can now make selections from the updated drop-down lists as to configuration for the analog or digital PSoC block. See Figure 28.



User Module Parameters	
Clock	48M
Input	Global_OUT_0
LSB_Duty_Cycle	3
LSB_Period	4
MSB_Duty_Cycle	2
MSB_Period	1
Output	Global_OUT_1

Figure 28: User Module Parameters

1. To update all User Module parameters click each arrow-option and make applicable selections.

You can also set parameters by left-clicking active areas on the block. To set Input parameters hover your mouse over the lower-left corner of the block until you see a superficial chip, then left-click and make your selection. Repeat this action in the lower-right corner for Output parameters and on the triangle in the upper-left corner for Clock parameters. These settings will immediately appear in User Module Parameters (in the left frame of the system).

Some parameters can only be set in User Module Parameters (instead of the right-click method) because they are specified integer values. You set these values by clicking the up/down arrows or double-clicking the value and typing over. If you enter a value that is out of range, you will see a dialog box specifying the acceptable range. (Click **OK** to close the dialog box.)

2. Repeat this process for all placed User Modules.

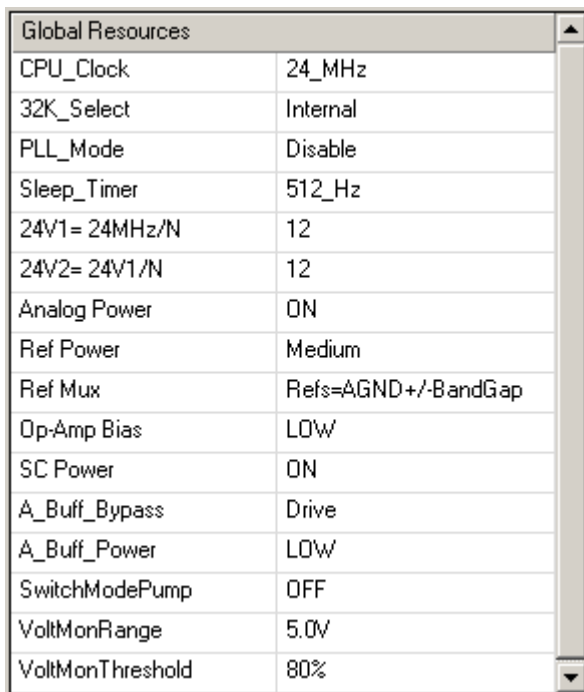
5.2.3. Global Resources

Global Resources are hardware settings that determine the underlying operation of the part (for the entire application). Such settings include the CPU_Clock. For example, this setting designates the speed in which the M8C processes. High MHz equal fast processing and low MHz equal slower processing. High takes more power, low less power. Therefore, when you set the value, you must strike a balance between speed and power to perfectly achieve your objective.

1. To update global resources of all User Modules (collectively) click each arrow-option and make applicable selections.

Similar to User Module Parameters, some parameters in Global Resources are specified integer values (such as 24V_1 and 24V_2). You set these values by clicking the up/down arrows or double-clicking the value and typing over. If you enter a value that is out of range, you will see a dialog box specifying the acceptable range. (Click **OK** to close the dialog box.)

See Figure 29.



Global Resources	
CPU_Clock	24_MHz
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
24V1= 24MHz/N	12
24V2= 24V1/N	12
Analog Power	ON
Ref Power	Medium
Ref Mux	Refs=AGND+/-BandGap
Op-Amp Bias	LOW
SC Power	ON
A_Buff_Bypass	Drive
A_Buff_Power	LOW
SwitchModePump	OFF
VoltMonRange	5.0V
VoltMonThreshold	80%

Figure 29: Global Resources

5.3. Deploying Interconnectivity

Specifying interconnections between the User Modules on the PSoC blocks can be done as you place each User Module (as previously discussed), or after you place each User Module. Interconnectivity between User Modules enables communication between PSoC blocks (which are the analog and digital peripheral blocks of a device that are customized by the placement and configuration of User Modules).

Interconnections can be specified on the device in Place User Module mode of Device Editor. To access the Place User Module mode, click the **Place User Module** icon in the Device Editor toolbar. Refer back to Figure 25. (If you have just placed your User Modules, you should already be in this subsystem.)

You can set interconnectivity parameters by left-clicking active areas on the PSoC block. Hover your mouse over random sites on the block until you see a superficial chip, then left-click and make your valid selection. Repeat this action in, on, and around all occupied blocks. See Figure 30. Some of these settings will immediately appear in the associated area (i.e., User Module Parameters and Global Resources), where you can also make these specifications.

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. Multiplexors may also be configured to route signals throughout the PSoC block architecture.

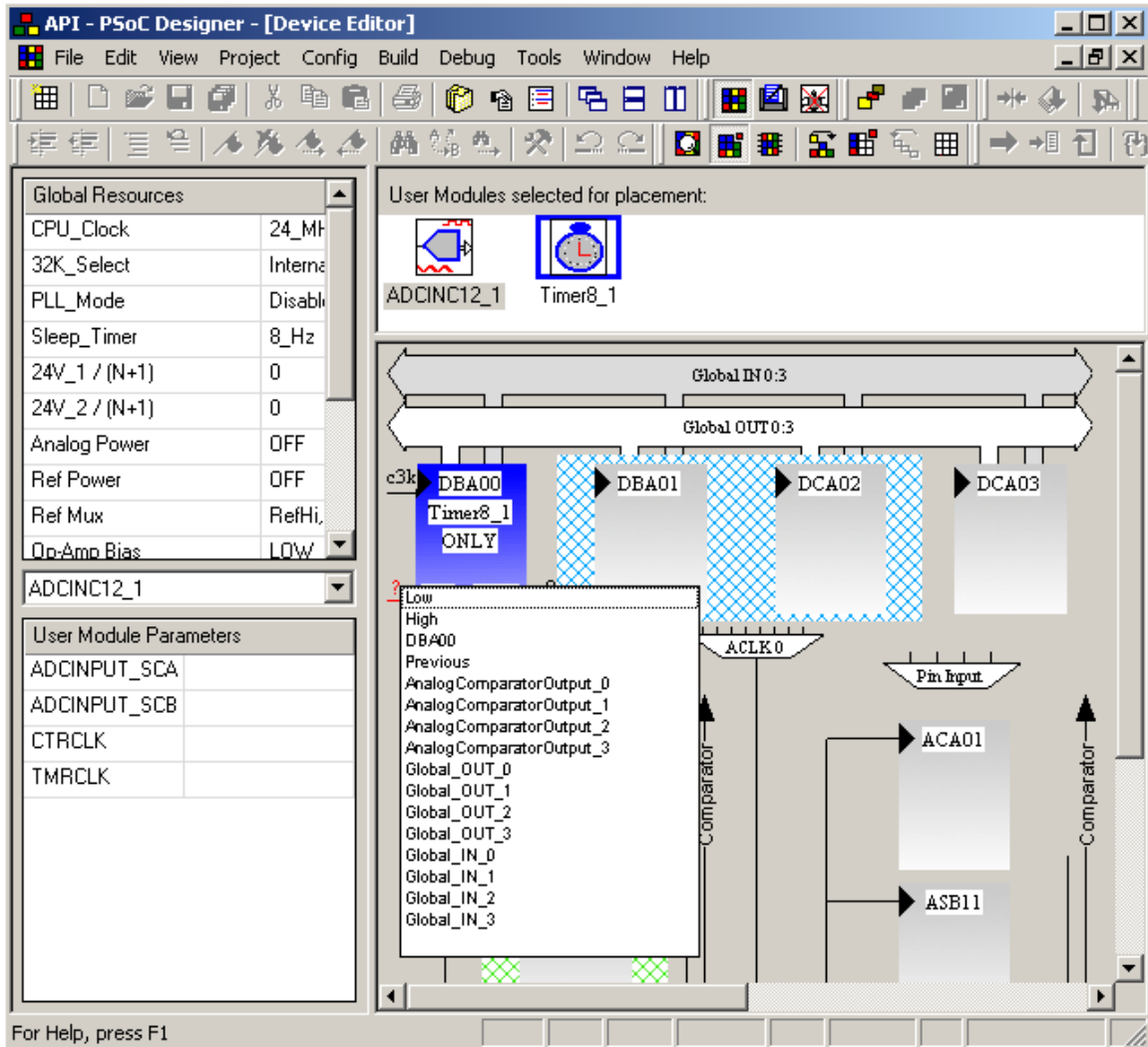


Figure 30: Interconnectivity Parameters

To save current configurations in Device Editor, click **File >> Save Project**.

5.4 Specifying Pin-out

Specifying the pin-out for each PSoC block is the fourth step to configuring your target 8C2xxxx device. When you specify a PSoC block to a pin-out you are making a physical connection between the software configuration and the hardware (M8C device). Also, at this time, you can specify the interconnectivity between PSoC blocks (i.e., input mux, analog clocking mux, and output buffers). These configurations will later be emulated and debugged within the device simulation unit (In-Circuit Emulator). See [Section 9. Debugger](#).

To access the Specify Pin-out mode of the subsystem, click the **Specify Pin-out** icon in the Device Editor toolbar. See Figure 31.



Figure 31: Specify Pin-Out in Device Editor Toolbar

In the middle frame you will see the pin-out bearing the number of pins of your target device. The 8C2xxxx family of devices consists of five pin-outs; 8, 20, 28, 44, and 48. Your specific device was chosen when you specified a Configuration method during the creation of your project. See Figure 32.

5.5. Tracking Device Space

Tracking the available space and memory of configurations for your device is something you do intermittently during the whole process of configuring your target 8C2xxxx device. Device space and memory resources need to be monitored so you are aware, on an ongoing basis, of the capacity and limitations you are working with on the M8C.

You can monitor device space and memory from within the Select User Module mode of Device Editor or by accessing View >> Output. Refer back to Figure 21. Resources are updated as each User Module is selected.

In the far-right frame of the Select User Module mode of the subsystem, you see a table to track Analog Blocks, Digital Blocks, RAM, and ROM. As you place User Modules, you can view how many analog and digital PSoC blocks you have available and how many you have used. You will also notice a live graph tracking the PSoC blocks you have used by percentage.

RAM and ROM monitors track the amount RAM and ROM required to employ each selected User Module.

See Figure 33.



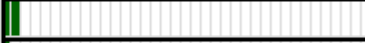

	Total	Used	
Analog Blocks	12	4	
Digital Blocks	8	6	
RAM	256	12	
ROM	16384	600	

Figure 33: PSoC Block Resources

5.6. Generating Application Files

Generating application files is the final step to configuring your target 8C2xxxx device. When you generate application files, PSoC Designer takes all device configurations and updates existing assembly-source and C compiler code (including the project library source `PsocConfig.asm`) and generates API and ISR shells. Read ahead in this section for details regarding APIs and ISRs. At this time, the system also creates a data sheet based on your part configurations that can be accessed in the Device Editor.

Once this process is complete, you can enter Application Editor and begin programming the desired functionality into your (now configured) device. For further details regarding programming, see [Section 6. Application Editor](#) and [Section 7. Assembler](#).

You can generate application files from within any of the three Device Editor modes; Select User Module, Place User Module, or Specify Pin-out.



To generate application files, click the **Generate Configuration** icon. This process is transparent to you and takes less than a minute.

Once the process is complete, a graphic dialog box will appear informing you that the application code has been generated successfully. Now, click the subsystem to where you would like to go next. Again, Application Editor is the place to begin source programming. See Figure 34.

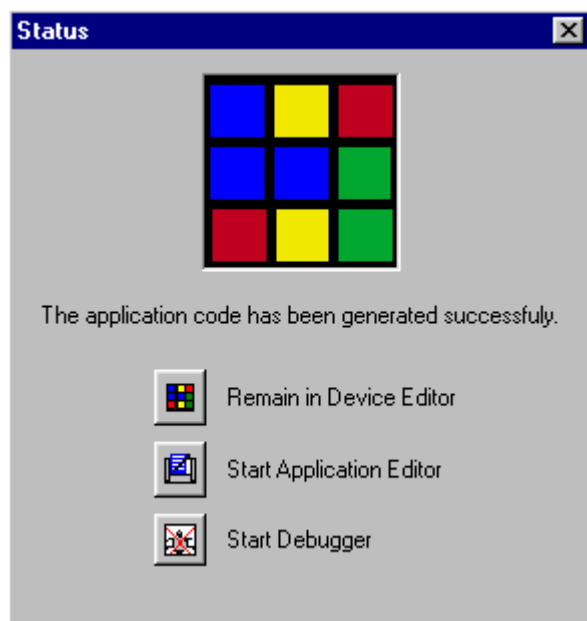
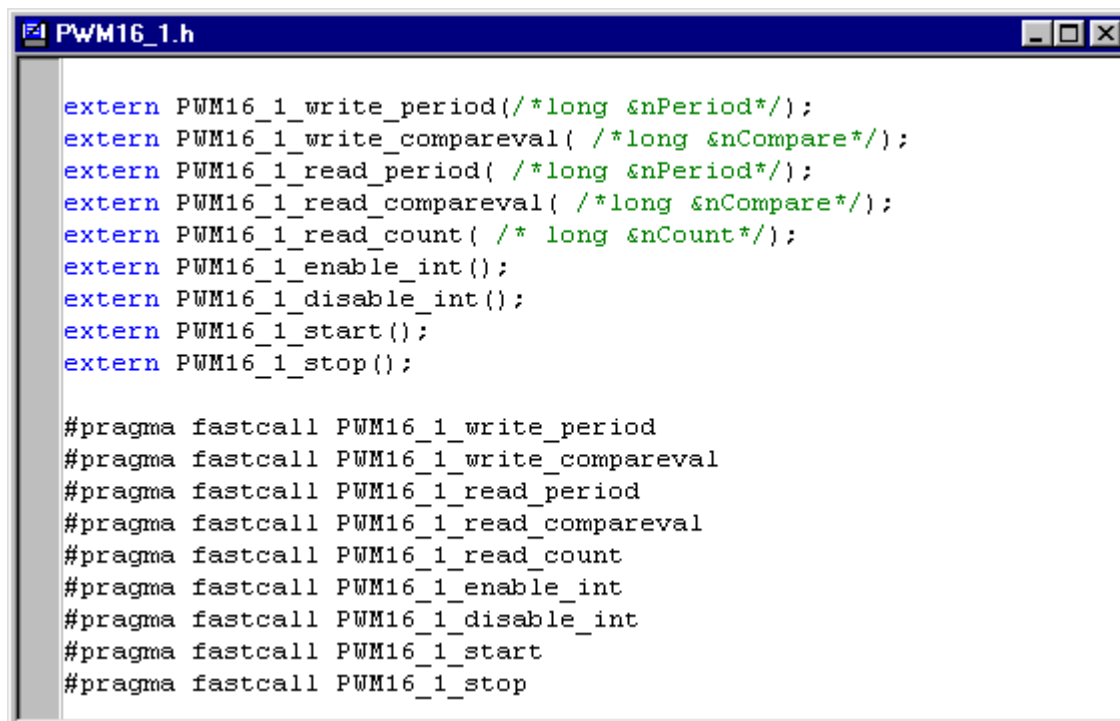


Figure 34: Application Generation Status

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

5.6.1. 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 shells provide the device-interface and interrupt-activity framework for source programming.* See the following example of an .h file for configurations of a 16-bit PWM (Pulse Width Modulator) created during application-code generation:



```
extern PWM16_1_write_period(/*long &nPeriod*/);
extern PWM16_1_write_compareval( /*long &nCompare*/);
extern PWM16_1_read_period( /*long &nPeriod*/);
extern PWM16_1_read_compareval( /*long &nCompare*/);
extern PWM16_1_read_count( /* long &nCount*/);
extern PWM16_1_enable_int();
extern PWM16_1_disable_int();
extern PWM16_1_start();
extern PWM16_1_stop();

#pragma fastcall PWM16_1_write_period
#pragma fastcall PWM16_1_write_compareval
#pragma fastcall PWM16_1_read_period
#pragma fastcall PWM16_1_read_compareval
#pragma fastcall PWM16_1_read_count
#pragma fastcall PWM16_1_enable_int
#pragma fastcall PWM16_1_disable_int
#pragma fastcall PWM16_1_start
#pragma fastcall PWM16_1_stop
```

Figure 35: PWM16_1.h File

Once you have generated your device configuration application code, the files for APIs and ISRs can be found in the source tree of Application Editor under the Library Source and Library Header folders.

This useful feature occurs each time device application code is generated and is transparent to the user. If you modify any API/ISR file and then re-generate your application files, your changes **will not** be overwritten *unless you have renamed a User Module.*

5.6.2. Working with ISRs

There are two types of interrupt vectors: (1) fixed function and (2) configurable PSoC blocks. The fixed function interrupts are:

- Reset
- Supply Monitor
- GPIO
- Sleep Timer

The configurable PSoC block interrupts include the eight (8) digital blocks and four (4) analog column blocks. The definition (e.g. interrupt vector action) of a configurable PSoC block interrupt depends on the User Module that occupies that block.

The Device Editor handles the details of getting User Module parameters into source code so that it will be configured correctly upon startup and expose subroutines that make it easy to 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 Device Editor also inserts a call or jump to the User Module's interrupt handler in the startup source file, *boot.asm*.

Interrupts Vectors and the Device Editor

Following is an example of how an interrupt handler is dispatched in the startup code. Shown below is Timer32 User Module mapped to PSoC blocks 0, 1, 2, and 3. 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.

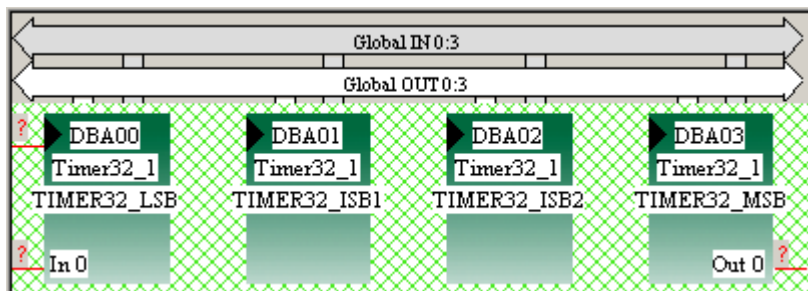


Figure 36: Timer32 on Four Digital PSoC Blocks

Upon device application generation (🏠) code is produced for the Timer32_1 User Module in *boot.asm*. The startup code is also altered with the addition of the call to the timer interrupt handler. See the following figure:

```
org 0
jmp __start
jmp Interrupt1
jmp Interrupt2
jmp Interrupt3
jmp Interrupt4
jmp Interrupt5
jmp Interrupt6
jmp Interrupt7
jmp Interrupt8
jmp Interrupt9
jmp Interrupt10
jmp Interrupt11
jmp Interrupt12
jmp Interrupt13
jmp Interrupt14
jmp Interrupt15

Interrupt1:
    // call    void_handler
    reti
Interrupt2:
    // call    void_handler
    reti
Interrupt3:
    // call    void_handler
    reti
Interrupt4:
    // call    void_handler
    reti
Interrupt5:
    lcall    Timer32_1INT
    reti
```

Figure 37: 32-Bit Timer Interrupt Hook

The following table shows how the *boot.asm* vector names map to fixed and PSoC block (configurable) interrupts:

<i>boot.asm</i> Interrupt Name	Data Sheet Interrupt Name	Type
__start	Reset	Fixed
Interrupt1	Supply Monitor	Fixed
Interrupt2	DBA00	PSoC Block
Interrupt3	DBA01	PSoC Block
Interrupt4	DBA02	PSoC Block
Interrupt5	DBA03	PSoC Block
Interrupt6	DCA04	PSoC Block
Interrupt7	DCA05	PSoC Block
Interrupt8	DCA06	PSoC Block
Interrupt9	DCA07	PSoC Block
Interrupt10	Analog Column 0	PSoC Block
Interrupt11	Analog Column 1	PSoC Block
Interrupt12	Analog Column 2	PSoC Block
Interrupt13	Analog Column 3	PSoC Block
Interrupt14	GPIO	Fixed
Interrupt15	Sleep Timer	Fixed

From our example, Interrupt5 corresponds to DBA03, which is also labeled in the Device Editor shown in Figure 36. There are no interrupt handlers at DBA00, DBA01, and DBA02 (Interrupt2, Interrupt3, and Interrupt4) 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.

Circumventing the Device Editor’s Interrupt Code/Vector Generation

You may find it necessary to modify the behavior of an interrupt following the vector action. For example, you may prefer to use a long jump (LJMP) instead of a long call (LCALL) to the interrupt handler. This can be done by hard coding the changes to the *boot.tpl* file in the ...*PSoC Designer**Templates* folder. *boot.tpl* is the template that the Device Editor uses to create *boot.asm* each time device application generation is executed.

5.6.3. A Word About *boot.asm*

When device configuration application files are generated, *boot.asm* is updated. Among other things, this file includes a jump table for interrupt handlers. (Additional details regarding this file are up ahead in section [6.1. File Definitions and Recommendations](#).)

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, etc.

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. Following are all 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 will be 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 you install an interrupt handler and make changes directly to *boot.asm*, the changes will not be preserved if device configuration generation is executed after the changes are made. If you make changes to *boot.asm* that you do not want overwritten, hard code the change in *boot.tpl* (template for *boot.asm*) instead.

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.

This page has intentionally been left blank.


Section 6. Application Editor

In this section you will learn definitions and recommended usage of critical files as well as how to modify files generated by PSoC Designer, add new files, and remove unwanted files.

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


6.1. File Definitions and Recommendations

Once you have finished configuring your device and generating application code, you are ready to program the desired functionality into the device. This is done in the Application Editor subsystem. Application Editor is where all source-code programming (editing and adding files) takes place.

 To access Application Editor, click the **Application Editor** icon. See Project Manager in section 3 to review subsystem navigation.

As discussed in [Section 3. Using the IDE](#), you will see the file source tree in the left frame. Refer back to Figure 10. The files you see were generated when the project was created, and updated after device configuration. See File Types and Extensions in section 3 for general facts about these files.

The following definitions of critical system files should further your system knowledge and better prepare you for carrying out your vision for the M8C:

boot.asm: This startup file resides in the source tree under Source Files and is key because it defines the boot sequence. Device Editor uses a template (*boot.tpl*) from the software installation ... \Templates directory to create *boot.asm*. Changes made to *boot.asm* will get overwritten when the code gets regenerated (application generation ). Following are components of the boot sequence:

- Originates reset vector for code that begins after the M8C is powered up.
- Holds interrupt table for code that is executed when an interrupt occurs.

Local jumps are enforced when the interrupt handler is originated (ORGed) within close proximity to the vectors. Therefore, the vector table will not fall out of alignment with (3 byte) long calls. Device configuration initialization will always occur, because this file is created (with the call to LoadConfig) for each configuration change.

- Calls device configuration initialization to enforce quick device configuration after reset.
- Creates a proper 'C' environment because 'C' code requires certain types of initializations.

The 'C' initialization code will occur even if the application is built using only assembly code.

- Calls `main` (or `_main`) to begin executing code.

boot.asm will be re-generated every time device configurations change and application files generated. This is done to ensure that interrupt handlers are consistent with the configuration. If you make changes to *boot.asm* that you do not want overwritten, hard code the change in *boot.tpl* (template for *boot.asm*) instead.

Policy dictates that this file belongs to PSoC Designer. Again, it is created from a template. Keep in mind, it is highly recommended that you *do not* modify the contents of this file.

main.asm: This file resides under Source Files and is key because it holds the “_main” label that is referenced from the boot sequence. *main.asm* is created to resolve the external reference from the boot sequence. Upon new project creation, the “_main” function contains a simple “forever” loop.

This file can be removed and replaced with a 'C' “main” if you determine that most of the application should be written in 'C' source. It is only created once, when a new project is created. No additional policies or recommendations are attached to this file.

PSocConfig.asm: This is always a required Library Source file because it contains the configuration that is loaded upon system access. Initially, and probably for a very brief moment, there is not a configuration. Therefore this file contains a function label (`LoadConfig`), to satisfy the boot-sequence reference, as well as a return.

The function to load the configuration (`LoadConfig`) is called from the boot sequence. PSoC Designer will overwrite *PSocConfig.asm* when a device configuration has changed and application files re-generated - *no exceptions*.

You must not reconfigure or modify any aspect of a device configuration if you wish to preserve changes that you have made to *PSocConfig.asm*.

You can, however, keep a copy of any changes and reapply them after a device reconfiguration. Remember, because this is a Library Source file, it is added/replaced to *libPsoc.a*.

If you wish to manipulate bits, all part register values reside in this file for your reference.




















6.2. Modifying Files

When you are ready to program and modify assembly-language source files, double-click the target file from under the source tree in the left frame. The open file will then appear in the main active window.

Open files are accessible from within other subsystems under the Window menu as Read Only but are displayed editable by default in Application Editor.

You can have as many files open as you wish (or that your computer will allow). For further details regarding open files and active windows, refer back to [Section 3. Using the IDE](#).

Following, is a description of all menu options available for modifying source files:

Icon	Option	Menu	Shortcut	Feature
	Application Editor	<u>V</u> iew >> A <u>p</u> plication Editor		Enables source tree and files for editing
	Compile/Assemble	<u>B</u> uild >> C <u>o</u> mpile/Assemble	[Ctrl] [F7]	Compiles/assembles the most prominent open, active file (.c or .asm)
	Build	<u>B</u> uild >> <u>B</u> uild	[F7]	Builds entire project and links applicable files
	New File	<u>F</u> ile >> <u>N</u> ew	[Ctrl] [N]	Adds a new file to the project
	Open File	<u>F</u> ile >> <u>O</u> pen	[Ctrl] [O]	Opens an existing file in the project
	Indent			Indents specified text
	Outdent			Outdents specified text
	Comment			Comments selected text
	Uncomment			Uncomments selected text
	Toggle Bookmark			Toggles the bookmark: Sets/removes user-defined bookmarks used to navigate source files
	Clear Bookmark			Clears all user-defined bookmarks
	Next Bookmark			Goes to next bookmark
	Previous Bookmark			Goes to previous bookmark
	Find Text	<u>E</u> dit >> <u>F</u> ind	[Ctrl] [F]	Find specified text
	Replace Text	<u>E</u> dit >> <u>R</u> eplace	[Ctrl] [H]	Replace specified text
	Repeat Replace			Repeats last replace
	Set Editor Options			Set options for editor
	Undo	<u>E</u> dit >> <u>U</u> ndo	[Ctrl] [Z]	Undo last action
	Redo	<u>E</u> dit >> <u>R</u> edo	[Ctrl] [Y]	Redo last action

Note that in all source files the maximum number of characters allowed per line is 2,048. The maximum per word is 256. These limits are imposed by the PSoC Designer development software.

6.3. Adding Files

Adding files to your project, or for use with other projects, is essential for complete, well-balanced M8C functionality.



1. To add a file, click the **New File** icon or File >> New.
2. In the New File dialog box, select a file from the File type field. See Figure 38. (For general facts about these files refer back to File Types and Extensions in section 3.)

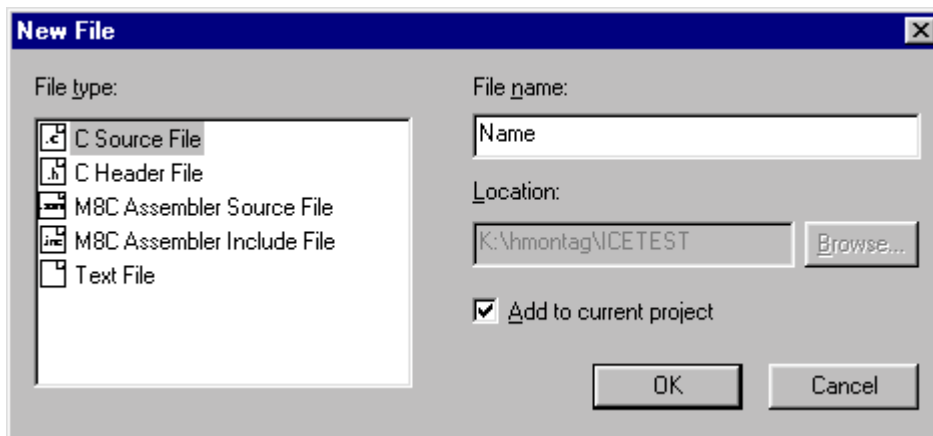


Figure 38: Add New File

3. In the File name field, type the name for the file.
4. In the Location field you will see that your current project directory is the default destination for your file. Click **Browse** to identify a different location if you do not want the default.

The **Browse** button will only be enabled if you uncheck the Add to current project field.

5. By default, your new file will be added to your current project. If you do not want to add this file to your current project, uncheck the Add to current project field. This will then enable the **Browse** button for you to identify a different location.
6. When finished, click **OK**. Click **Cancel** if you wish to cancel the operation.

Your new file will be added to the file source tree and appear in the main active window.

You are also able to add C Compiler files created outside PSoC Designer. Do this by accessing **Project >> Add to Project >> Files**, identifying .c file, and pointing file dialog to and from existing dependency project files. Keep in mind that you will be adding a *copy* of your original file to the project, not the original itself. See *PSoC Designer: C Compiler Language User Guide* for further options and guidelines.

For high-level guidance on programming assembly-language source files for the M8C, see [Section 7. Assembler](#) in this user guide. For comprehensive details, see *PSoC Designer: Assembly Language User Guide*.

6.4. Removing Files

You can remove files from your project one of three ways. Either comment-out an unwanted file through code manipulation, remove the file by clicking once to highlight it in the source tree and accessing **P**roject >> **R**emove from Project, or right-click the file in the source tree and select Remove from Project (**[Delete]** key). The second and third actions remove the file permanently, whereas the first action simply bypasses it. All three ways are acceptable to PSoC Designer.


Section 7. Assembler

In this section you will receive high-level guidance on programming assembly-language source files for the M8C.

For comprehensive details, see *PSoC Designer: Assembly Language User Guide*.

7.1. 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, constructed by you, to produce executable code. This code is then compiled and built into a single executable file that can be downloaded into the In-Circuit Emulator (ICE), where the functionality of the microprocessor can be emulated and debugged.

 To access assembly-language source, click the **Application Editor** icon in the toolbar.

The project source files appear in the left frame (source tree). Double-click individual files to appear in the main active window where you can add and modify code using the enabled edit icons.

7.2. The Microprocessor

The M8C is an enhanced 8-bit microprocessor core. It supports 8-bit operations and has been optimized to be small and fast.

The Internal registers are: the accumulator 'A'; the 'F' flag register; the index register 'X'; the stack pointer 'SP'; and the program counter 'PC'. All registers are 8 bits wide except 'PC' which is composed of two 8-bit registers (PCH and PCL) which together form a 16-bit register.

7.2.1. Address Spaces

There are three separate address spaces implemented in the Assembler: Register space (REG), data RAM space, and program memory space.

The Register space is 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.

The 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, although for RAM sizes 128 bytes or smaller, not all bits are used. The Extended Address flag bits (XA[2:0]) are used to address beyond the first 256 bytes of RAM. Depending on the memory size implemented on a particular device, any or all of the Extended Address bits may not be implemented. These 3 bits provide an 11-bit RAM address for addressing up to 2 kilobytes as 8 pages of 256 bytes each. The flag register must be manipulated to change RAM page addresses.

All stack operations force XA[2:0] on the bus to be zero (leaving flag values intact) so that the stack is constrained to the first 256 bytes page.

The program memory space is organized into 256 byte pages, such that the PCH register contains the memory page number and the PCL register contains the offset into that memory page. The M8C automatically advances PCH when a page boundary needs to be crossed. The user need not be concerned with program memory page boundaries, as they are invisible within the programming module. The one exception to this is that non-jump instructions ending on a page boundary will take an extra cycle to complete. Jump instructions are not affected in this manner.

7.2.2. Instruction Format

Instruction addressing is divided into two groups: (1) Logic, arithmetic, and data movement functions (unconditional); (2) jump and call instructions, including INDEX (conditional).

Logic, arithmetic, and data movement functions are one-, two-, or three-byte instructions. The first byte of the instruction contains the opcode for that instruction. In two-byte instructions, the second byte contains either a data value or an address.

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 memory sizes larger than 4 kilobytes, a three-byte format is used in Big Endian format.

7.2.3. Addressing Modes

Ten addressing modes are supported; Source Immediate, Source Direct, Source Indexed, Destination Direct, Destination Indexed, Destination Direct Immediate, Destination Indexed Immediate, Destination Direct Direct, Source Indirect Post Increment, and Destination Indirect Post Increment. For examples of each see section 3 in *PSoC Designer: Assembly Language User Guide*.

7.2.4. Destination of Instruction Results

The result of a given instruction is stored in the entity, 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 is a good illustration of this feature (i2 == second instruction byte, i3 == third instruction byte):

Syntax	Operation
AND A, expr	$acc \leftarrow acc \& i2$
AND A, [expr]	$acc \leftarrow acc \& [i2]$
AND A, [X + expr]	$acc \leftarrow acc \& [x + i2]$
AND [expr], A	$[i2] \leftarrow acc \& [i2]$
AND [X + expr], A	$[x + i2] \leftarrow acc \& [x + i2]$
AND [expr], expr	$[i2] \leftarrow i3 \& [i2]$
AND [X + expr], expr	$[x + i2] \leftarrow i3 \& [x + i2]$

The ordering of the entities within the instruction determines where the result of the instruction is stored.

7.3. Assembly File Syntax

Assembly language instructions reside in source files with .asm extensions in the source tree of Application Editor. Each line of the source file may contain five keyword-types of information.

The following table gives critical details about each keyword-type:

Keyword Type	Critical Details
Label	A symbolic name followed by a colon (:).
Mnemonic	An assembly language keyword.
Operands	Follows the Mnemonic.
Expression	Is usually addressing modes with labels and must be enclosed by parentheses.
Comment	Can follow Operands or Expressions and start in any column if the 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).

See section 4 in *PSoC Designer: Assembly Language User Guide* for type definitions and an example of assembly-file syntax.

7.4. List File Format

When you build a project (using all assembly files), 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 also provides a listing of errors and warnings, and a reference table of labels.

See section 5 in *PSoC Designer: Assembly Language User Guide* for an excerpt (*main.asm*) of Example_ADC_28pin (PSoC Designer Example project) and its listing file (*Example_ADC_28pin.lst*).

7.5. Assembler Directives

The PSoC Designer Assembler allows the assembler directives listed below:

Symbol	Assembler Directive
AREA	Area
BLK	RAM Block (in Bytes)
BLKW	RAM Block in Words (16 Bits)
DB	Define Byte
DS	Define ASCII String
DSU	Define UNICODE String
DW	Define Word (2 Bytes)
DWL	Define Word with Little Endian Ordering
ELSE	Alternative Result of IF...ELSE...ENDIF
ENDIF	End of IF...ELSE...ENDIF
EQU	Equate Label to Variable Value
EXPORT	Export
IF	Conditional Assembly
INCLUDE	Include Source File
MACRO/ENDM	Macro Definition Start/End
ORG	Area Origin

See section 6 in *PSoC Designer: Assembly Language User Guide* for descriptions and sample listings of supported assembler directives.

7.6. Instruction Set

All instructions are 1, 2, or 3 bytes wide and fetched from program memory, in a separate address space from data memory. The first byte of an instruction is an 8-bit constant, referred to as the Opcode. Depending on the instruction, there can be one or two succeeding bytes that encode address or operand information.

The following notation will be used for the instructions:

Notation	Description
A	Primary Accumulator
CF	Carry Flag
expr	Expression
F	Flags (ZF, CF, and Others)
I	Operand 1 Value
K	Operand 2 Value
PC	(PCH,PCL)
SP	Stack Pointer
X	X Register
ZF	Zero Flag

To access a complete instruction in detail within PSoC Designer, click your cursor on the target instruction in the file and hit **[F1]**.

See section 7 in *PSoC Designer: Assembly Language User Guide* for the complete instruction set.

7.7. Compiling/Assembling Files

Once you have finished programming all assembly-language source (in addition to any .c source), you are ready to compile/assemble the group of files. Compiling translates source code into object code. (The Linker then combines modules and gives real values to symbolic addresses, thereby producing machine code.) Each time you compile/assemble, the most prominent, open source file will be compiled.

PSoC Designer can decipher the difference between .c and assembly language files and compile/assemble accordingly.



To compile the source files for the current project, click the **Compile/Assemble** icon in the toolbar. The elapsed time will be between 10-40 seconds, depending on file content. Compiling must be done in Application Editor.

As discussed in section 3, the status (or error-tracking) window of Application Editor is where the status of file compiling/assembling resides. Refer back to Figure 15.

Each time you compile/assemble files, the status window is cleared and the current status entered as the process occurs.

When compiling is complete, you will see the number of errors. Zero errors signify that the compilation/assembly 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 **Section 8. Compiling/Assembling Files** in *PSoC Designer: Assembly Language User Guide*. For further details on compiling and building see [section 8](#) in this user guide.


Section 8. Builder

In this section you will learn details of building a project and of the C Compiler as well as basic, transparent functions of the system Linker/Loader and Librarian.

For comprehensive details on the C Compiler, see *PSoC Designer: C Language Compiler User Guide*.

8.1. Building a Project

Once you have compiled the assembly-language source and .c files, you are ready to build your project. Building your project links all the programmed functionality of the source files (including device configuration) and loads it into a .rom file, which is the file you download for debugging. (Linking and loading is discussed ahead in this section.) Building is the final step before entering the debugging phase of the programming-a-system-on-chip process.

 To build the current project, click the **Build** icon in the toolbar. The elapsed time will be between 10-40 seconds, depending on the number of files and their content. Building must be done in Application Editor.

This action builds the entire project and assembles all .asm and .c files from the `c:\project name\ directory` and places them in the `c:\project name\obj` and `c:\project name\output` directories as .rom and .lst files as well as assorted .o and .dbg files. For descriptions of these files, refer to **File Types and Extensions** in [section 3](#).

As with compiling/assembling, the status (or error-tracking) window of Application Editor is where the status of building your project resides. Refer back to [Figure 15](#) in [section 3](#) or [Figure 39](#) here.

Each time you build your project, the status window is cleared and the current status entered as the process occurs. See [Figure 39](#).

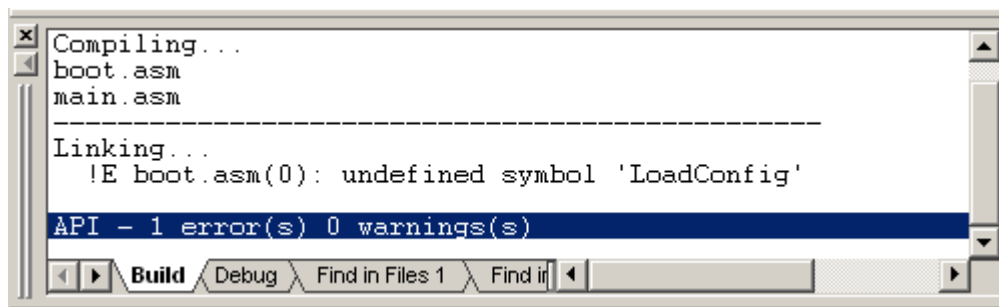


Figure 39: Status Window

To save all open files in Application Editor, click **File >> Save All**.

When the build is complete, you will see the number of errors. Zero errors signify that the build was successful. One or more errors indicate problems with one or more files. Unlike the compilation/assembly process, where syntax errors are revealed, this process focuses on revealing location and value conflicts. Such conflicts/errors include *undefined symbol* and *address already contains a value*.

When you build your project, PSoC Designer automatically compiles/assembles the files first. The build will not run if there are any compilation errors. If there are errors, compilation will error-out, list errors, and halt the build. You must resolve all syntax errors before you can build the project.

For a list of all identified compile and build errors with solutions see **Section 8. Compiling/Assembling Files** in *PSoC Designer: Assembly Language User Guide*.

8.2. C Compiler

The Cypress MicroSystems PSoC family of devices and PSoC Designer support a high-level C language compiler by ImageCraft. Even if you have never worked in standard C language before, this system resource enables you to quickly create a complete C program for the M8C or other PSoC family devices. The C language compiler in PSoC Designer allows users more design flexibility.

The embedded, optimizing C Compiler provides all the features of C, but is tailored to PSoC Designer architecture. It includes a built-in macro assembler allowing assembly-language code to be seamlessly merged with C code. The link libraries use absolute addressing, or can be compiled in relative mode and linked with other software modules to get absolute addressing.

The compiler compiles each .c source file to an M8C .asm assembly file. The PSoC Designer assembler then translates each .asm (either those produced by the compiler or assembly files that have been added) into a relocatable object file, .o. After all the files have been translated into object files, the builder/linker combines them together to form an executable file.

The 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 *PSoC Designer: C Language Compiler User Guide*.

8.3. Linker/Loader

The linking and loading functions in the build process of PSoC Designer are transparent to the user. As discussed earlier in this section, building your project links all the programmed functionality of the source files (including device configuration) and loads it into a .rom file, which is the file you download for debugging.

The linking process links intermediate object and library files generated during compilation/assembly, checks for unresolved labels, and then loads results into a .rom and a .lst file as well as assorted .o and .dbg files. Again, for descriptions of these files, refer to **File Types and Extensions** in [section 3](#).

8.4. Librarian

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

There are two types of Librarian files; Library Source and Library Headers, which can be found in the source tree. Source file types include archived and assembly language such as *libPSoc.a* and *PSocConfig.asm*. Header files are intermediate reference/include files created during application-code generation and compilation. Both types are generated and used by PSoC Designer and unique to each specific project. See **File Definitions and Recommendations** in [section 6](#) for recommended usage.

This page has intentionally been left blank.



Section 9. Debugger

In this section you will learn how to connect and download your project to the In-Circuit Emulator, debug/perfect functionality, and program the part.

The PSoC Designer Debugger provides in-circuit emulation that allows you to test the project in a hardware environment while viewing and debugging device activity in a software environment. Following are the necessary components separated and then connected:

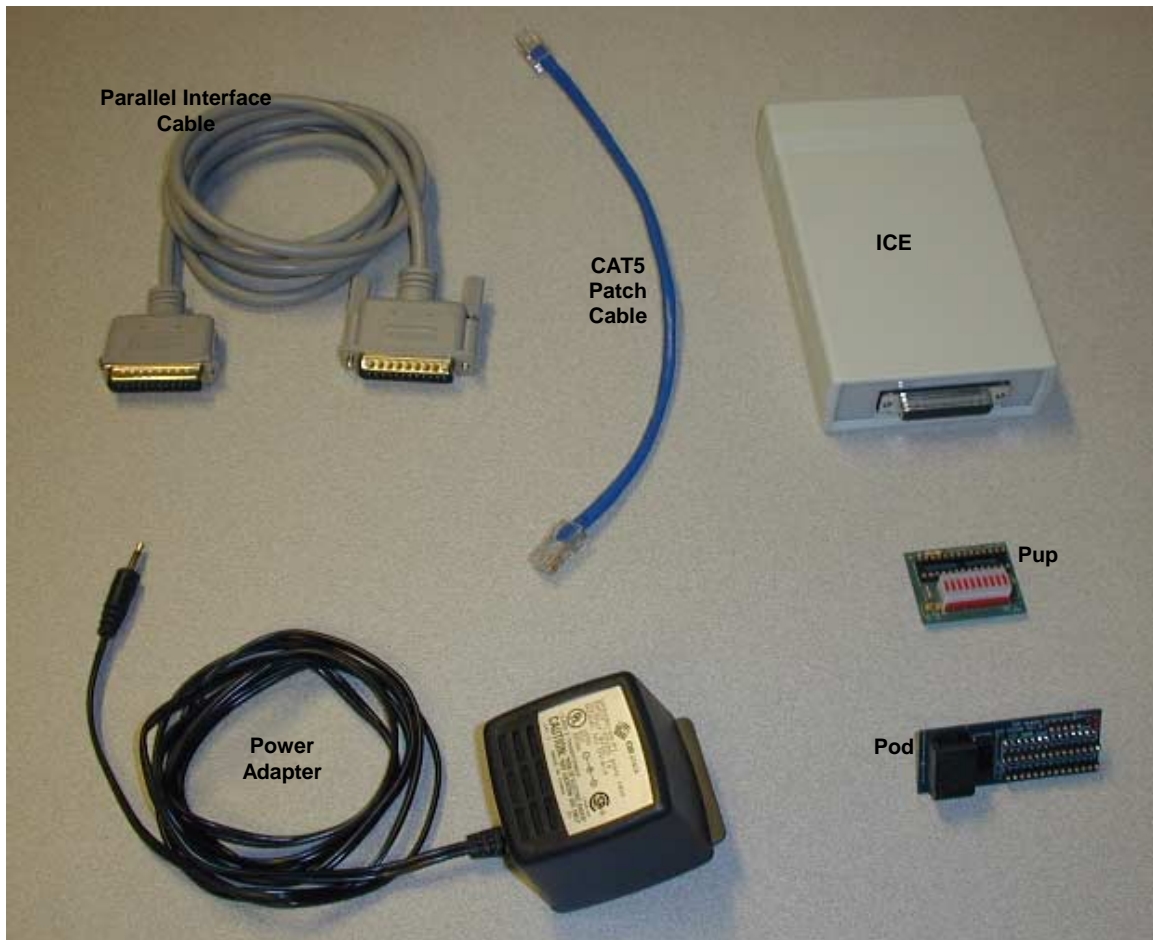


Figure 40: Debugger Hardware Components

Note that the CAT5 Patch Cable should be no longer than 1 ft. It must also have 8 wires in order to connect from the ICE to the Pod (as some data CAT5's only have 4 wires).

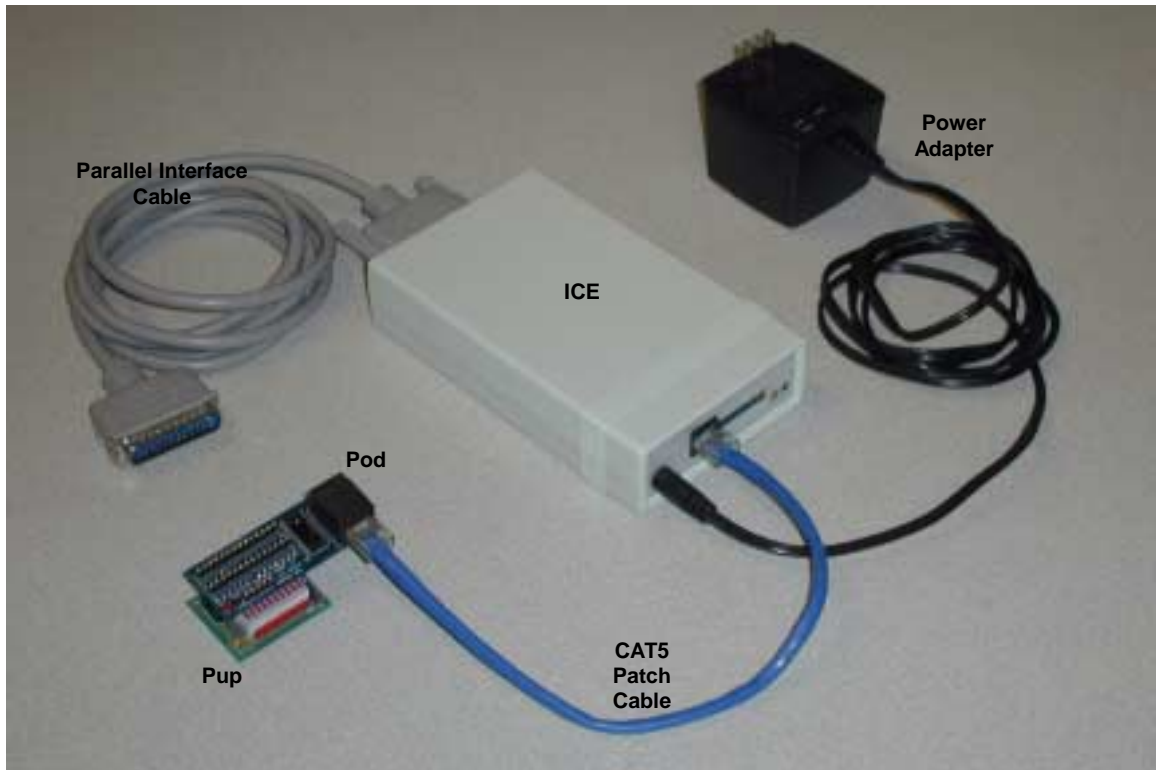


Figure 41: Hardware Components Connected

9.1. Connecting to the ICE

Physically connecting your computer to the In-Circuit Emulator (ICE) and its related hardware is the first step of the two-step process to be done before you can download and debug your project. The second step is connecting to the ICE inside PSoC Designer.

To physically connect your computer to the ICE (and related hardware), execute the following steps and refer to Figure 41:

1. Locate the parallel interface cable, ICE, power adapter, CAT5 Patch cable, Pod, and Pup.
 - a. Plug the parallel interface cable into the LPT1 port (back of computer).
 - b. Plug the available end of the parallel interface cable into the ICE.
 - c. Plug the power adapter into the ICE (and the power).
 - d. Plug the CAT5 Patch cable into the ICE and the Pod.
 - e. Connect the Pup to the Pod (if you are planning to run one of the tutorial/demonstration projects).

Following is a closer look at the Pod, top and bottom:

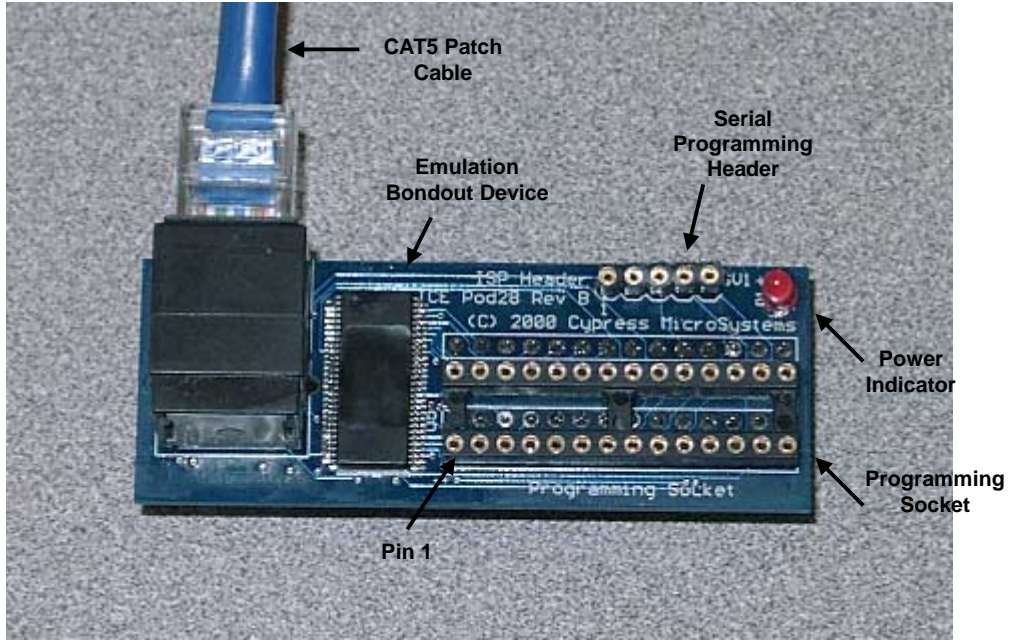


Figure 42: Pod (Top)

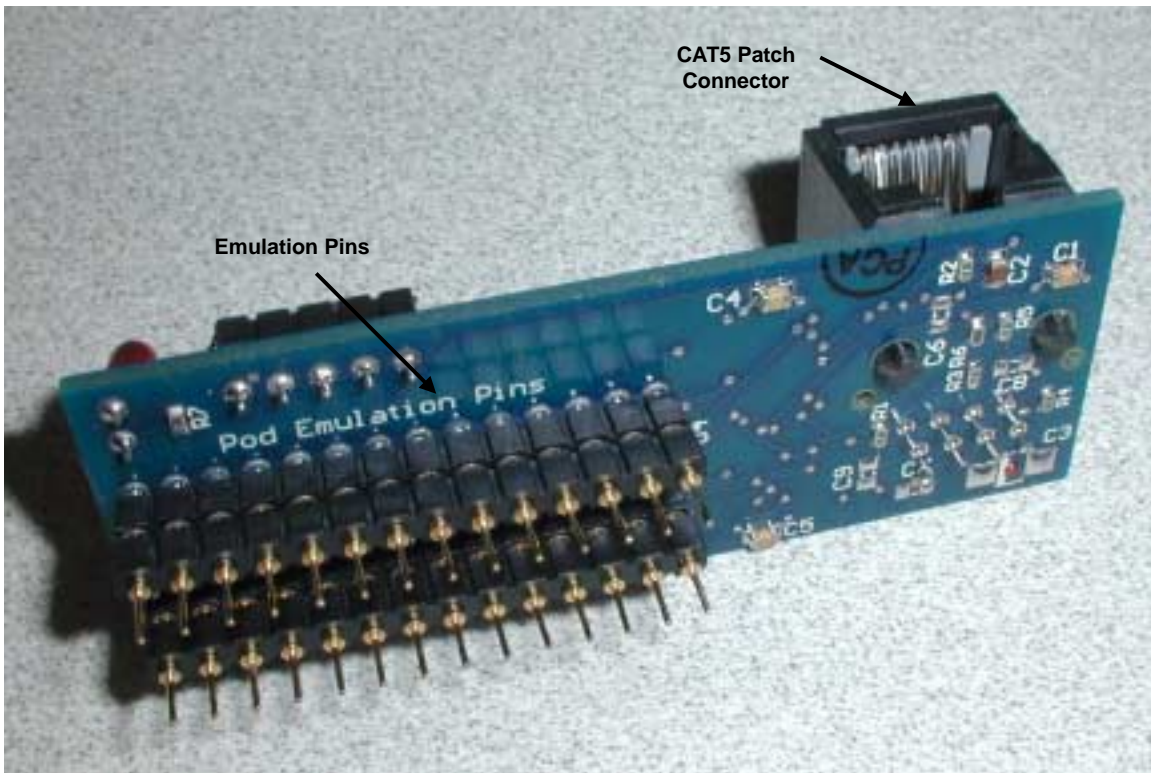


Figure 43: Pod (Bottom)

If your PC's main connection to its printer is through LPT1, you may need to temporarily re-route printing to an alternate port, the network, or a file. This is done through Control Panel >> Printers.

2. Reboot your machine and launch BIOS during boot up by pressing [**F2**] or [**Delete**].
3. In BIOS Setup, select EPP mode, as this setting works most often (for both desktops and laptops).

Because the BIOS varies per machine, the correct mode cannot be known in advance and may take some trial and error. Options include EPP, ECP, Normal (Output Only), and Bi-directional (ECP+EPP).

To test the connection, run through the following steps and execute, "Click the **Connect** icon."



If you are unable to connect, try Normal (Output Only) mode if you are using a desktop, and Bi-directional (ECP+EPP) if you are using a laptop.

Note that if you are running Windows 98 you must switch the setting from Normal (default) to EPP mode in order to successfully connect.

4. Reboot your machine once again to initiate any change made to the BIOS Setup.

If you are running Windows NT/2000, reboot twice (due to a RegEdit delay).

Once you have made the physical connection, you are ready to make the internal connection from PSoC Designer to the ICE. The ICE enables communication and debugging between PSoC Designer and the Pod. To connect to the ICE from inside PSoC Designer, execute the following steps:

1. Confirm that the Pod is connected to the ICE with the CAT5 Patch cable then open PSoC Designer.
2.  Access the Debugger subsystem.
Debugger
3.  Click the **Connect** icon.
Connect

Upon successful connection, you will receive notification and a green light displaying a status of Connected will display in the lower-right corner of the subsystem.

If you are unable to connect, you will see a red light displaying a status of Not Connected. At this, switch the current BIOS setting, as discussed in step 3.

If you have run through the steps a second time and are still unable to connect, please do not alter the IRQ and/or address settings in the BIOS, as these settings are difficult to trace. Default IRQ and address settings are generally compatible with PSoC Designer hardware.

Consult [Troubleshooting](#) at the end of this user guide for in depth details regarding hardware connection.

9.2. Downloading to Pod

Before you can begin a debug session you need to download your project .rom file to the Pod. By doing this, you are loading 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 (M8C) in the programming socket of the Pod. Otherwise, debug sessions may fail.

Execute the following step:

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

The system automatically downloads your project .rom file located in the ...\`output` folder of your project directory. A progress indicator will report download status.

The Pod now can be directly connected to and debugged on your specific circuit board.

9.3. Debug Strategies

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

In the status bar of the Debugger subsystem you will find ICE connection indication, debugger target state information, and (in blue) **Accumulator, X, Stack Point, Program Counter, and Flag** register values.

9.3.1. Trace


This feature of PSoC Designer 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 displays a continuous, configurable listing of program addresses and operations from the last breakpoint. Each time program execution starts, the trace buffer is cleared. When the trace buffer becomes full it continues to operate and overwrite old data.



Project
View

To assist troubleshooting efforts, you can view read-only versions of your application source files inside the Debugger subsystem. If the project source tree is not showing in the left frame, click View >> Project and double-click any file you would like to view.

The Trace window is displayed when Trace is chosen from the Debug menu (or the icon selected ). It is configured by selecting either Debug >> Trace Mode or Tools >> Customize from the menu. Configuration options include PC Only, PC/Registers, or PC/Timestamp.

PC Only mode lists the PC value and instruction only. PC/Registers mode lists the PC, instruction, data, A register, X register, SP register, F register, and ICE external input. PC/Timestamp mode lists the PC, instruction, A register, ICE external input, and timestamp.

The ICE external input value is the binary representation of the 8 center pins on the 10-pin ICE header. The right and left outside pins are connected to ground while the inputs accept a 5-volt TTL level signal. The timestamp is displayed as a 32-bit relative count of clock cycles from the CPU clock source.

The current size of the trace buffer defaults to 16K and is not configurable. However, you can select a different size upon trace configuration (Tools >> Customize).

9.3.2. Breakpoints

This feature of PSoC Designer allows you to stop program execution at predetermined address locations. When a break is encountered, the program is halted at the address of the break, without executing the address's code. Once halted, the program can be restarted using any of the available menu/icon options.

To set breakpoints, first open the file you wish to debug. Do this from the source tree. (If your project file source tree is currently not showing, click View >> Project.) Breakpoints are selected and deselected by clicking your mouse in the left margin of the open file or by using the **Add** and **Remove** (by line) options in the Breakpoints dialog box accessed through Debug >> Breakpoints. See Figure 44.

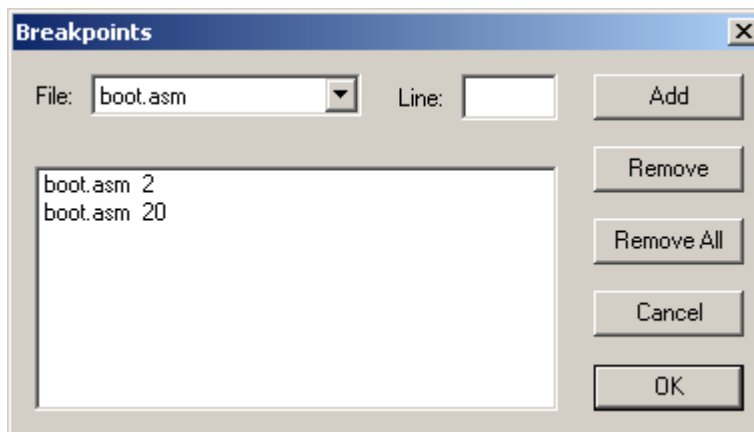


Figure 44: Debug Breakpoints

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

9.3.3. CPU and Register Views

There are five accessible “watch” windows that are readable and write-able during debugging. They are CPU Registers, Bank Registers 0,1, RAM, and FLASH (accessed at View >> Debug Windows).

The CPU Register window allows you to examine and change the contents. Click the drop-arrow to access a register then double-click and type over the value.

The CPU register values can also be viewed in blue across the bottom of PSoC Designer.

Each register is viewable by clicking the applicable lower tab of the Registers window. Double-click on a location and enter a new value to update the location's value.

The current status of all locations can be saved to a .txt file by right-clicking at the top of the window and selecting Save or Save As.

Exercise caution when changing register values as they can alter hardware functionality.

9.3.4. Watch Variables

Watch Variables can be set at Debug >> Watch Variables (or by right-clicking Add in the Watch Variables window).

In the ASM Watch Properties dialog box you can specify the address you wish to view, the label for the location, the data type located at the address, the location as either RAM or FLASH space, and a display preference of either decimal or hexadecimal. See Figure 45.

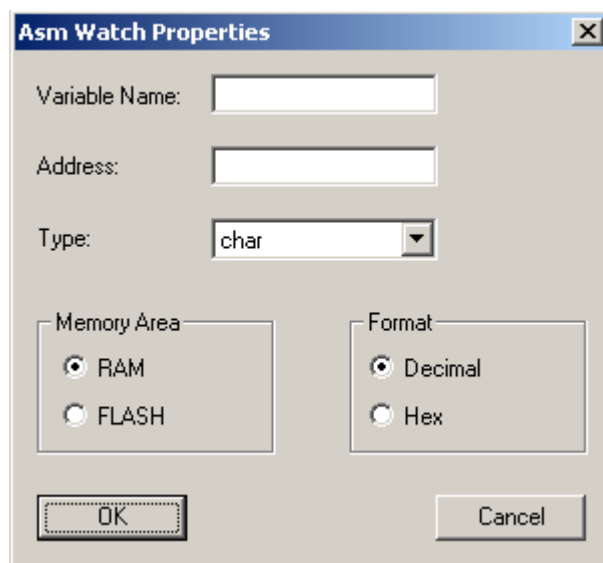


Figure 45: Debug ASM Watch Properties

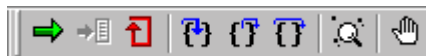
Right-click Delete or Properties in the Watch Variables window to delete or modify settings. To modify a value, double-click and type over.

9.3.5. Events

The Events window is selectable from Debug >> Events and allows you to configure conditional breaks and traces.

9.4. Menu Options

The PSoC Designer Debugger toolbar is shown below:



Following, is a description of all debugging menu options:

Icon	Menu/Tool Tip	Shortcut	Feature
	Debugger		Enables Debugger subsystem
	Connect		Connect PSoC Designer to ICE
	Download to Emulator		Download project .rom file to hardware emulator (Pod). This file holds all device configurations and source-code functionality
	Program Part		This programs the chip by placing and storing ROM data in the FLASH memory
	Start/Go	[F5]	Start debugger
	Stop/Halt	[F6]	Stop debugger
	Reset	[Ctrl] [Shift] [F5]	Reset device to 0 and restart debugger
	Step Into	[F11]	Step into next statement
	Step Out	[Shift] [F11]	Step out of current function
	Step Over	[F10]	Step over next statement
	Activate Trace	[Ctrl] [F]	Activate M8C-trace debugging feature
	Toggle Breakpoint		Toggles the breakpoint: Sets/removes user-defined breakpoints for use in the Debugger subsystem

9.5. Programming the Part

Programming the part occurs once debugging is complete. By doing this, you are storing the ROM data directly in the FLASH memory of the part. The Cypress MicroSystems device can be reprogrammed multiple times due to its FLASH Program Memory. Following is the Pod Programming Socket, which is connected to the CAT5 Patch Cable:

Only the five required serial programming pins are available on the Programming Socket. These required pins are the same pins that make up the Serial Programming Header (V_{CC} , V_{SS} , X_{RES} , P1[1] SCLK, and P1[0] SDATA). The Programming Socket cannot be used for emulation.

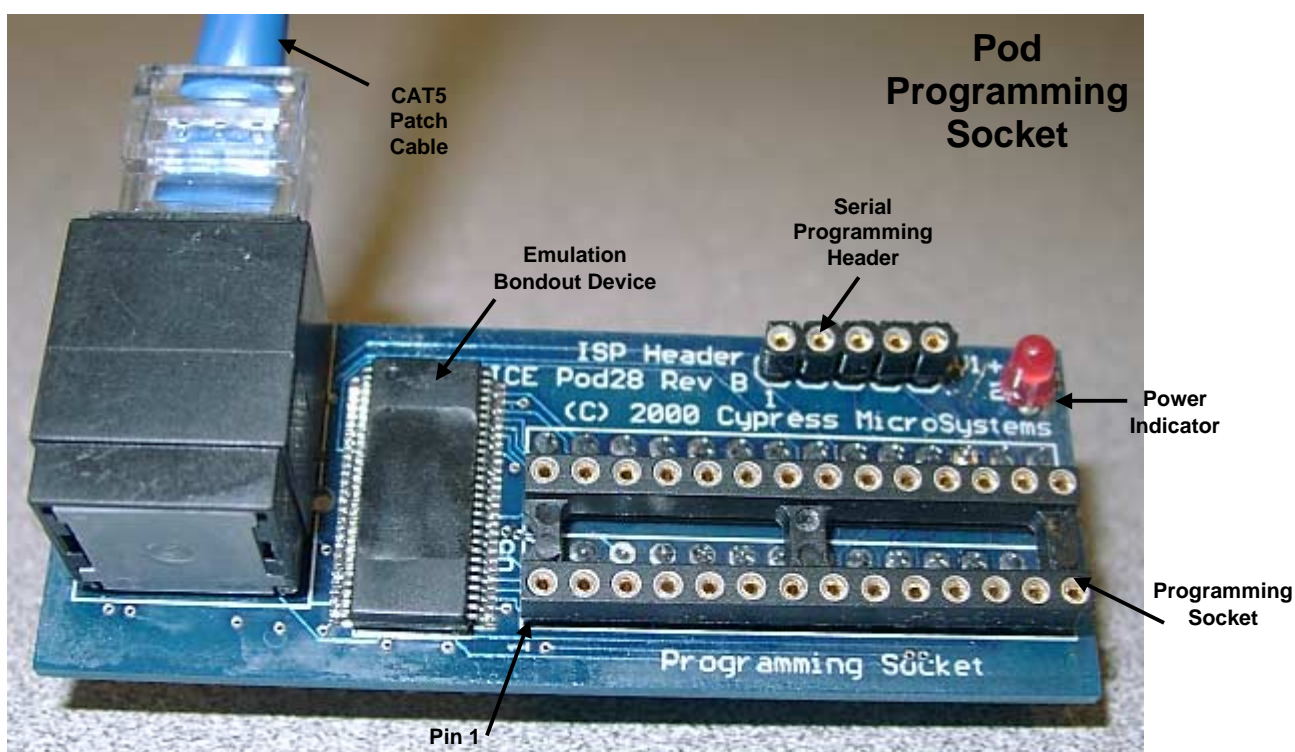



Figure 46: Pod Programming Socket

Make sure the Pod is not connected to a circuit board (your development board or the PSoC Pup) when you program the part. Otherwise, programming (the part) may fail.

Execute the following steps:

1. To program the part, place the part in the Programming Socket on the Pod. (Note the position of Pin 1 on the Programming Socket to ensure correct operation.)
2.  Click the **Program Part** icon and select the .rom file from the ...\`output` folder of your project directory.

Alternatively, the device can be programmed on the target board using the Serial Programming Header on the Pod. The five connections that must be made from the Serial Programming Header to the pins on the target device are listed below:

Header Pin	Device Pin
1	V_{cc}
2	V_{ss}
3	X_{res}
4	P1 [1]/SCLK
5	P1 [0]/SDATA

It is important to note that there is a finite limit to the amount of current that can be supplied to the V_{cc} pin from the emulator Pod (500 mA at 5V). If you draw greater current through the V_{cc} pin on the programming header, this could damage the emulator. You must supply the connections on the target board for serial programming in the system.

Once programming is complete, you can connect the Pod or part to your development circuit board to see how the M8C integrates with your existing product architecture.

This page has intentionally been left blank.

Section 10. Project Tutorial

In this section you will learn to create, configure, compile, build, and debug a project that demonstrates the Analog to Digital User Module.

Note that this is a basic project with the sole purpose of giving you a taste for what can be done with the M8C. Refer to the related sections for specific details.

10.1. Create

Creating a project is the first step in the process. This step creates a project directory with folders for all PSoC Designer project files to reside.

For the purpose of this tutorial, accept default settings unless otherwise advised.

Execute the following steps:

1. Create new project. Upon opening PSoC Designer, select Start new project.
2. Name it Tutorial_DAC.
3. Proceed through **Finish**.

For additional details see [section 4](#).

10.2. Configure

Configuring your chosen device is the second step to programming your M8C. It is a six-step process.

Execute the following steps:



To access Device Editor, click the **Device Editor** icon.

1. Select applicable User Modules.
 - a. DAC6SC.
 - b. Timer16.

2. Track usage of memory and space.

	Total	Used	
Analog Blocks	12	1	
Digital Blocks	8	2	
RAM	256	2	
ROM	16384	132	

Figure 47: DAC6SC and Timer16 User Module/PSoC Block Resources

3. Place and configure User Modules.
 - a. Click **Place User Modules** icon.
 - b. Select the DAC6SC User Module.
 - c. Advance placement position to PSoC block ASB13 using the **Next Position** icon.
 - d. Place DAC6SC by right-clicking and selecting >> Place.
 - e. Set the following value for the DAC6SC under User Module Parameters:

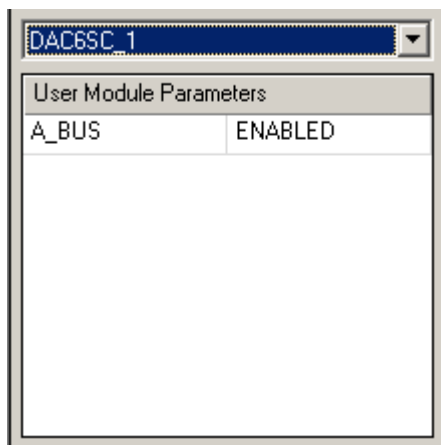


Figure 48: DAC6SC User Module Parameters

- f. Click on (to select) Timer16 User Module.
- g. Place Timer16 on the default PSoC block by right-clicking and selecting >> Place.
- h. Set the following values under User Module Parameters:

User Module Parameters	
Clock	24V2
Input	High
Interrupt_Type	Terminal Count
LSB_Captureval	0
LSB_Period	0
MSB_Captureval	0
MSB_Period	155
Output	None

Figure 49: Timer16 User Module Parameters

- i. Set the following values for the part under Global Resources:

Global Resources	
CPU_Clock	3_MHz
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
24V1= 24MHz/N	16
24V2= 24V1/N	16
Analog Power	ON
Ref Power	Low
Ref Mux	Refs=AGND+/-BandGap
Op-Amp Bias	LOW
SC Power	ON
A_Buff_Bypass	Drive
A_Buff_Power	LOW
SwitchModePump	OFF
VoltMonRange	3.3V
VoltMonThreshold	80%

Figure 50: DAC6SC and Timer16 Global Resources

4. Make interconnections.
 - a. Enable AnalogOutBuffer_3 to connect to Port_0_2 by right-clicking the Comparator Bus and double-clicking Port_0_2.

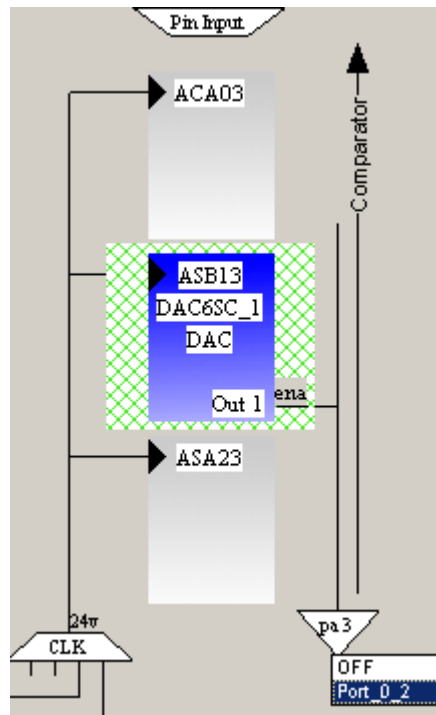


Figure 51: AnalogOutBuffer_3 Port_0_2

5. Set pin-outs.
 - a. Click the **Specify Pin-out** icon.
 - b. At PortPin P0[2] (in left frame of Device Editor) select AnalogOutBuf_3 and leave the drive at its default, HighZ.

PortPin	Select	Drive
P0[0]	StdCPU	Pull Down
P0[1]	StdCPU	Pull Down
P0[2]	AnalogOutBuf_3	High Z
P0[3]	StdCPU	Pull Down
P0[4]	StdCPU	Pull Down
P0[5]	StdCPU	Pull Down
P0[6]	StdCPU	Pull Down
P0[7]	AnalogInput	High Z
P1[0]	StdCPU	Pull Down
P1[1]	StdCPU	Pull Down
P1[2]	StdCPU	Pull Down
P1[3]	StdCPU	Pull Down
P1[4]	StdCPU	Pull Down
P1[5]	StdCPU	Pull Down
P1[6]	StdCPU	Pull Down
P1[7]	StdCPU	Pull Down
P2[0]	StdCPU	Pull Down
P2[1]	StdCPU	Pull Down

Figure 52: PortPin P0[2]

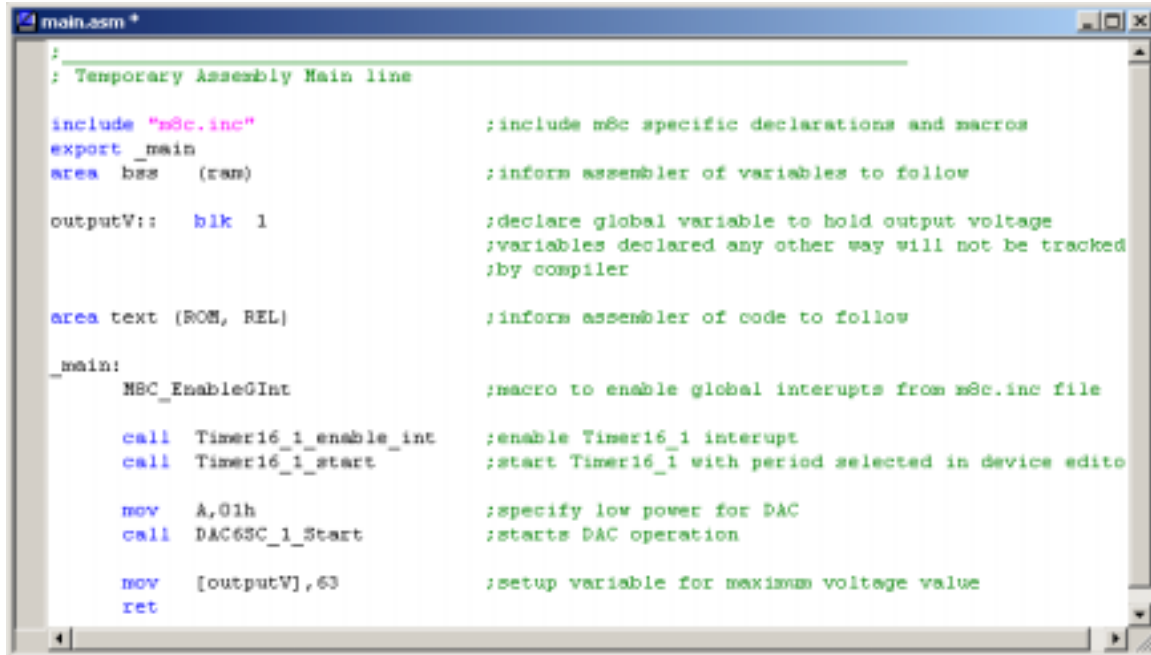
6. Generate application files/APIs and ISRs by clicking **Generate Application** icon. When the process is complete, select Application Editor as your next step.

10.3. Compile/Assemble

Compiling/assembling is the next step. This is done once you have finished programming all assembly language source files (see sections 6 and 7).

To program and compile assembly source files execute the following steps:

1. In the source tree under Source Files double-click *main.asm*.
2. Type the following source code:



```

main.asm +
;
; Temporary Assembly Main line

include "m8c.inc"           ;include m8c specific declarations and macros
export _main
area bss (ram)             ;inform assembler of variables to follow

outputV:: blk 1           ;declare global variable to hold output voltage
                          ;variables declared any other way will not be tracked
                          ;by compiler

area text (ROM, REL)      ;inform assembler of code to follow

_main:
    M8C_EnableGInt        ;macro to enable global interrupts from m8c.inc file

    call Timer16_1_enable_int ;enable Timer16_1 interrupt
    call Timer16_1_start    ;start Timer16_1 with period selected in device edito

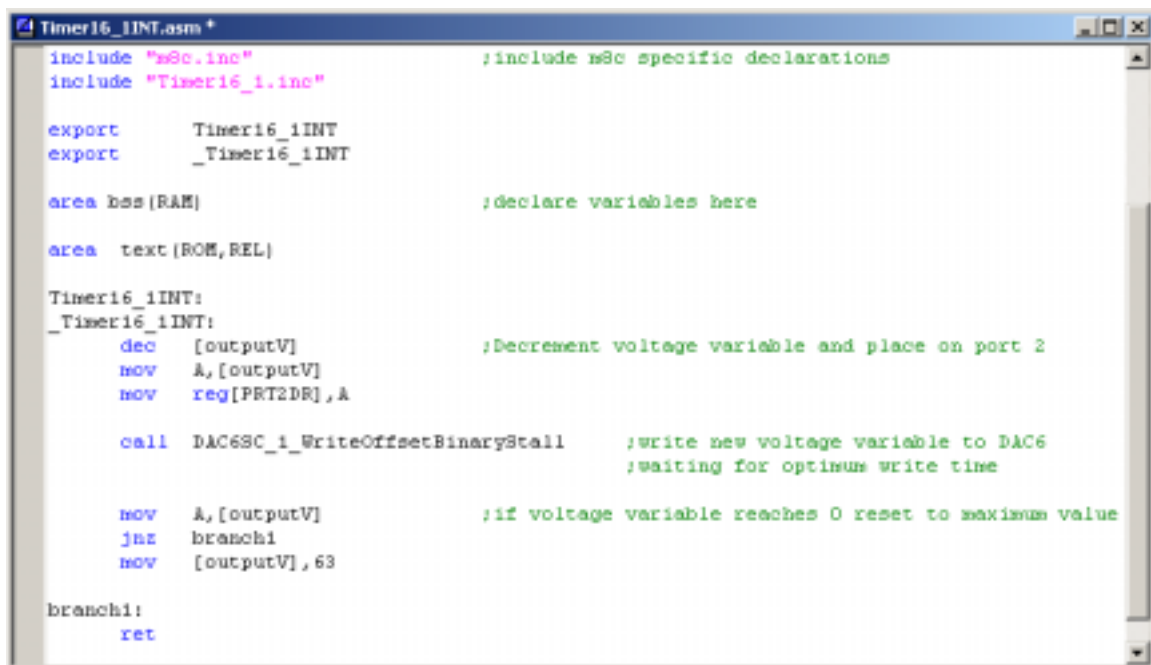
    mov A,01h              ;specify low power for DAC
    call DAC6SC_1_Start    ;starts DAC operation

    mov [outputV],63       ;setup variable for maximum voltage value
    ret

```

Figure 53: *main.asm* Source Code for Tutorial

3. In the source tree under Library Source double-click *Timer16_1INT.asm*.
4. Type the following source code:



```

Timer16_1INT.asm +
include "m8c.inc"           ;include m8c specific declarations
include "Timer16_1.inc"

export Timer16_1INT
export _Timer16_1INT

area bss(RAM)             ;declare variables here

area text (ROM, REL)

Timer16_1INT:
_Timer16_1INT:
    dec [outputV]          ;Decrement voltage variable and place on port 2
    mov A,[outputV]
    mov reg[PRT2DR],A

    call DAC6SC_1_WriteOffsetBinaryStall ;write new voltage variable to DAC6
                                          ;waiting for optimum write time

    mov A,[outputV]        ;if voltage variable reaches 0 reset to maximum value
    jnz branch1
    mov [outputV],63

branch1:
    ret

```

Figure 54: *Timer16_1INT.asm* Source Code for Tutorial

5. **Compile/Assemble.** To compile the source files for the current project, click the **Compile/Assemble** icon in the toolbar. The elapsed time will be between 10-40 seconds, depending on file content.

As discussed in section 3, the status (or error-tracking) window of Application Editor is where the status of file compiling/assembling resides. Refer back to Figure 15.

Each time you compile/assemble files, the status window is cleared and the current status entered as the process occurs.

To save all open files in Application Editor, click **File >> Save All**.

For further details, see sections 6 and 7.

10.4. Build

Building is the next step. This is done once you have compiled the source files. Building your project links all the programmed functionality of the source files (with device configurations) and loads it into a .rom file, which is the file you download for debugging. Building is the final step before entering the debugging phase of the programming-a-system-on-chip process.

1. To build the current project, click the **Build** icon in the toolbar. The elapsed time will be between 10-40 seconds, depending on the number of files and their content. (Note that “building” compiles/assembles first, so in the future you can bypass the **Compile/Assemble** icon.)

Similar to compiling, the status (or error-tracking) window of Application Editor is also where the status of file building resides.

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

For additional details, see [section 8](#).

10.5. Debug

Debugging is the final step in programming your M8C. It can occur after you have created your project, configured the device, (programmed and) compiled the files, and built the project.

Properly debugging involves the following steps:

1. Enter the Debugger subsystem by clicking the **Debugger** icon.
2. Connect your computer and PSoC Designer to the In-Circuit Emulator as described in section 9, 9.1.
3. If PSoC Designer is not already connected to the ICE (step 2) click the **Connect** icon.
4. Download the project file to the Pod by clicking the **Download to Emulator** icon.
5. Click the **Start/Go** icon to execute the program (click the **Stop/Halt** icon to stop).
6. Employ techniques and strategies to test and perfect functionality as described in section 9, 9.3.
7. Program the part.
 - a. Click the **Program Part** icon.
 - b. Place part to be programmed on Pod when prompted.

For complete details see [section 9](#).

10.6 Results

When downloaded and run from the ICE, this project demonstrates the DAC User Module. The program cycles through the 64 possible values of the DAC6 using the Timer16 module interrupt routine as a delay. The current digital value is then supplied to the DAC and LEDs attached to port 2 of the PSoC Pup™ board. The analog output of the DAC is routed through the analog buffer for column 3 and connected to port 0 pin 2 that is available on the user header of the PSoC Pup board. Using a voltmeter, the output analog voltage can be observed.

If difficulties are encountered with the tutorial project, additional information can be found in this user guide as well as in the PSoC Designer help system. A fully documented and working version of this project, `Example_DAC_output_28pin`, is available in the `...\Examples` folder of the directory in which PSoC Designer is installed and can be used as a reference.

Troubleshooting


Following are solutions for pre-identified (potential) system obstacles:

1. During installation of PSoC Designer I receive an error message stating, "You can not expand the support files."

Symptom: During installation of PSoC Designer I receive an error message stating, "You can not expand the support files."

Possible Cause: This error message occurs by starting the installation process and not pressing any **Next** buttons, then starting the installation process again. Therefore, there are two instances of the installation.

Resolution: Proceed gracefully through installation to completion. You can uninstall PSoC Designer after a successful installation by running its uninstall program through Start >> Programs >> Cypress MicroSystems >> Uninstall.

2. I am unable to connect to the In-Circuit Emulator (ICE). (Not Connected appears in red in lower-left corner of PSoC Designer upon clicking the **Connection** icon .)

If you are using a desktop PC with either Windows 98 or ME, try a, c, d, e, f, and g.

If you are using a desktop PC with either Windows NT or 2000, try a, c, d, e, f, h, and i.

If you are using a laptop/notebook with either Windows 98 or ME, try b, c, d, e, f, and g.

If you are using a laptop/notebook with either Windows NT or 2000, try b, c, d, e, f, h, and i.

- a. Desktop Machine Settings
 - b. Notebook Machine Settings
 - c. Parallel port (LPT1) configured to some other device
 - d. Serial port should not be configured to use IRQ7
 - e. Onboard parallel port should be set to "Enabled"
 - f. Under Advanced settings all PCI slots should be "Auto"
 - g. Reboot Windows 98, Windows 98 Second Edition, or Windows ME
 - h. Reboot twice Windows NT 4.0 or Windows 2000
 - i. Windows NT 4.x Device Services
-

Symptom: I am unable to connect to the ICE.

Possible Cause a: Generally, there are little differences between clone desktop machines when compared to name brand proprietary machines from vendors. However, BIOS settings can vary significantly from one brand or model to another.

Resolution: Set or verify the following for desktop machines:

- BIOS setting EPP or Normal (Output Only)
- Parallel port is not marked "Disabled"
- IRQ setting is IRQ7. (Change if it is not set)
- Use default address for the parallel port (usually 378H)

Possible Cause b: Even though the BIOS settings in most notebooks vary significantly, they are less complicated than their desktop counterparts.

Resolution: Set or verify the following for notebook machines:

- BIOS setting EPP or Bi-Directional (ECP+EPP)
- Parallel port is not marked "Disabled"
- IRQ setting is IRQ7. (Change if it is not set)
- Use default address for the parallel port (usually 378H)

Following is additional information on hardware connection to be implemented by advanced users (or in the company of).

Possible Cause c: A printer, or some other device, will not release access to the parallel port. Either a printer has been configured to the same parallel port as the ICE (LPT1), or a printer will not release access to the port. One possible reason is that the printer is actually a fax machine or scanner.

Resolution: If your computer has two or more parallel port connectors, change the port setting for the ICE or re-route printing to an alternate port, the network, or a file. This is done through Control Panel >> Printers.

Possible Cause d: On most machines there are at least two serial ports. The default address for Serial Port 1 is typically 3F8H/IRQ4. The default address for Serial Port 2 is typically 2F8H/IRQ3. If a default has been changed to IRQ7, it is best to switch back to a default. (Remember, BIOS address changes should be implemented by advanced users (or in the company of)).

Resolution: Review your settings to ensure that neither port is configured to use IRQ7. If either of the serial ports is configured to use IRQ7, then there will most likely be configuration conflicts. If a default has been changed to IRQ7, it is best to switch back to a default. This is done under Advanced in the BIOS settings.

Possible Cause e: The parallel port configuration is the single most important configuration section to review. Most configuration problems are resolved simply by ensuring the parallel port is configured properly. Check to ensure that the onboard parallel port is *not set* to “Disabled.”

Resolution: Check to ensure that the onboard parallel port is *not set* to “Disabled.” This is done under Advanced in the BIOS settings.

Possible Cause f: Under “Advanced” settings, there is generally a section called PCI Configuration. The settings under this section control how PCI devices on the machine use addresses. The number of configurations that your machine allows will depend on how many PCI slots your motherboard has. The important selection to set for *all of the PCI slots is “Auto.”* This will allow PCI peripherals to use addresses as needed, or by hard-set jumpers on the PCI cards. Make sure that none of the PCI slots are configured to use IRQ7.

Resolution: Check to ensure that *all PCI slots are set to “Auto.”* Also, make sure that none of the PCI slots are configured to use IRQ7. These settings are under Advanced in the BIOS settings.

Possible Cause g: An important note with different operating systems is that after installing PSoC Designer on a machine running Windows 98, Windows 98 Second Edition, or Windows Millennium, the machine will need to be rebooted. If you have not rebooted, it could hamper your ability to connect to the ICE. Same holds true if you have modified BIOS settings.

Resolution: If you are using one of the operating systems mentioned above and did not reboot after installation or you modified the BIOS, reboot.

Possible Cause h: An important note with different operating systems is that after modifying the BIOS settings on a machine running Windows NT 4.0 or Windows 2000, the machine will need to be rebooted *twice*. If you have not rebooted, it could hamper your ability to connect to the ICE.

Resolution: If you are using one of the operating systems mentioned above and did not reboot after modifying the BIOS (or rebooted once), reboot *twice*.

Possible Cause i: If you are running Windows NT 4.x, it is possible that the device service of the ICE driver for the parallel port has not been started.

Resolution: Use the Control Panel >> Devices to check and start the DRIVERX service.

If you are running Windows 98, try rebooting your computer to initiate the parallel port as this operating system has no accessible device services.

Are you connected? If you have exhausted all the recommended options, please consider the following:

- Try making the connection on an alternative PC (to rule out faulty ICE and related hardware)
 - Contact the Cypress MicroSystems Applications Engineering Hotline at 425.939.1014 or email at support@cypressmicro.com
 - Contact your PC hardware vendor/manufacturer
-

Data Dictionary

Following, is system and industry-related terminology used throughout the PSoC Designer suite of product documentation.

Term	Definition
Active Windows	Subsystem-related windows that are open and workable
Analog PSoC Blocks	Basic programmable op-amp 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
API	Application Programming Interface. APIs for source programming are created during application code generation in Device Editor
Application 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
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
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
Debugger	PSoC Designer subsystem where users debug and perfect project functionality
Device Editor	PSoC Designer subsystem where users choose/configure their device
Digital PSoC Blocks	8-bit logic blocks that can be given a personality. The personality can be to act as a counter, timer, serial receiver, serial transmitter, CRC generator, pseudo-random number generator, or SPI.
Family of Devices	8C2xxxx family of devices consists of five pin-outs; 8, 20, 28, 44, and 48
ICE	In-Circuit Emulator that allows users to test the project in a hardware environment (Pod) while viewing and debugging device activity in a software environment (PSoC Designer)
IDE	Integrated Development Environment (for PSoC Designer)
ISR	Interrupt Service Routine. ISR shells for source programming are created during application code generation in Device Editor
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
M8C	Enhanced 8-bit microprocessor core (of 8C2xxxx family of devices) that supports 8-bit operations and is optimized to be small and fast
Pod	Part of the ICE that emulates functionality, in which debugging occurs
PSoC™	Programmable System on Chip
PSoC Blocks	Analog and digital peripheral blocks of a device that are customized by the placement and configuration of User Modules
PSoC Designer	Integrated Development Environment for Cypress MicroSystems' Programmable System-on-Chip technology
Source Tree	Project file system displayed by default in left frame of Application Editor
Subsystem	PSoC Designer has three subsystems; Device Editor, Application Editor, and Debugger
User Module	Accessible, pre-configured function that once placed and programmed will work as a peripheral on the target device

Index

Accessing the Assembler.....	55	Menu Options	73
Adding Files.....	52	Modifying Files.....	51
APIs and ISRs.....	43	Notation Standards.....	3
Assembler Directives.....	58	Placing User Modules.....	32
Assembly File Syntax.....	57	Product Upgrades.....	8
Build.....	83	Programming the Part	70
Building a Project	61	Project Manager	18
C Compiler	62	Purpose	7
Compile/Assemble	81	Removing Files.....	53
Compiling/Assembling Files	59	Section 1. Introduction.....	7
Configuration Methods	26	Section 2. Installation.....	9
Configure.....	77	Section 3. Using the IDE	15
Connecting to the ICE	66	Section 4. Creating a Project.....	23
Create.....	77	Section 5. Device Editor.....	29
Create a Project	23	Section 6. Application Editor.....	49
Data Dictionary.....	89	Section 7. Assembler	55
Debug.....	84	Section 8. Builder	61
Debug Strategies.....	74	Section 9. Debugger.....	65
Deploying Interconnectivity	37	Section 10. Project Tutorial	77
Documentation Conventions	2	Section Overview.....	7
Downloading to Pod	69	Selecting User Modules.....	29
Edit Windows.....	20	Software Requirement Checklist	9
File Definitions and Recommendations.....	49	Specifying Pin-out.....	39
File Types and Extensions	15	Status Window.....	21
Generating Application Files	42	Support	8
Hardware Requirement Checklist	9	The Microprocessor	55
Installing the System	10	Tracking Device Space.....	41
Instruction Set	59	Troubleshooting.....	85
Librarian.....	63	Two-Minute Overview	1
Linker/Loader	63	Working with ISRs	44
List File Format.....	58		