



PSoC® Creator™ User Guide

Document # 001-93417 Rev *K

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
<http://www.cypress.com>

© Cypress Semiconductor Corporation, 2014-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

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

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

Contents



1	Welcome to PSoC Creator	5
	Revision History	5
2	Getting Started	7
	Design Tutorials	7
	How To	46
3	Understanding PSoC Creator	48
	Concepts	48
	General Tasks	51
	PSoC Creator Framework	73
4	Using Design Entry Tools	156
	Schematic Editor	157
	Code Editor	182
	Design-Wide Resources	207
	Symbol Editor	258
	UDB Editor	276
	Other Tools	289
	Common Design Entry Toolbars	294
	Design Elements Palette	296
	Working with Text	297
	Using Text Substitution	298
	Working with Lines	301
	Working with Shapes	302
	Zooming	303
	Scrolling	304
	Design Entry Reserved Words	304
5	Building a PSoC Creator Project	308
	Build Toolbar Commands	309
	Build Menu	310
	Build Settings	311
	Mapper, Placer, Router	329
	Control File	330
	Directives	336
	Generated Files (PSoC 3, PSoC 4, PSoC 5LP)	341

Generated Files (PSoC 6).....	344
Generated Files (FM0+).....	348
Source Code Control	350
Static Timing Analysis	352
CyPrjMgr Command Line Tool	362
CyHexTool Command Line Tool.....	372
CyElfTool Command Line Tool.....	374
Keil Compiler.....	375
Reentrant Code in PSoC 3	376
6 Integrating into 3rd Party IDEs	378
PSoC 6 Designs.....	379
PSoC 4 and PSoC 5LP Designs.....	424
PSoC 3 Designs.....	478
FM0+ Designs.....	489
Keil µVision IDE Notes.....	493
3rd Party Bootloader Support	506
7 Programming and Debugging	516
MiniProg3	517
Select Debug Target	518
Device Configuration.....	521
Using the Debugger	522
8 Completing the Project.....	553
Review Device Datasheet.....	553
Optimize Compiler Settings	554
Download and Archive Development Tools	554
Archive the Project.....	555
Set Build Configuration	555
Select Programming Protocol	556
Enable Device Protection.....	556
Select Optional Reset Line	557
Select Flash Security Protection.....	558
Enable Write Once Latch Flash Protection	558
Evaluate General Programming Options	559
9 Reference Material	560
Component Author Guide	560
Tuner API Reference Guide	561
Third Party References	561
10 Contact Us	562
11 Register PSoC Creator	563
12 Index.....	565

1 Welcome to PSoC Creator



PSoC Creator helps you configure and program analog- and digital-peripheral functionality into a Cypress PSoC device. Using PSoC Creator, you can select and place Components, write C and/or Assembly source, and debug and program the project/part. When used with associated hardware, this dynamic hardware-software combination allows you to test the project in a hardware environment while viewing and debugging device activity in a software environment.

Note This document refers to PSoC 4 devices throughout (in addition to other devices). References to PSoC 4 should be interpreted to include PSoC 4, PSoC Analog Coprocessor, PRoC® BLE (Bluetooth Low Energy), and CCGx.

This PSoC Creator help contains the following sections:

Getting Started	Tutorials to get you started using PSoC Creator.
Understanding PSoC Creator	Information and tasks to better understand PSoC Creator.
Using Design Entry Tools	Tasks and interface descriptions for the graphical design entry tools.
Building a PSoC Creator Project	Topics for configuring and building PSoC Creator projects.
Integrating into 3rd Party IDEs	Information and tasks for generating PSoC Creator design files for use with a 3rd Party IDE
Programming and Debugging	Topics for programming the device and using the debugger.
Completing the Project	Topics for finalizing a PSoC Creator design.
Reference Material	3rd party tool chain docs and other reference material.

Revision History

Document Title: PSoC® Creator™ User Guide		
Document Number: 001-93417		
Revision	Date	Description of Change
**	7/17/14	New document.
*A	7/25/14	Updates to include references to PSoC 4 BLE and PRoC BLE.
*B	12/12/14	Updated screen captures and Design-Wide Resources section.
*C	5/14/15	Updates for PSoC Creator 3.2 and PSoC 4100M and PSoC 4200M.
*D	9/10/15	Updates for PSoC Creator 3.3 and PSoC 4200L.
*E	12/30/15	Updates for PSoC 3.3 Service Pack 1.
*F	9/9/16	Updates for PSoC Creator 4.0.
*G	4/13/17	Updates for PSoC Creator 4.1.
*H	8/25/17	Updates for PSoC Creator 4.2 Beta release.

Document Title: PSoC® Creator™ User Guide**Document Number:** 001-93417

Revision	Date	Description of Change
*I	12/18/17	Updates for PSoC Creator 4.2 Beta 2 release.
*J	2/14/18	Updates for PSoC Creator 4.2 Production release.
*K	2/26/18	Minor doc edit.

2 Getting Started



This section contains tutorials to help get you started using PSoC Creator. There are two sets of tutorials:

- [Design Tutorials](#) - Walkthroughs to get you creating designs quickly.
- [How To](#) - Miscellaneous tutorials to help increase your efficiency using PSoC Creator.

Design Tutorials

This section contains the following design tutorials to help you get started creating designs with PSoC Creator. Code examples for these tutorials are contained in the [Find Code Example](#) dialog, available from the Start page. These tutorials are intended to provide quick introductions to begin using PSoC Creator. For more information, refer to the following

- PSoC 3: AN54181: www.cypress.com/go/PSoC3GettingStarted
- PSoC 4: AN79953: www.cypress.com/go/PSoC4GettingStarted
- PSoC 4 BLE: AN91267: www.cypress.com/go/AN91267
- PSoC 5LP: AN77759: www.cypress.com/go/PSoC5GettingStarted
- PRoC BLE: AN94020: www.cypress.com/go/AN94020
- PSoC Creator Training: www.cypress.com/go/creatorstart/creatortraining

You may also obtain various kits to use with PSoC Creator and associated devices. When installed, these kits provide additional documentation and tutorials, available on the PSoC Creator [Start Page](#).

Beginner:

- [My First Design "Hello World Blinky"](#)
- [Starter Projects](#)

Intermediate:

- [Basic Design](#)
- [Debugging a Design](#)

Advanced:

- [Library Component Project](#)

- [Basic Hierarchical Design](#)

See Also:

- [Find Code Example](#)
- [How To](#)

My First Design "Hello World Blinky"

This tutorial provides an introduction to PSoC Creator and the process of developing a design. The design process includes:

- [Create a New Project](#)
- [Add/Configure Components](#)
- [Write C Code](#)
- [Program the Device](#)

This is the first of a few design tutorials included in this PSoC Creator Help file. This design will show you how to blink an LED. Then you will add another Component to display "Hello World" on an LCD.

Note If you prefer not to create a new empty project, you can open a completed code example for this tutorial, named "HelloWorld_Blinky," using the [Find Code Example](#) dialog. A link to the dialog is located on the PSoC Creator Start page. There are also several [Starter Designs](#) you can create from the New Project dialog.

Create a New Project:

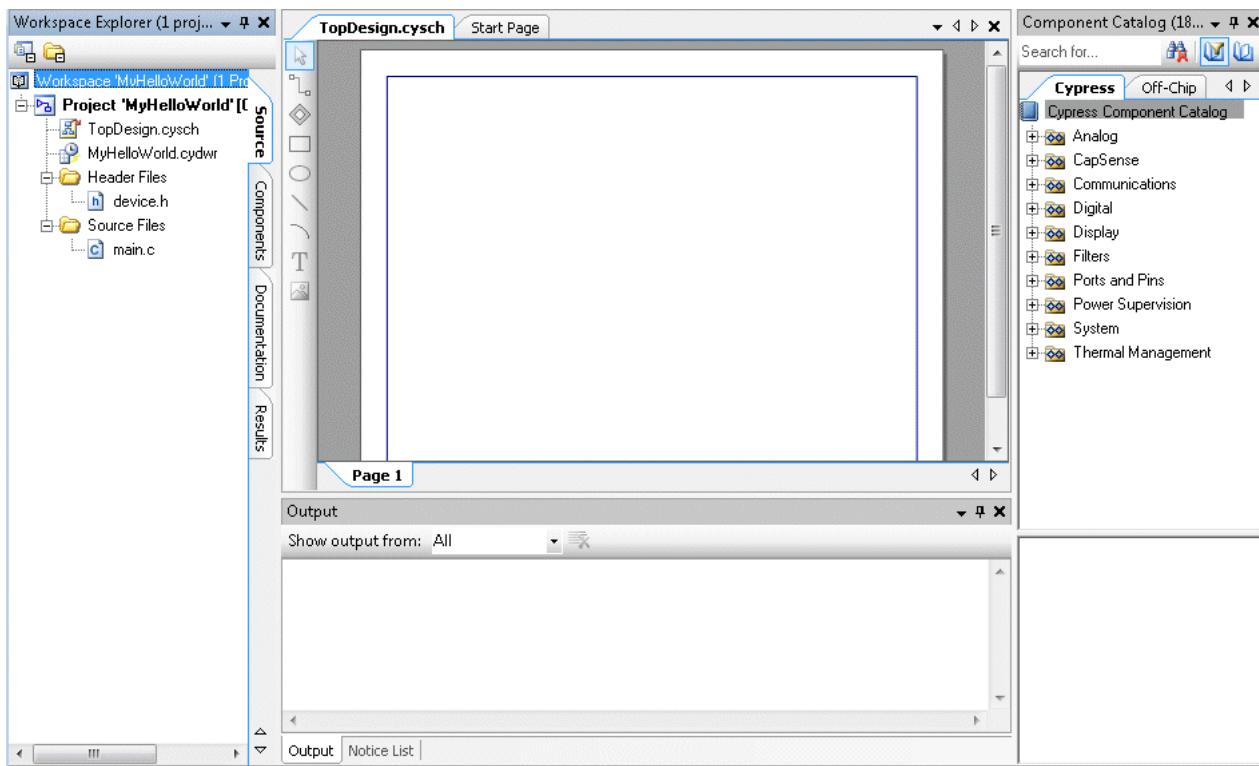
The first step of creating a design is to create the basic design project.

1. From the **File** menu, select **New > Project** or click  to open the [New Project wizard](#).
2. For **Target device**, select the default PSoC 3 device, or select the specific device you want to use. For this project, we are using the default PSoC 3 device CY8C3866AXI-040. If you select a different device, then you will need to adjust your pin settings accordingly.
3. On the "Select project template" page, select **Empty schematic**.
4. In **Name**, type the name of your project, for example: "MyHelloWorld."

Note The project name cannot exceed 80 characters.

5. In **Location**, type the path where you want the project to be saved, or click [...] and navigate to the appropriate directory.
6. Click **Finish**.

By default, PSoC Creator creates a new [workspace](#) containing the new project. Files and folders are added to the [Workspace Explorer](#) shown in the **Source** tab.



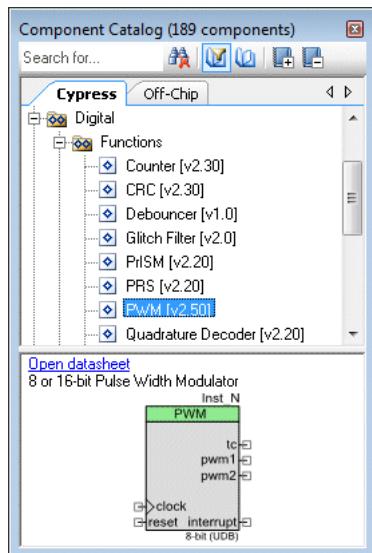
The [Schematic Editor](#) displays the top-level schematic file (*TopDesign.cysch*) as a document window, and the [Component Catalog](#) opens to display a list of Components to use in your design.

Note If you created this project in the same workspace as another project, make this the active project by selecting **Set as Active Project** from the **Project** menu.

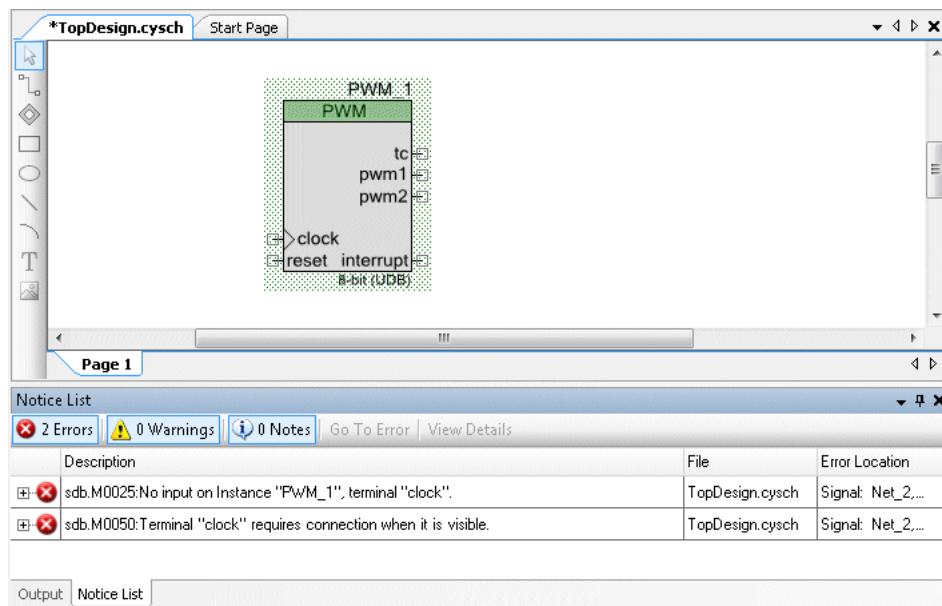
Add/Configure Components:

After the new project has been created, add Components to the schematic canvas and configure them as appropriate.

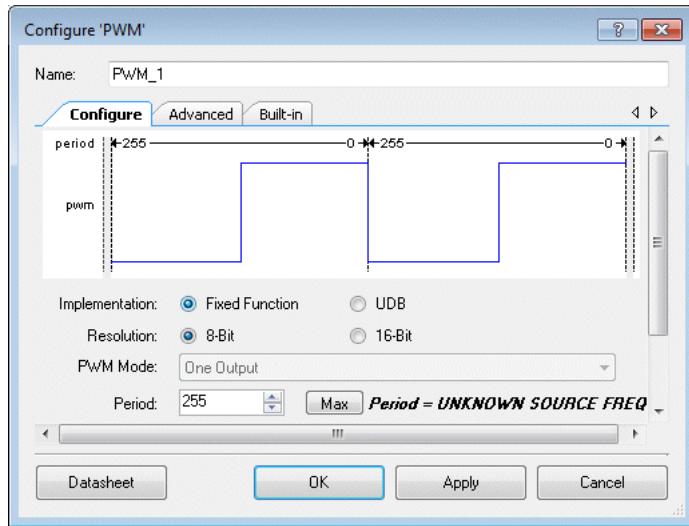
1. In the Component Catalog, expand the "Digital > Functions" folder and drag a PWM Component onto your design.



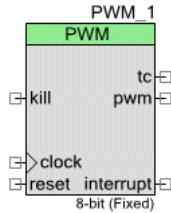
The Notice List window indicates that there are connection errors.



- Double-click the PWM Component to open the Configure dialog, change the **Implementation** setting to "Fixed Function," and click **OK**



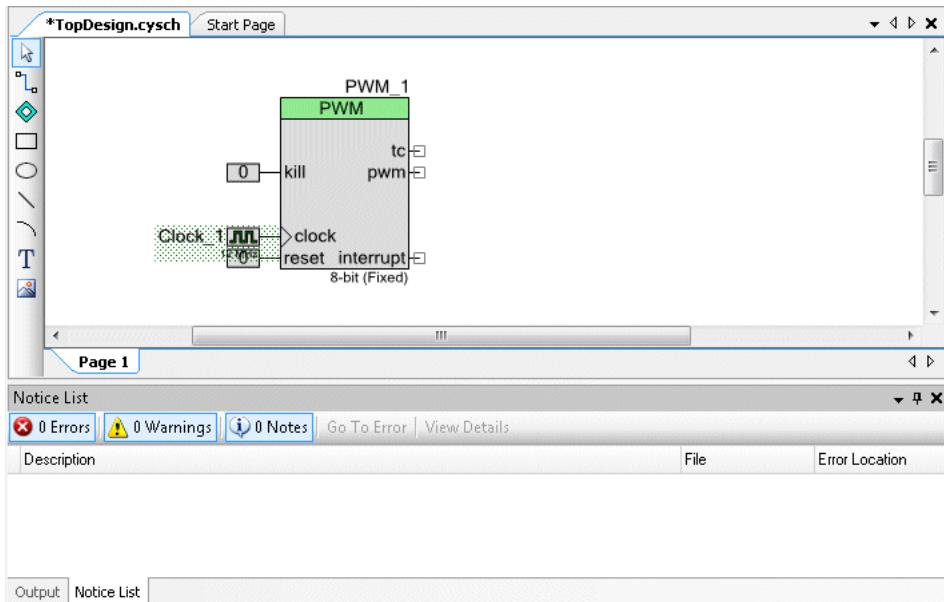
Notice the Component instance's appearance changes: **kill** input added, one **pwm** output available, and the label is updated.



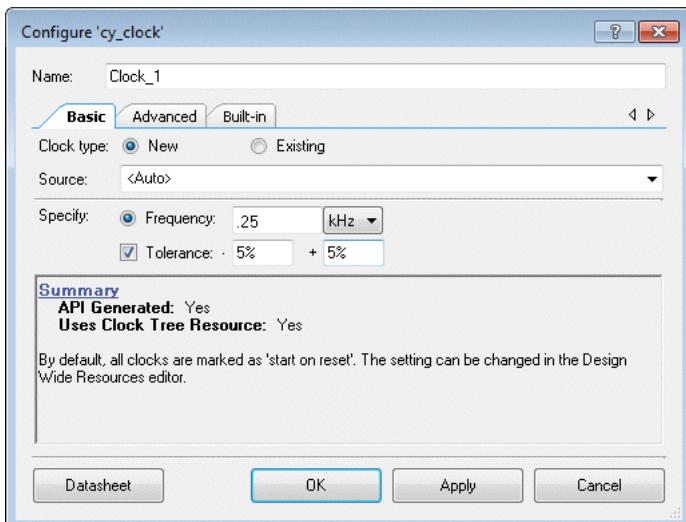
- In the Component Catalog, expand the "Digital > Logic" folder, drag a Logic Low Component onto your design, and connect it to the **kill** terminal on the PWM. Connect another Logic Low to the **reset** terminal.

4. In the Component Catalog, expand the "System" folder, drag a Clock Component onto your design, and connect it to the **clock** terminal on the PWM.

The Notice List window clears.

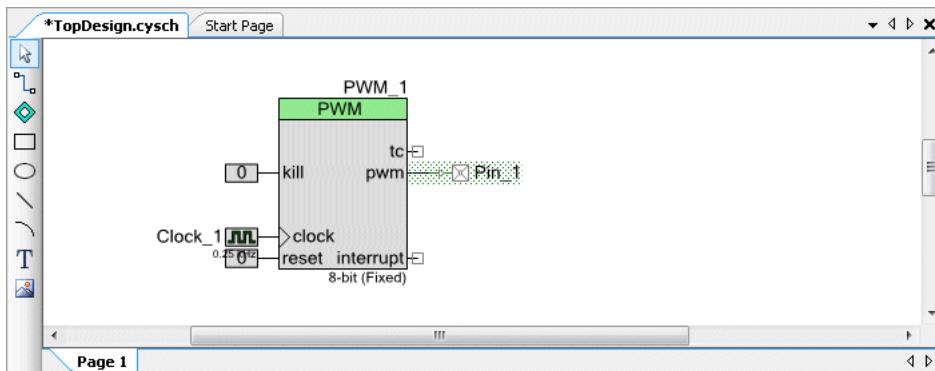


5. Double-click the Clock to open the Configure dialog, change the **Desired Frequency** value to 0.25 kHz, and click **OK**.



6. In the Component Catalog, expand the "Ports and Pins" folder, drag a Digital Output Pin Component onto your design, and connect it to the **pwm** terminal on the PWM.

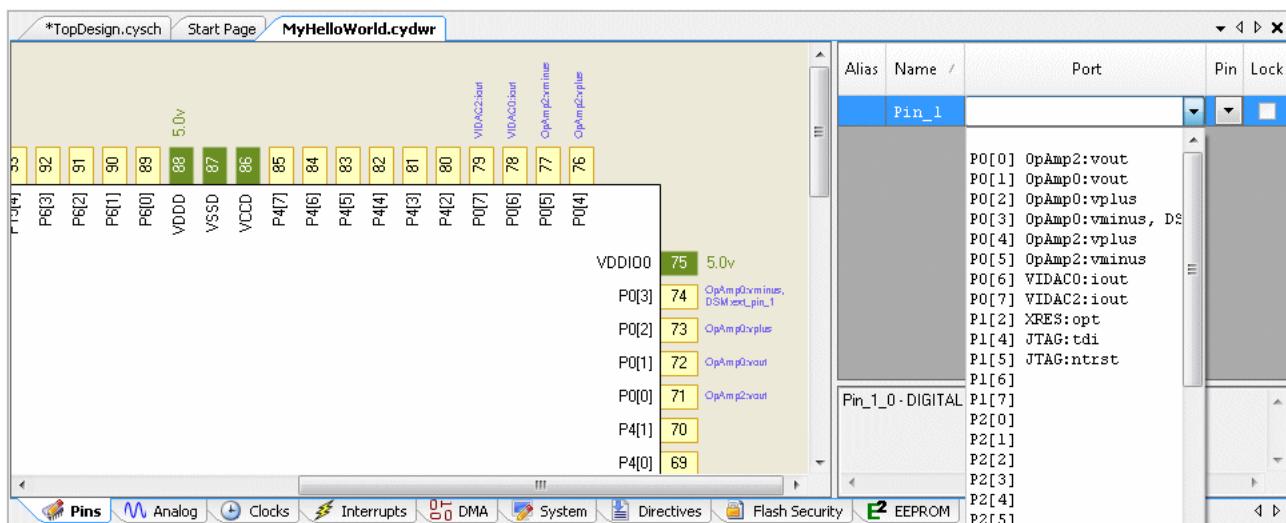
Your design should now look like the following image.



Assign Pin:

PSoC Creator will automatically assign the pin to a physical port/pin on the device. To specify a specific pin, use the [Pin Editor](#).

1. In the Workspace Explorer, double-click the *HelloWorld.cydwr* file to open the Design-Wide Resources Pin Editor.



2. Pull down the menu in the **Port** or **Pin** column and assign Pin_1 to the following pin, depending on the kit you have:

- For the PSoC 3 FirstTouch Starter Kit (CY8CKIT-003), use P4[1] (or pin 70).
- For the PSoC Development Kit (CY8CKIT-001), use P0[0] (or pin 71).

Write C Code:

1. In the Workspace Explorer, double-click the main.c file to open it.
2. Add the following function to main():

```
PWM_1_Start();
```

Program the Device:

1. Connect the MiniProg3 to your kit and plug it into your PC with a USB cable.
2. If you are using the PSoC Development Kit (CY8CKIT-001), use a wire to connect the P0[0] pin to one of the four LEDs. Refer to the appropriate documentation for the kit for more information.
3. Click **Program** .
4. If the [Select Debug Target dialog](#) displays, select your device, then click **Connect** and **OK**.

PSoC Creator will build your design, generate code, and program the device. When programming is complete, the selected LED on the board will blink; press the **Reset** button if needed.

Expand the Design:

If you are using the PSoC Development Kit (CY8CKIT-001), you can expand the design to display "Hello World" on the LCD.

1. On the Component Catalog, expand the "Display" folder and drag a Character LCD onto your schematic.
2. Open the Pin Editor, and assign LCD_Char_1:LCDPort to P2[6:0].
3. Open the *main.c* file, and edit it to add the following to main():

```
LCD_Char_1_Start();
LCD_Char_1_PrintString("Hello World");
```

4. Click **Program** .

PSoC Creator will build your design, generate code, and program the device. When programming is complete, the LCD will display the words "Hello World."

From this point, you can modify the design to do different things as desired.

See Also:

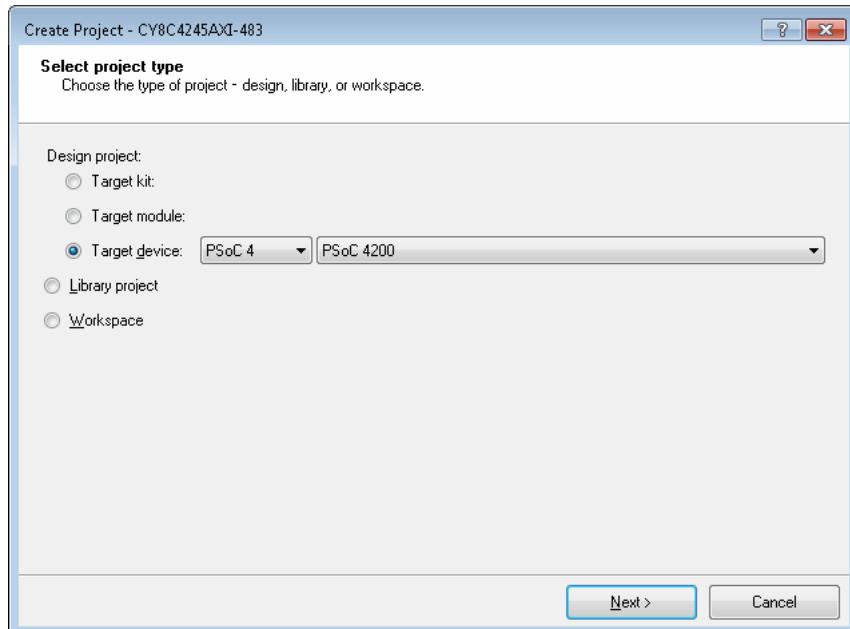
- [Code Examples](#)
- [Find Code Example](#)
- [Workspace Explorer](#)
- [Schematic Editor](#)
- [Component Catalog](#)
- [Pin Editor](#)

Code Examples

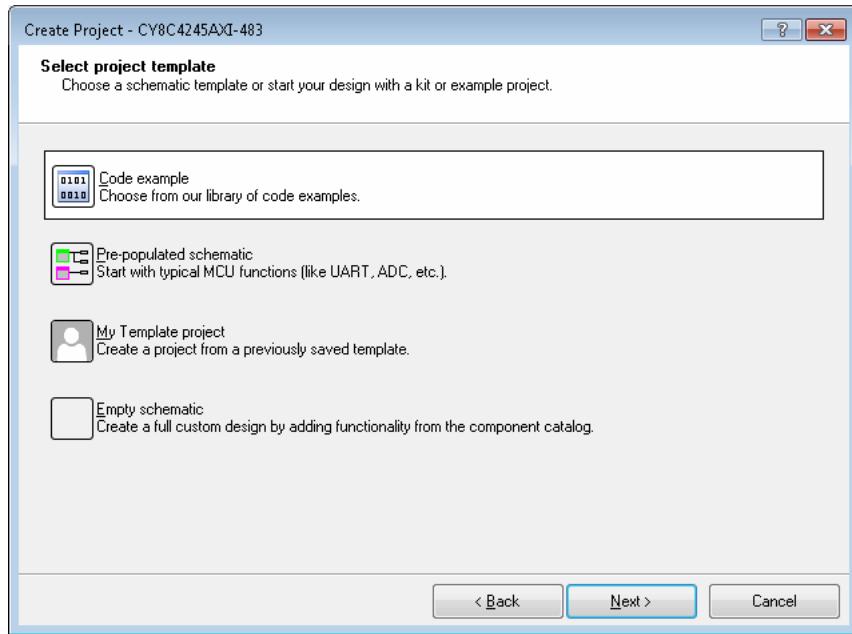
PSoC Creator provides several Code Example projects. These projects highlight features that are unique to PSoC devices. They allow you to create a design with various Components and code already provided, instead of creating a new empty design.

To Use Code Example Projects:

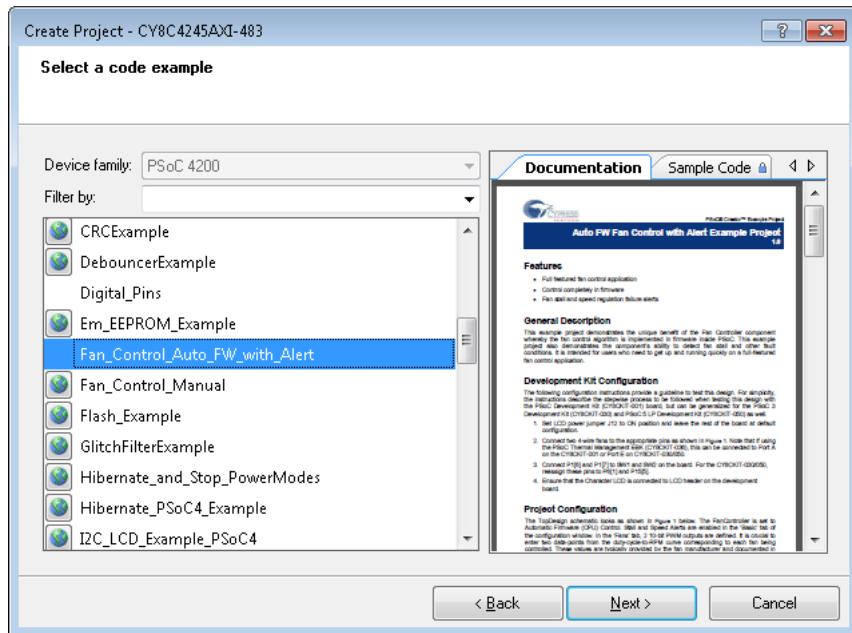
1. From the **File** menu, select **New > Project** or click  to open the [New Project wizard](#).
2. Select the appropriate **Design project** option and click **Next >**.



3. On the "Select project template" page, select the **Code example** option and click **Next >**.

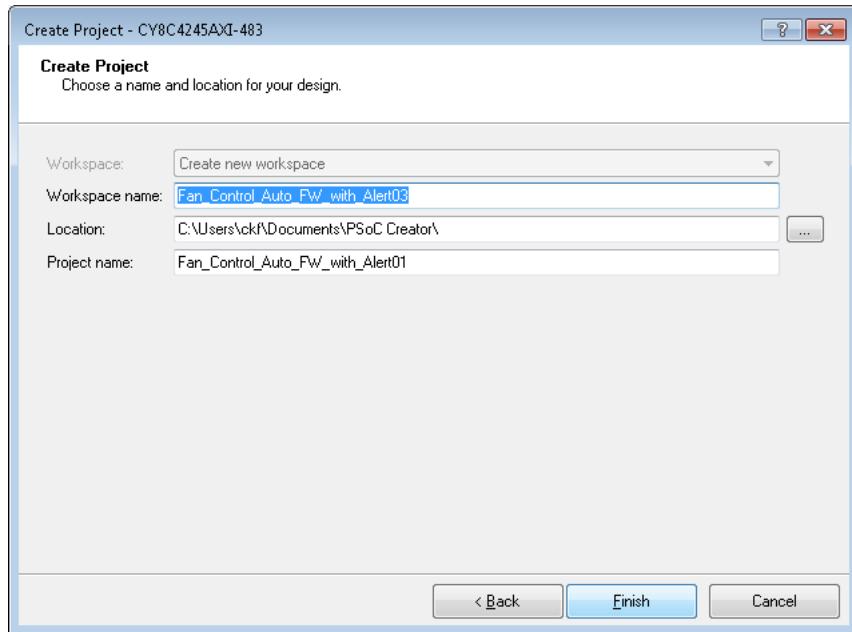


- On the "Select a code example" page, select a project from the list. This page is very similar to the [Find Code Example](#) dialog. If needed, use the **Filter by** field to narrow the number of examples listed.



Note If an example shows an icon next to the name, it means you need to install or update the example from the web. Click on the icon to do that.

- Click **Next >**.
- On the "Create Project" page, select the appropriate options.



7. Click **Finish**.

The selected project is created and files are displayed in the [Workspace Explorer](#). The project is ready to build and program the device. For more information, refer to the documentation included with the design. If the documentation is not available, select the **Include Starter Design documentation...** option in the [Options Dialog](#), and re-create the project.

See Also:

- [Creating a New Project](#)
- [Workspace Explorer](#)
- [My First Design "Hello World Blinky"](#)
- [Options Dialog](#)

My Templates

My Template projects are similar to [code examples](#) and [pre-populated schematic projects](#) in that these projects contain various levels of information already, including device configuration, Components, Design-Wide Resources, etc. The main difference with My Template projects is that they are created and copied to a specific directory by you for later use when creating a new project.

To Specify the My Templates Location:

The default location for storing My Templates projects is <user>\Documents\PSoC Creator\My Templates.

To change this, open the [Options dialog](#). Under **Project Management**, use the **Browse...** button to navigate to the desired location.

To Copy a Project to My Templates:

In the [Workspace Explorer](#), under the **Source** tab, right-click on a design project and select **Copy to My Templates (<project_name>)**. You can also use the **Project** menu to select the same option.

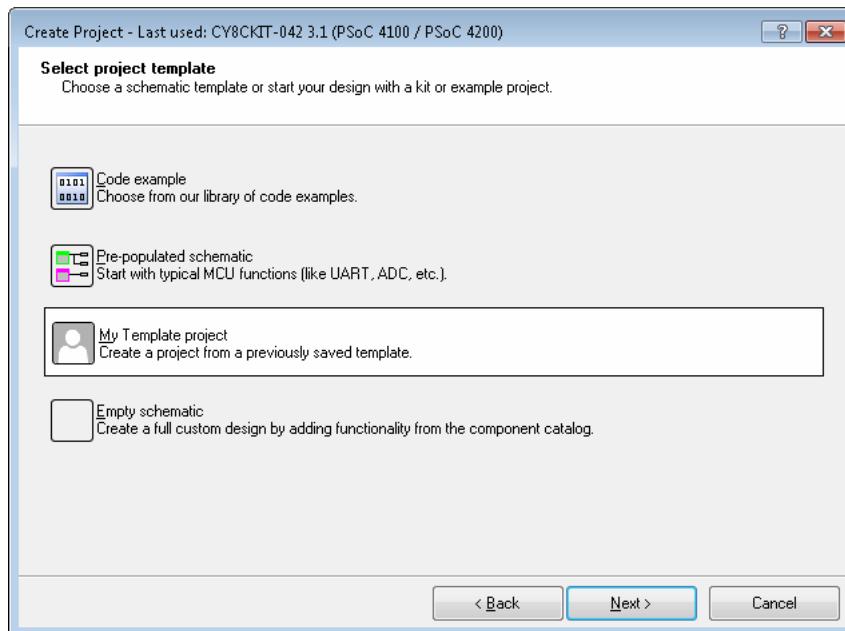
Note When copying a project to the My Templates location, any relative project dependencies will need to be updated when a new project is created from it. Any referenced files outside of the project's cydsn directory will not be included in the new My Template project.

To Create a Project from My Templates:

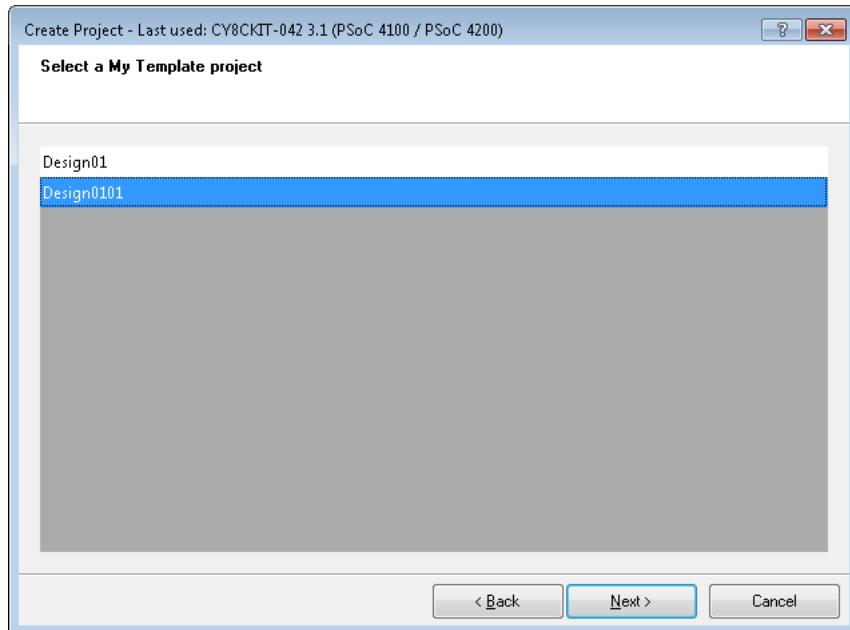
Note When creating a project from My Templates, if the created project is of a different device family than the My Templates project, then the Design-Wide Resource files will no longer be applicable and a new <project>.cydwr file will be generated.

After copying one or more projects to the My Templates location, follow the usual instructions to [create a new project](#), except, as follows:

1. On the Select project template step, select the My Template project option and click **Next >**.



2. Then on the **Select a My Template project** step, select the desired project from the list, and click **Next >**.



3. On the final step, specify the name and location of the project/workspace as usual.

Basic Design

This tutorial provides more details about configuring Components and working with Design-Wide Resources (DWR). It is intended to be a design focusing on a set of basic Components to show various ways of configuring them. This tutorial does not cover every step in the design process in great detail. Instead, it describes concepts relevant to parts of the design. You can then use the same principles in any design. For introductory instructions for creating a design, refer to the ["Hello World Blinky" tutorial](#).

Note If you prefer not to create a new project, you can open the completed code example for this tutorial, named "BasicDesign," using the [Find Code Example](#) dialog. A link to the dialog is located on the PSoC Creator Start page.

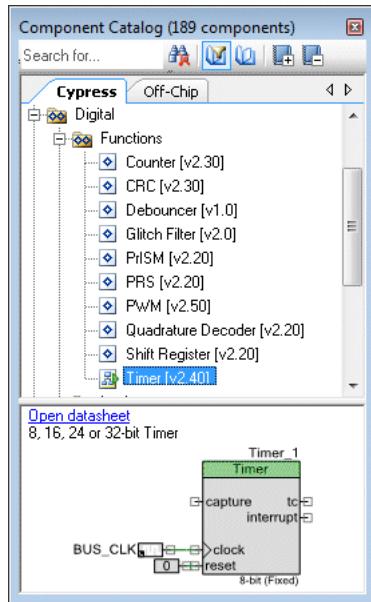
Create a New Project:

As shown in the "Hello World" tutorial, the first step for a design is to create a new project. As needed, follow the instructions in that tutorial. You can also refer to [Creating a New Project](#).

Select and Configure Digital Components:

For this tutorial, the main digital Component will be the Timer. Then, you will add support Components to your design, configure them, and connect them to the Timer.

1. Expand the Digital > Functions folder in the [Component Catalog](#), click on the Timer Component, and drag it onto your design canvas.



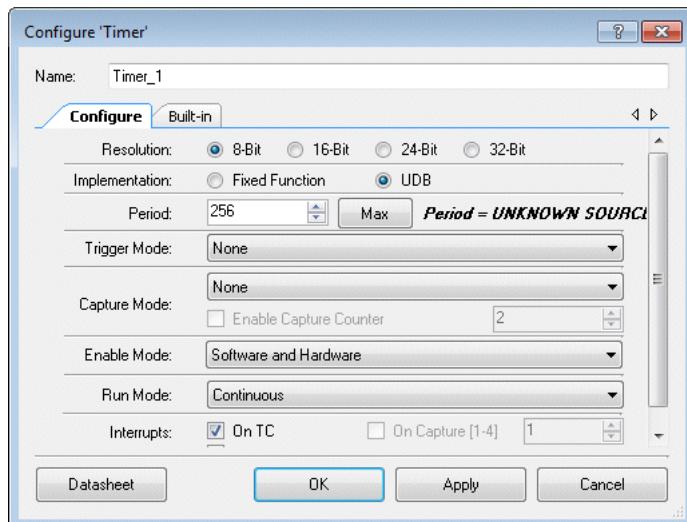
Notice that the Timer is already connected to a Clock Component and a Logic Low Component. This is because the Timer is an example of a [Schematic Macro](#).

2. Remove the Clock and the Logic Low Components because this example will use different Components.
3. Double-click the Timer Component to open the Configure dialog.

By default, the Timer is configured as 8-bit, Fixed Function, and Software Only enable. For more information about the Timer Component, refer to its datasheet.

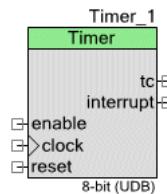
Change the following parameters:

- Implementation to UDB
- Enable Mode** to "Software and Hardware"
- select the **Interrupt on TC** check box



4. Click **OK**.

Notice that the "enable" terminal displays on the Timer.



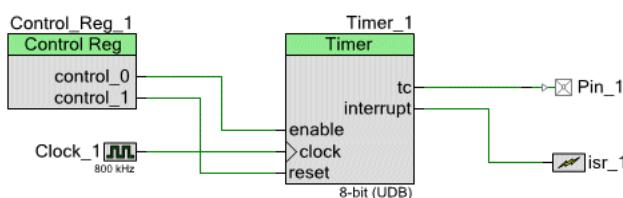
5. From the Component Catalog, add the following Components to your design:

- Control Register (from the Digital > Registers folder) -- Used to enable the Timer under software control, as well as provide a reset signal for the Timer.
- Digital Output Pin (from the Ports and Pins folder) -- Used to provide a divided clock for another Component in the design. The divided clock is 3.1 KHz.
- Clock (from the System folder) -- Used to provide the input clock. In this case at 800 KHz.
- Interrupt (from the System folder) -- Used to act as a heartbeat for the design. The heartbeat can be used for timing of software functions.

6. Configure the following Component instances as follows:

Instance	Parameter(s)
Control_Reg_1	NumOutputs: 2
Clock_1	Desired Frequency: 800 Khz

7. Arrange the Components on your design, and use the **Wire** tool (from the [Design Elements Palette](#), to connect them, similar to the following. See [Working with Wires](#) for more information.

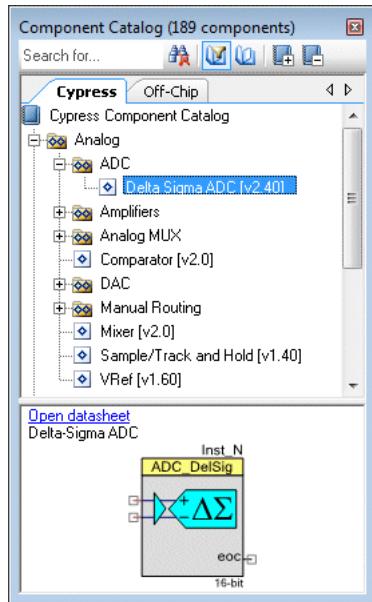


When complete, all DRC warnings and errors should be cleared.

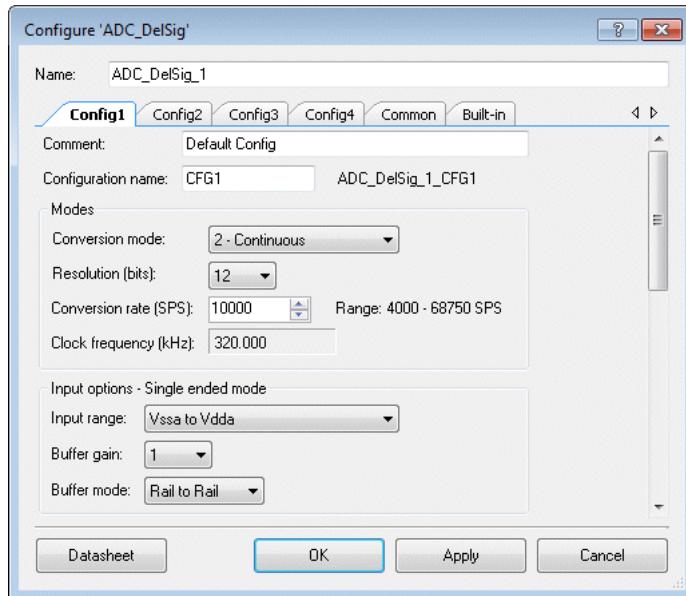
Select and Configure Analog Components:

The next part of the design will contain a Delta Sigma ADC as the main analog Component, along with support Components.

1. From the Component Catalog, drag a Delta Sigma ADC Component (from the Analog > ADC folder) onto your design.



- Double-click the Component to open the Configure dialog.

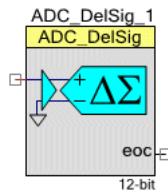


By default, the Delta Sigma ADC is configured as 16-bit, with an input range of +/- 1.024 V. For more information about the Delta Sigma ADC Component, refer to its datasheet.

- Change parameters as follows:

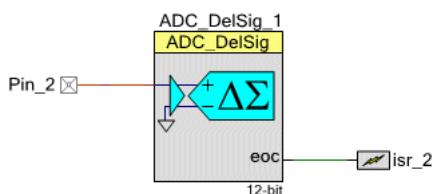
- Under the **Common** tab, set **Input Mode** to "Single ended."
- Under the **Config1** tab, change the **Resolution** parameter to "12" and **Input Range** parameter to "Vssa to Vdda".
- Click **OK**.

Notice that the label below the Component reads 12 Bit Resolution.



4. From the Component Catalog, add the following Components to your design:
 - Analog Pin (from the Ports and Pins folder) -- Used to connect the input signal to the ADC.
 - Interrupt (from the System folder) -- Used to trigger reading the result of a conversion from the ADC.

5. Arrange the Components on your design, and use the **Wire** tool  (from the [Design Elements Palette](#), to connect them, similar to the following:



Notes

- First, notice that the wire from the Analog Pin to the ADC is red by default, and the wire from the ADC to the Interrupt is green by default. PSoC Creator uses these colors to indicate analog and digital signals. These colors can be changed using the Options dialog.
- Second, note that a wire is not needed to connect the Components. You can connect them directly by their terminals . This tutorial uses wires to show the different signal colors.

When complete, all DRC warnings and errors should be cleared.

Edit Source Code:

1. Build the project to generate source files for the Components.

When complete, the Workspace Explorer expands the Generated_Source folder to show all the c files and header files for the various Components in your design.

2. Open the header files to view (and copy) the function prototypes you will enter in the *main.c* file.
3. Copy and paste the following code and replace any code in your *main.c* file.

Note The comments in the following code are for your information.

```
/*
 * File: main.c
 *
 * Version: 2.0
 *
 * Description:
 *   This is a basic design source code.
 *
 ****
#define ADC_NUMBER_SAMPLES (15)
```

```

/* Initialize array elements to zero. */
uint16 ADC_Samples[15] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

/* Defining and initializing the index */
int8 ADC_Sample_Index = 0;

/* Initialize the average result */
uint32 ADC_Sample_Average = 0;

/* Sample read from ADC */
int16 ADC_Current_Sample = 0;

/* Indicator for when sample is available */
int8 ADC_Sample_Available = 0;
//*********************************************************************
* Function Name: InterruptHandler
*****Summary:
* The Interrupt Service Routine for isr_1.
*
* Parameters:
* None.
*
* Return:
* void.
*
*****CY_ISR(InterruptHandler)
{
    Timer_1_ReadStatusRegister(); /* Read the Status Register */
}

*****Function Name: main
*****Summary:
* Main function performs following functions:
* 1: Start the clock
* 2: Start the Timer
* 3: Start the interrupts
* 4: Start ADC DelSig and its interrupts
* 5: Testing for sample available from ADC
* 6: Storing the sample into the array
* 7: Comparing the samples
*
* Parameters:
* None.
*
* Return:
* None.
*
*****void main()
{
    int8 i;

    CyGlobalIntEnable;
    /* Start the Interrupt */
    isr_1_Start();
    isr_1_Disable();
    isr_1_SetVector(InterruptHandler);
    isr_1_Enable();
}

```

```

/* Enable the Timer; reset disabled */
Control_Reg_1_Write(0x01);

/* Start the LCD */
LCD_Start();
/* Start the Timer */
Timer_1_Start();

/* Start the ADC */
ADC_DelSig_1_Start();

/* Start the ADC conversion */
ADC_DelSig_1_StartConvert();

LCD_Position(0, 0);
LCD_PrintString("ADC OUTPUT:");
for(;;)
{
    /* Check whether ADC conversion complete or not */
    if (ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_WAIT_FOR_RESULT))
    {
        /* Get the result */
        ADC_Current_Sample = ADC_DelSig_1_GetResult16();
        ADC_Sample_Available = 1;
        /* Print the ADC result on LCD */
        LCD_Position(0, 11);
        LCD_PrintInt16(ADC_Current_Sample);
    }
    /* Testing for sample available from the ADC */
    if (ADC_Sample_Available)
    {
        ADC_Sample_Available = 0;

        /* storing the sample into the array, based on the index */
        ADC_Samples[ADC_Sample_Index++] = ADC_Current_Sample;

        /* comparison */
        if (ADC_Sample_Index == ADC_NUMBER_SAMPLES)
        {
            ADC_Sample_Average = 0;
            for (i = 0; i < ADC_NUMBER_SAMPLES; i++) ADC_Sample_Average += ADC_Samples[i];
            ADC_Sample_Average /= ADC_NUMBER_SAMPLES;
            ADC_Sample_Index = 0;
        }
    }
}
/* [] END OF FILE */

```

Build the Project:

Click **Build ..**. PSoC Creator will build the project and display messages accordingly. Address any errors or warnings as needed.

If you have a PSoC development kit with an LCD, you can click **Program**  to program the device and observe the output on the LCD.

Next Steps:

This tutorial contains the basic process of configuring Components in a design. To continue this design you will need to obtain hardware to program the device, as well as to use the debugger features. In the mean time, you can refer to the following related topics:

- [Debugging a Design](#)

Debugging a Design

This tutorial will use the design created with the [Basic Design](#) tutorial and go through several debugging concepts, such as setting a breakpoint, adding a watch, and stepping. This design will show the values of an array at different points while executing the code. It will also show the total and average of the specified variable.

Open Example Design:

If you created a design with the Basic Design tutorial, you can use that one. Refer to [Opening an Existing Project](#) for instructions to open the design, as needed.

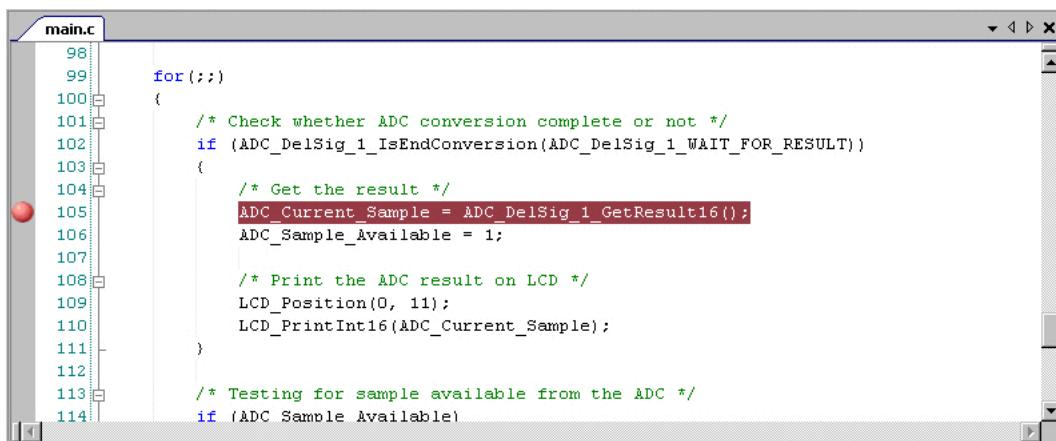
You can also open the completed code example for this tutorial, named "BasicDesign," using the [Find Code Example](#) dialog. A link to the dialog is located on the PSoC Creator Start page.

Set Breakpoints and Step:

In this section, you will set a couple breakpoints and run the debugger to verify that the code is being executed as desired. You will also use the Step function to show where in the stack you are currently viewing. For more information about breakpoints, see [Breakpoints Window](#). For more information about stepping, see [Debugger Toolbar Commands](#).

1. In the Workspace Explorer, double-click the *main.c* file to open it.
2. Scroll to the for loop section and click in the margin of line 105 to insert a breakpoint at the line of code.

A breakpoint indicator appears in the margin next to the desired line of code.



```

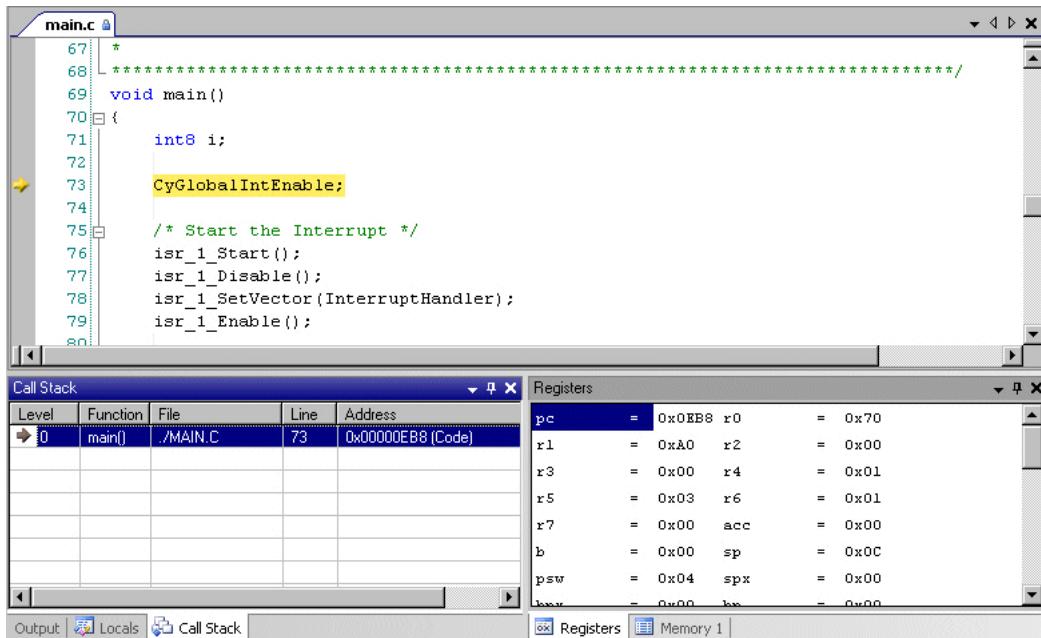
main.c
98
99
100    for(;;)
101    {
102        /* Check whether ADC conversion complete or not */
103        if (ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_WAIT_FOR_RESULT))
104        {
105            /* Get the result */
106            ADC_Current_Sample = ADC_DelSig_1_GetResult16();
107            ADC_Sample_Available = 1;
108
109            /* Print the ADC result on LCD */
110            LCD_Position(0, 11);
111            LCD_PrintInt16(ADC_Current_Sample);
112
113            /* Testing for sample available from the ADC */
114            if (ADC_Sample_Available)

```

3. Click **Debug**  to start the debugger.

Depending on the [Debugger Option settings](#), the Debugger will run to Main.

4. Open the Call Stack window from the **Debug > Windows** menu. Notice the current line indicator is in main().



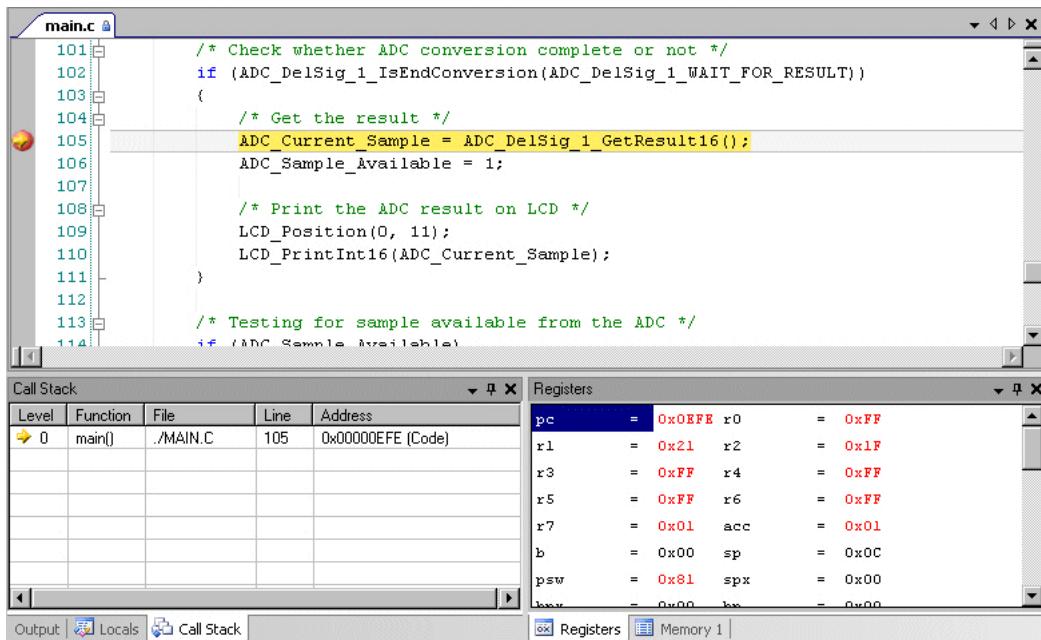
The screenshot shows the PSoC Creator IDE interface. At the top is the code editor for `main.c`. Below it is the Call Stack window, which displays the current call stack levels. The first level is `main()` at line 73. To the right is the Registers window, showing CPU register values. The bottom navigation bar includes tabs for Output, Locals, Call Stack, Registers, and Memory 1.

Level	Function	File	Line	Address
0	main()	./MAIN.C	73	0x00000EB8 (Code)

pc	r0	=	0x70
r1	r2	=	0x00
r3	r4	=	0x01
r5	r6	=	0x01
r7	acc	=	0x00
b	sp	=	0x0C
psw	spx	=	0x00
lwn	kn	=	0x00

5. Click **Continue** .

The Debugger will run to the breakpoint you set. Notice the current line indicator over the breakpoint icon, as well as in the Call Stack window.



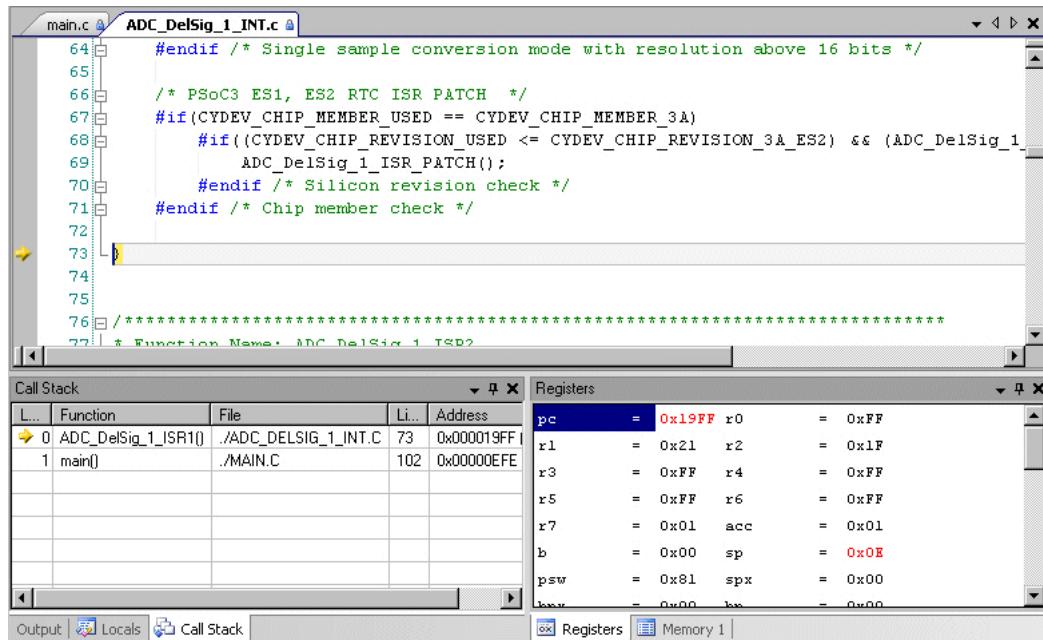
The screenshot shows the PSoC Creator IDE interface after the **Continue** button was clicked. The code editor now highlights line 105. The Call Stack window shows the same information as before. The Registers window shows updated register values. The bottom navigation bar includes tabs for Output, Locals, Call Stack, Registers, and Memory 1.

Level	Function	File	Line	Address
0	main()	./MAIN.C	105	0x00000EFE (Code)

pc	r0	=	0xFF
r1	r2	=	0x1F
r3	r4	=	0xFF
r5	r6	=	0xFF
r7	acc	=	0x01
b	sp	=	0x0C
psw	spx	=	0x00
lwn	kn	=	0x00

6. Click **Step Into** .

The debugger opens the `ADC_DelSig_1_INT.c` file and stops at line 73. Again notice the current line indicator in the Call Stack window.



The screenshot shows the PSoC Creator IDE interface. The code editor window displays C code for `main.c` and `ADC_DelSig_1_INT.c`. The registers window shows CPU register values: pc = 0x19FF, r0 = 0xFF, r1 = 0x21, r2 = 0x1F, r3 = 0xFF, r4 = 0xFF, r5 = 0xFF, r6 = 0xFF, r7 = 0x01, acc = 0x01, b = 0x00, sp = 0x0E, psw = 0x81, spx = 0x00, and r00 = 0x00, r01 = 0x00. The call stack window shows the current stack trace: `ADC_DelSig_1_ISR1()` at address 0x000019FF and `main()` at address 0x000000EFE.

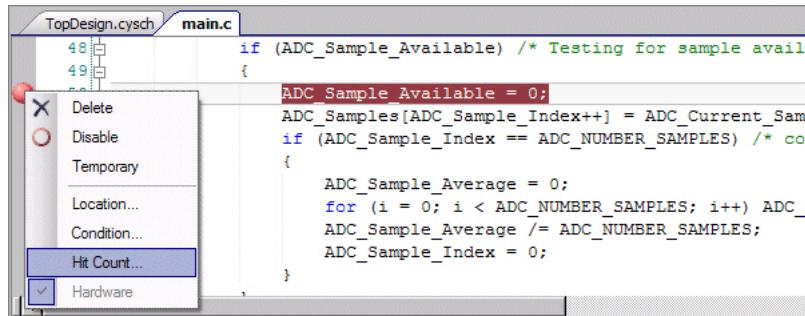
- Stop the debugger by clicking .

Now that you have confirmed the debugger stopped at the desired breakpoint, you can disable the breakpoint by right-clicking on the icon in the margin and selecting **Disable**.

Set Hit Count and Variable Watchpoint:

In this section, you will set a hit count breakpoint as well as a variable watchpoint to monitor the values of an array.

- Open the `main.c` file and scroll to line 50.
- Click in the margin to set a breakpoint and then right-click on the breakpoint icon and select **Hit Count...**



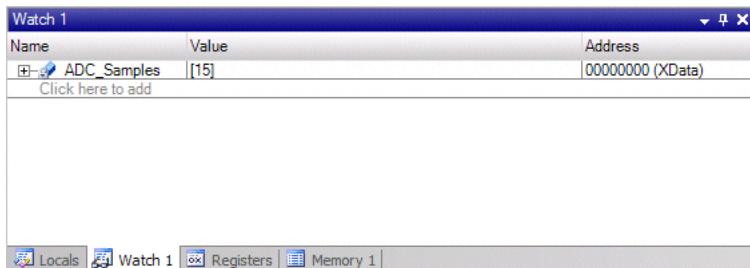
The [Breakpoint Hit Count window](#) opens.

- Enter the hit counter number of "5" and click **OK**.
- Click **Debug**  to start the debugger.

Depending on the [Debugger Option settings](#), the Debugger will run to Main.

- Right-click on the `ADC_Samples` array in line 51 and select **Add Watch**.

The Watch 1 window opens showing the ADC_Samples array. Expand the list to see all the array values.



6. Click **Continue** .

The debugger stops on line 50 and the Watch Window shows the ADC_Samples array with two values assigned.

Name	Value	Address	Type
ADC_Samples	[2278, 2277, 0 <repeats 13 times>]	00000000 (XData)	unsigned int [15]
0	0x08E6	00000000 (XData)	unsigned int
1	0x08E5	00000002 (XData)	unsigned int
2	0x0000	00000004 (XData)	unsigned int
3	0x0000	00000006 (XData)	unsigned int
4	0x0000	00000008 (XData)	unsigned int
5	0x0000	0000000A (XData)	unsigned int
6	0x0000	0000000C (XData)	unsigned int
7	0x0000	0000000E (XData)	unsigned int
8	0x0000	00000010 (XData)	unsigned int

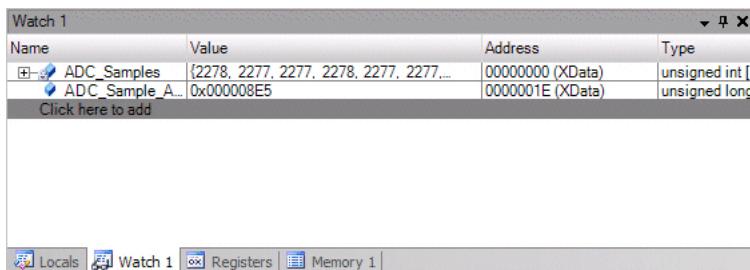
Note Breakpoints are hit twice when interrupts are enabled. This happens because the breakpoint gets hit, but before the line of code is actually executed an interrupt takes over and gets processed. When the interrupt has completed, the processor returns to the original line of code. This causes the breakpoint to be hit again.

7. Disable the breakpoint in line 50.
8. Add another watch to the ADC_Sample_Average in line 54 of the *main.c* file.
9. Right-click in line 56, and select **Run to Cursor**.
10. In the Watch Window, notice that all the values in the ADC_Samples array now have a value and that the ADC_Sample_Average is the total of those values.

Name	Value	Address	Type
ADC_Samples	(2278, 2277, 2277, 2277, 2277, 2278,...)	00000000 (XData)	unsigned int [15]
ADC_Sample_Avg	0x0F1017	00000024 (XData)...	char

11. Right-click in line 57, and select **Run to Cursor**.

12. In the Watch Window, notice that the ADC_Sample_Average is now the average.



For more information about the PSoC Creator Debugger, refer to various topics under the [Using the Debugger](#) section of this Help.

Library Component Project

By default, PSoC Creator provides a library of Components that are available for you to use in your designs. There may be cases where you wish to create your own libraries. This tutorial covers the basic process for creating a library by showing you how to create a 4-bit shifter. This is **only an example** intended to show you how to create basic Components. For more detailed instructions about creating Components, refer to the [Component Author Guide](#).

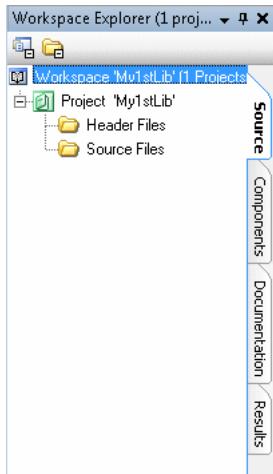
Note If you prefer not to create a new project, you can open the completed code example for this tutorial, named "LibraryComponent," using the [Find Code Example](#) dialog. A link to the dialog is located on the PSoC Creator Start page. Once open, select the **Components** tab in the [Workspace Explorer](#) to view the Component files.

Create a New Project:

The first step is to create the basic library project:

1. If there is currently an open project or workspace, select **Close Workspace**  from the **File** menu.
2. From the **File** menu, select **New > Project** or click  to open the [New Project wizard](#). Select the Library option and click **Next >**.
3. In **Name**, type the name of your project, for example: "My1stLib."
4. In **Location**, type the path where you want the project to be saved, or click [...] and navigate to the appropriate directory.
5. Click **Finish**.

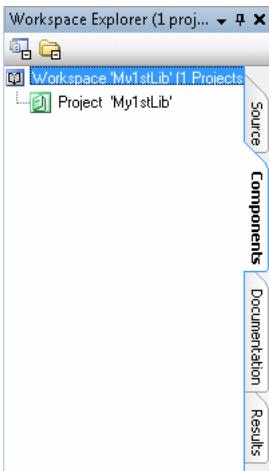
PSoC Creator creates your project and adds files and folders to the Workspace Explorer with the **Source** tab displayed by default. For more information, see [Workspace Explorer](#).



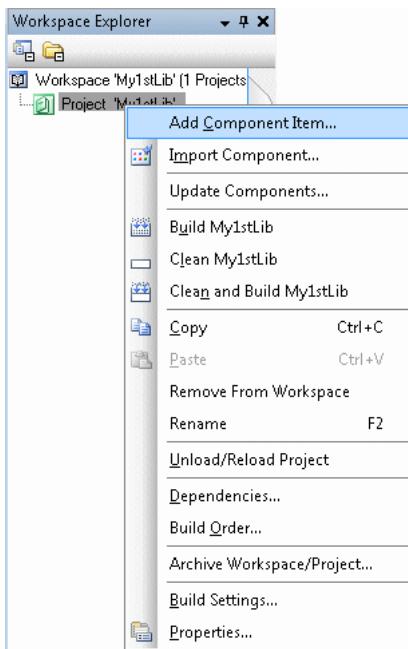
Add a Shifter Component:

The basic library item is a Component. In order to add a Component, you must first add a Symbol, since every Component must have one and only one symbol:

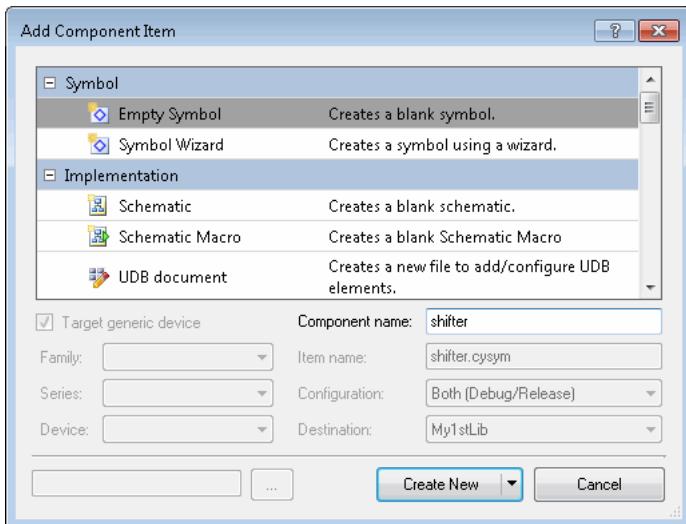
1. Click on the **Components** tab of the Workspace Explorer.



2. Right-click on the project in the Workspace Explorer and select **Add Component Item...**



The Add Component Item dialog displays.

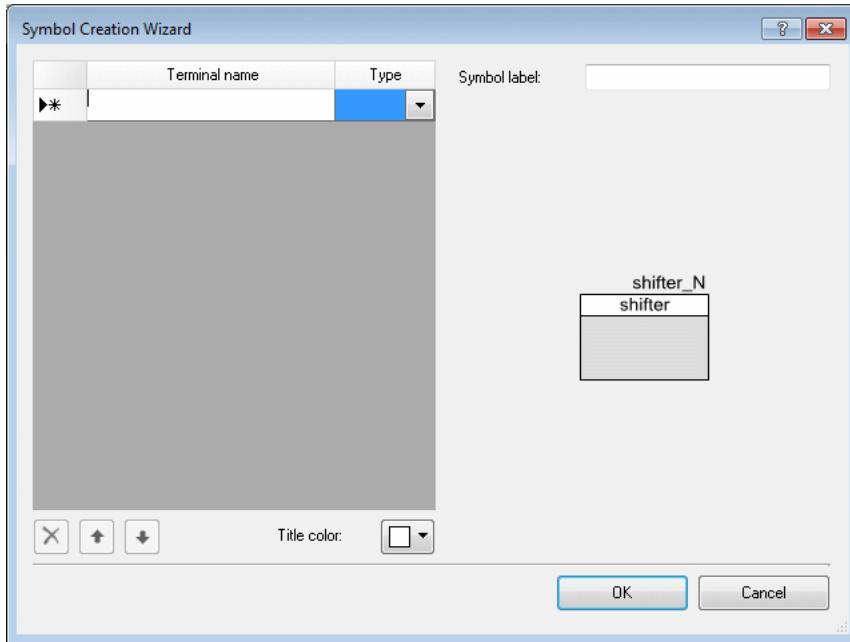


3. Under **Symbol**, click the Symbol Wizard template file.
 4. In **Component name**, type the name to be displayed for your Component, for example: "shifter."

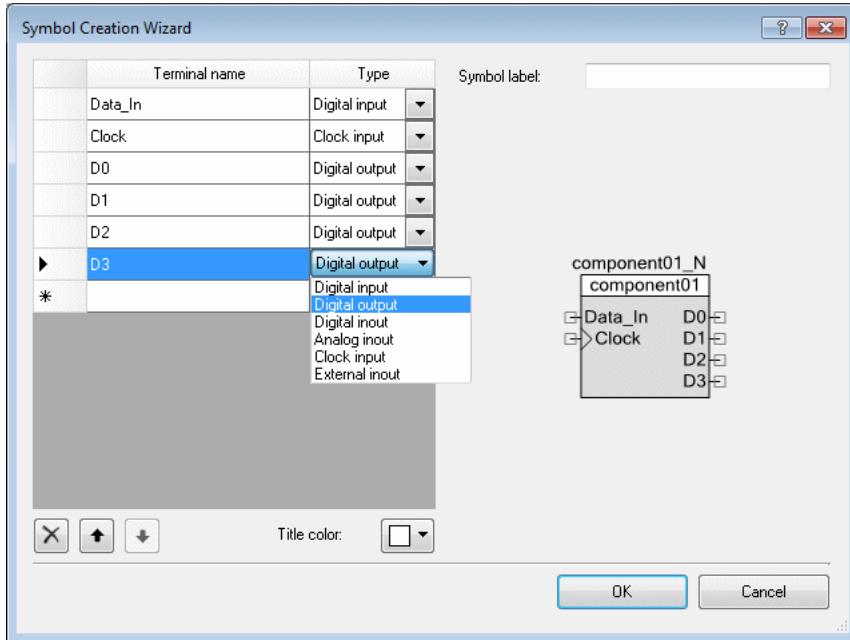
Note The symbol will inherit the Component name, and the name cannot contain spaces.

5. Click Create New.

The Symbol Creation Wizard displays.



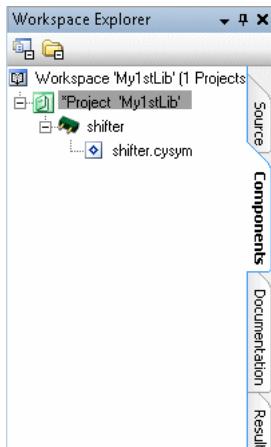
6. Using the **Terminal name** and **Type** fields in the table, create two input terminals named Data_In and Clock, respectively, and four digital output terminals named D0 through D3.



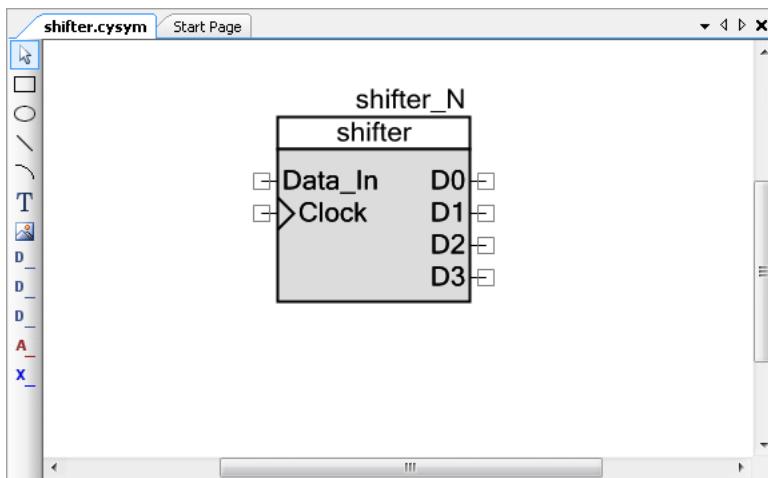
See [Symbol Wizard](#) for more information.

7. Click **OK**.

Your project shows the Component and symbol added to your project. The new Component displays in the Workspace Explorer tree, with a symbol file (.cysym) shown as the only Component item.



The [Symbol Editor](#) also opens the .cysym file and displays the created symbol.



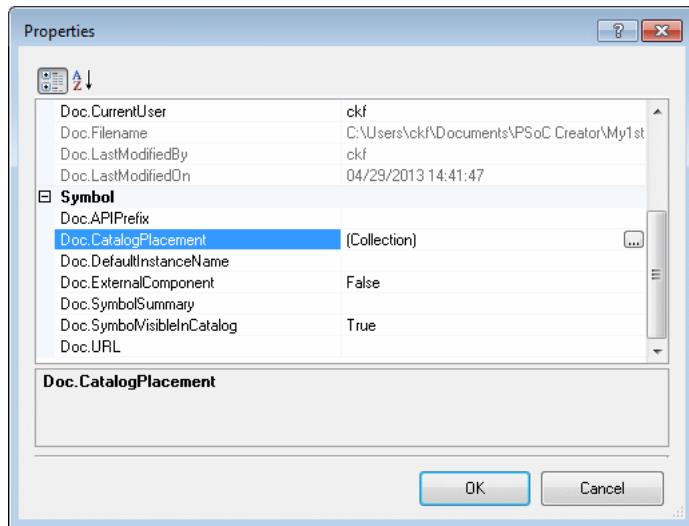
Notice that the symbol has a text box above it with shifter_N. This is an instance text label that allows you to name a Component when it is instantiated in a design. For more information, see [Working with Text](#). You can change the default instance name using the symbol document property Doc.DefaultInstanceName in the [Properties dialog](#).

Specify Placement in Component Catalog

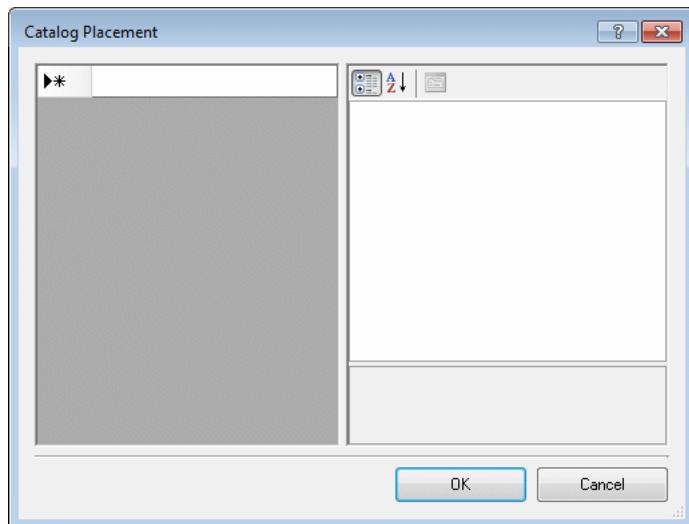
When complete, Components will display in the Component Catalog. If you wish, you can control how they are displayed.

1. Right-click on the symbol canvas and select **Properties** to open the Properties dialog.

2. Click in the **Doc.CatalogPlacement** field to show the ellipsis [...] button.



3. Click the ellipsis [...] button to open the Catalog Placement dialog.



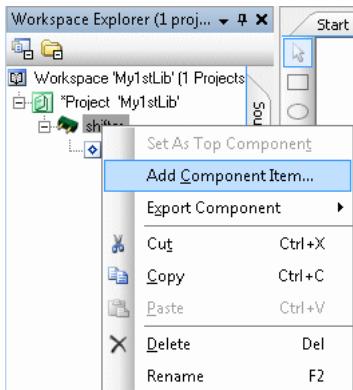
4. Click in the right column of the first row next to ►*, and enter:
Example/LogicCircuits/
5. Click **OK** to close the Catalog Placement dialog.
6. Click **OK** to close the Properties dialog.
7. Click **Save**  to save your symbol file.

See [Defining Catalog Placement](#) for more information about this dialog.

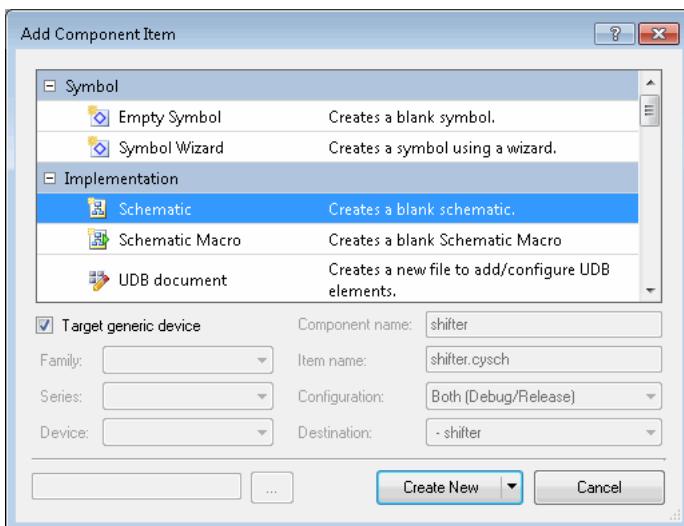
Add a Shifter Implementation

In order for your symbol to display in the Component Catalog, you must have an implementation. For this tutorial, we will add a schematic.

1. Right-click on the Component in the Workspace Explorer and select **Add Component Item...**



The Add Component Item dialog displays.



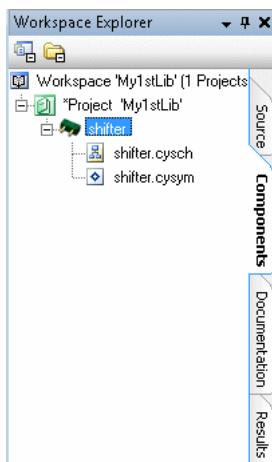
2. Under **Implementation**, click the Schematic icon.
3. Under **Target**, leave the selection on **Generic Device**.

Note The schematic will inherit the Component name, and it cannot be changed.

4. Click **Create New**.

Note The [Select Sheet Template dialog](#) may display for you to choose a canvas template. If so, click on the desired template, and click **OK**.

Your project shows the schematic added to your project. The Component displays in the Workspace Explorer tree, with a schematic file (.cysch) shown in addition to the symbol file.

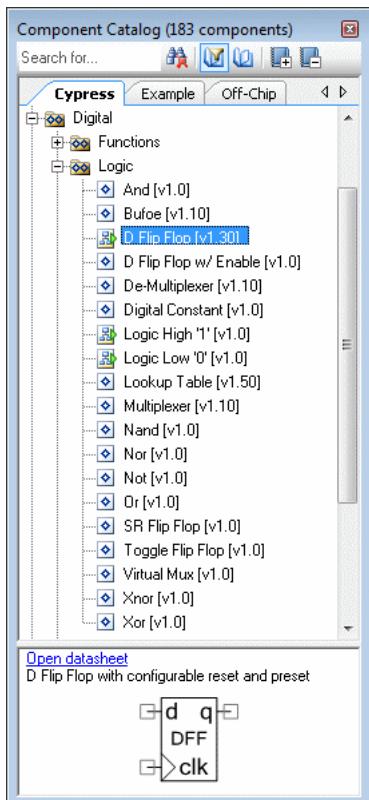


The Schematic Editor opens the .cysch file and displays an empty canvas, along with the Component Catalog. For more information see [Schematic Editor](#).

Complete the Shifter Schematic

Now that you have an empty canvas, you need to draw the schematic to implement your symbol.

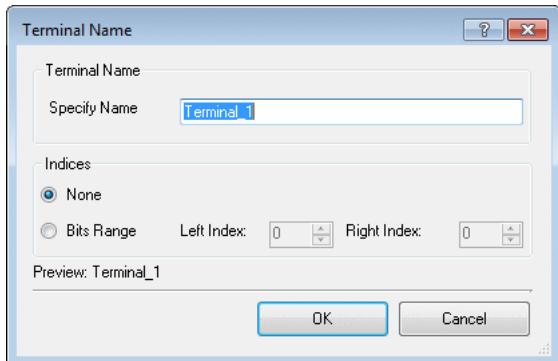
1. In the Component Catalog, expand the Digital > Logic tree.



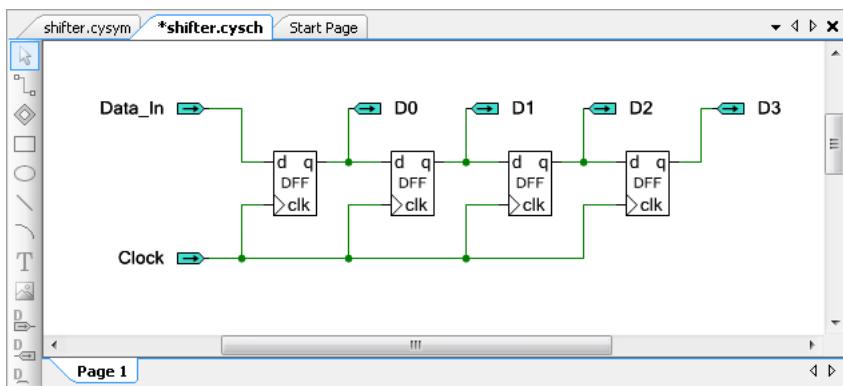
- Click and drag a D Flip Flop onto your canvas.

- Add a total of four D Flip Flop Components.
2. From the [Design Elements Palette](#), select the **Digital Input** terminal , and click the canvas to place the terminal.

The [Terminal Name dialog](#) opens. See also [Working with Schematic Terminals](#).



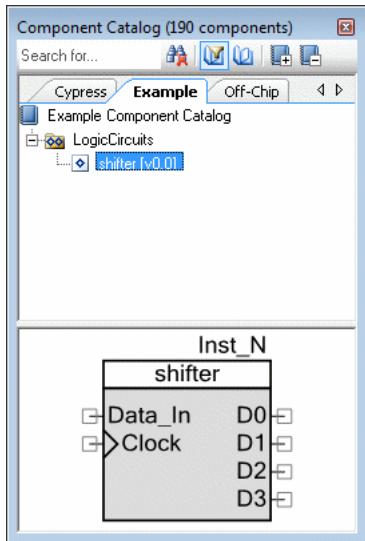
3. In **Specify Name**, type "Data_In" and then click **OK**.
4. Repeat the process to add one additional **Digital Input** and four **Digital Output** terminals; name the additional terminals Clock, D0, D1, D2, and D3, respectively.
5. Click the **Draw Wire** tool .
6. Connect the terminals to the logic gates similar to the following image:



See also [Working with Wires](#).

7. Click **Save**  to save your schematic file.

When you have completed the process, the Component will be listed the Component Catalog under the tab named Example.



Using the Library:

Once you have created a library with one or more Components, you can use those Components as library elements in your design projects. Go to the [Basic Hierarchical Design](#) tutorial to see how to use your Library in a design.

Basic Hierarchical Design

PSoC Creator allows you to create Components and reuse them. That is the basic definition of a hierarchical design; Components and designs become building blocks for additional Components and designs.

This tutorial will show you how to create an 8-bit shifter using the [LogicCircuit Library](#) you created in a previous tutorial. This is **only an example** intended to show you how to use Components you created in a hierarchical design. For more detailed instructions about creating Components, refer to the [Component Author Guide](#).

Note If you prefer not to create a new project, you can open the completed code example for this tutorial, named "Shifter" using the [Find Code Example](#) dialog. A link to the dialog is located on the PSoC Creator Start page.

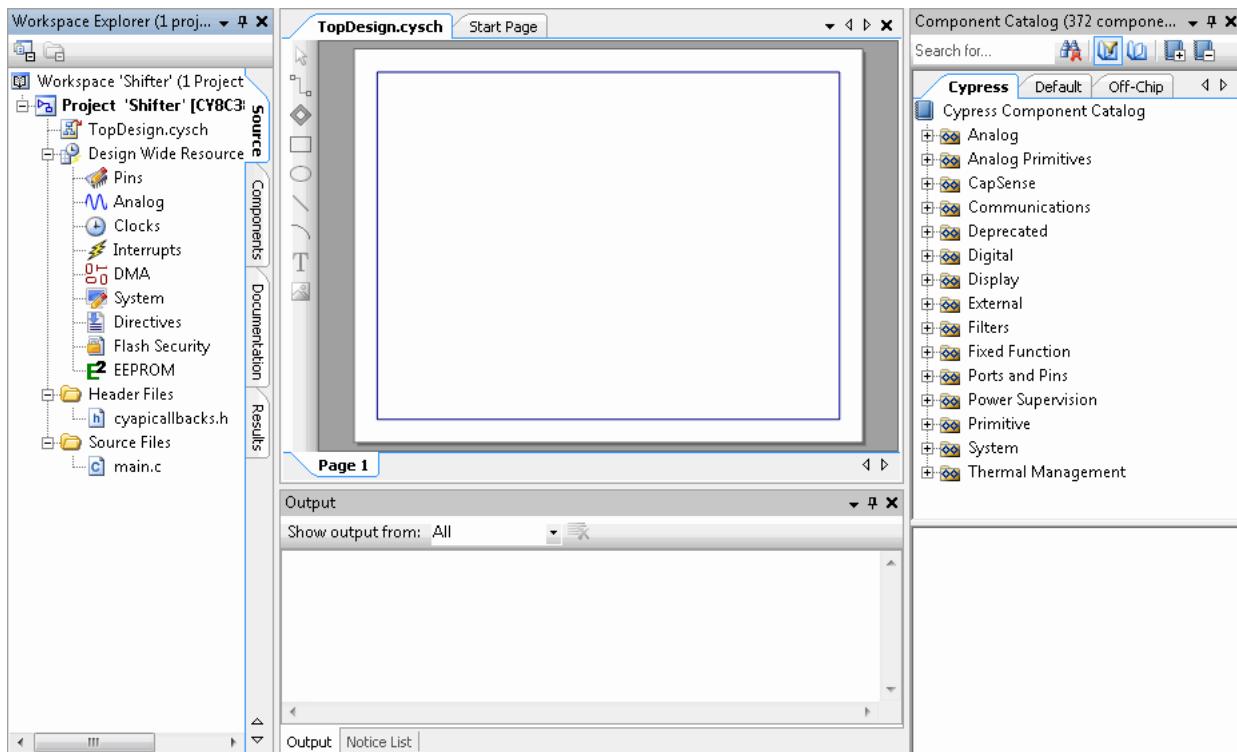
Create a New Project:

The first step is to create the basic design project:

1. If there is currently an open workspace, select **Close Workspace**  from the **File** menu.
2. From the **File** menu, select **New > Project** or click  to open the [New Project wizard](#).
3. For **Target device**, select the default PSoC 3 device, or select the specific device you want to use. For this project, we are using the default PSoC 3 device CY8C3866AXI-040. If you select a different device, then you will need to adjust your pin settings accordingly.
4. In **Name**, type the name of your project, for example: "Shifter."

5. In **Location**, type the path where you want the project to be saved, or click [...] and navigate to the appropriate directory.
6. Click **Finish**.

PSoC Creator creates your project and adds files and folders to the Workspace Explorer shown in the **Source** tab. The [Schematic Editor](#) displays the top-level schematic file as a document window, and the [Component Catalog](#) opens to display a list of Components to use in your design.

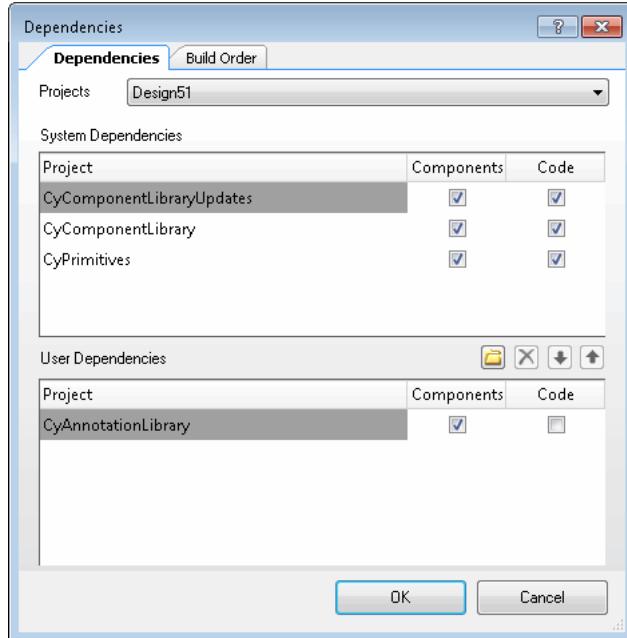


Add the Logic Circuit Library:

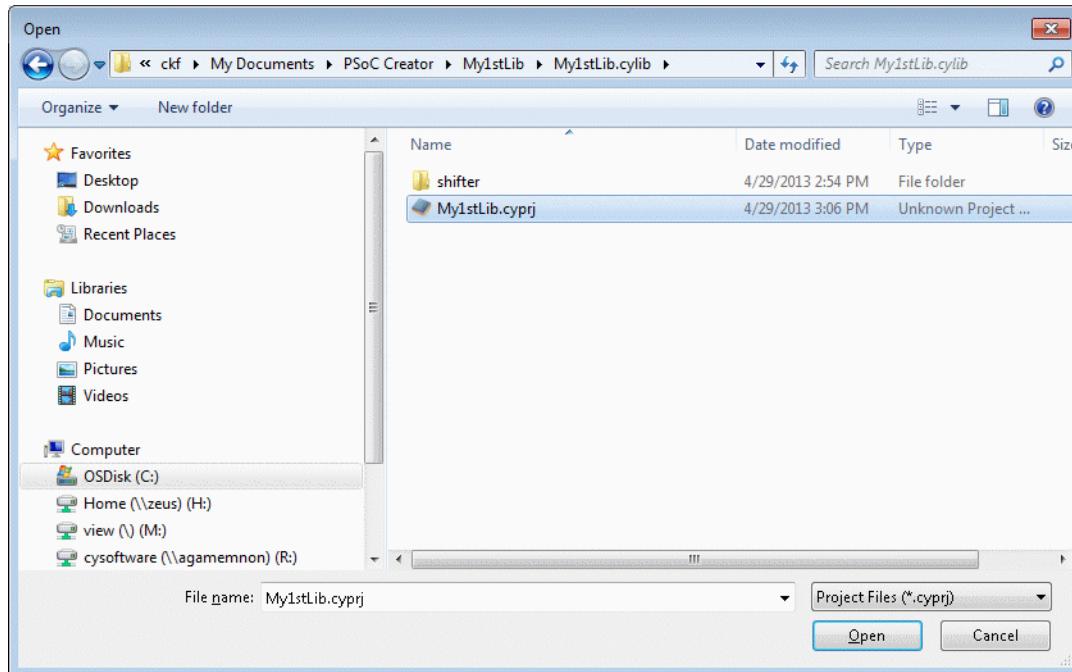
By default, all new designs come with a pre-defined library of Components for you to use. These Components are shown in the [Component Catalog](#).

To add the Logic Circuit library, you need to add the library project to your library search path.

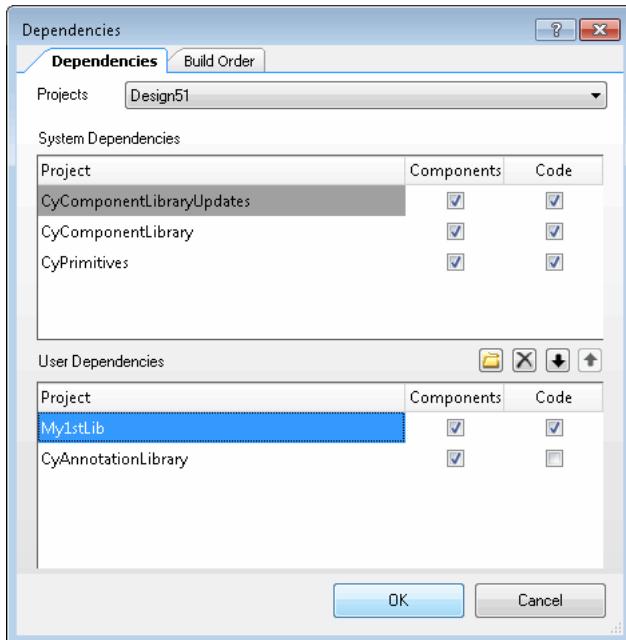
1. From the **Project** menu, select **Dependencies...** to open the [Dependencies dialog](#).



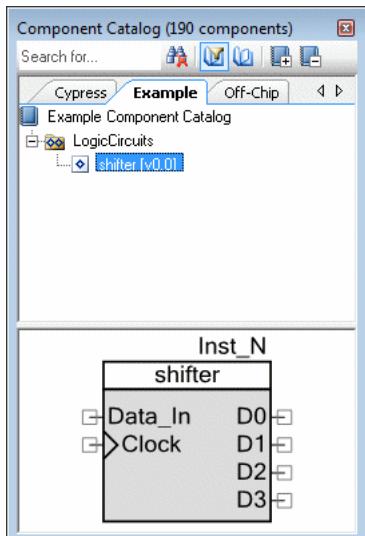
2. Under **User Dependencies**, click **New Entry**  to open a file browser dialog, navigate to find the **My1stLib.cyprj** file containing the library, and click **Open**.



3. Notice that the library project is listed under **User Dependencies > Project** and click **OK** to close the Dependencies dialog.



The Component Catalog now has a new tab named Example containing the shifter Component.



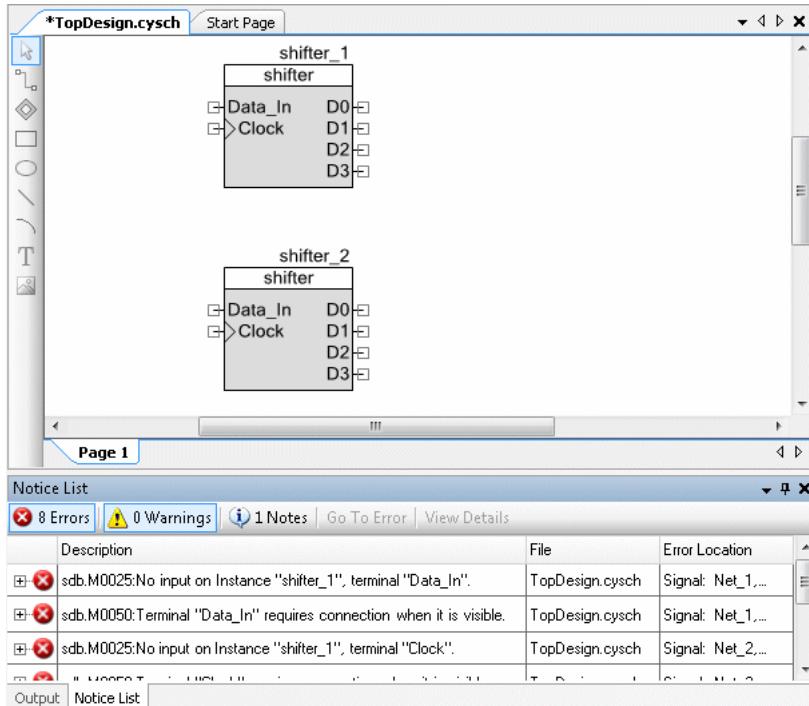
Complete the Design:

Now that you have added the library Components, you can use them to create the carry-ripple adder design.

1. In the Component Catalog, click the **Example** tab, expand the "LogicCircuits" folder, then click and drag the shifter Component onto your schematic canvas; add a second shifter Component.

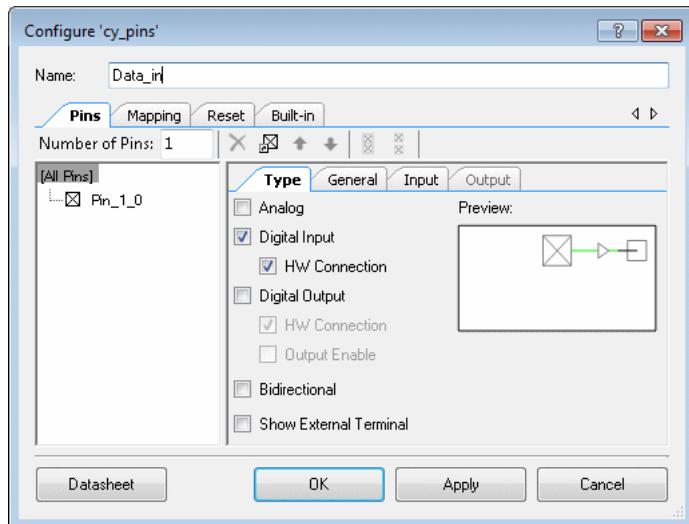
Notice that as you place Components, the Notice List window displays a list of connection errors. This is part of the dynamic rule checking (DRC) system. It's a cue that your design has errors, which you will resolve as you complete the design. For more information about this window, see [Notice List Window](#).

Notice also that the Components are named "shifter_1" and "shifter_2."



2. In the Component Catalog, click the **Cypress** tab, expand the "Ports and Pins" folder and drag a Digital Input Pin onto your schematic canvas; also add eight Digital Output Pins.

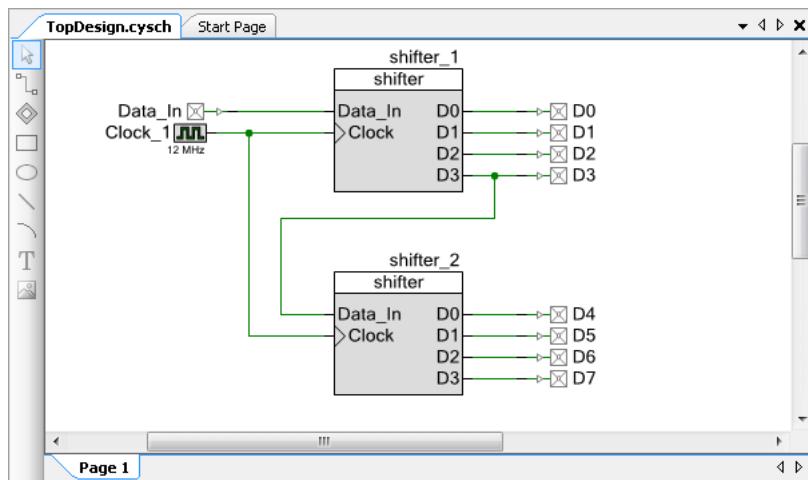
3. Double-click the Digital Input Pin to open the Configure dialog, and change the **Name** to Data_In.



4. Click **OK** to close the dialog.
5. Rename the Digital Output Pins to D0 through D7.
6. In the Component Catalog, expand the "System" folder and drag a Clock onto your schematic canvas.
7. Arrange the Components and terminals on your canvas as appropriate.
8. Click the **Draw Wire** tool .
9. Connect the pins and the clock to the shifters.

See [Working with Wires](#), [Drawing Buses](#), and [Wire Labels and Names](#), as needed for instructions about different techniques,

When you're finished making all the connections, your schematic should look similar to the following image:



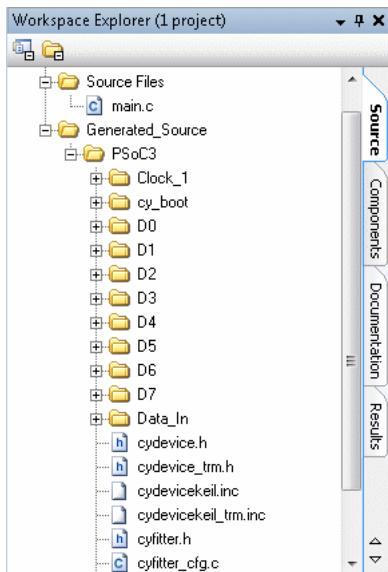
Notice that all the errors have cleared in the Notice List window.

Tip You can copy and paste similar wires to different locations; therefore, if you equally space items in your design, you can replicate them more easily.

10. From the Build menu, select **Build Shifter**.

The Output window shows messages indicating a successful build.

The Workspace Explorer shows a Generated Files folder with generated design files.



Close the Design:

From a basic perspective, this 8-bit shifter is an example of a hierarchical design. The design was built with two 4-bit shifters, which were built on D Flip Flops. You could use this shifter to create a Component for use in another design, and that design could be used in another, and so on. You will use the same basic principles and tools for every design you create with PSoC Creator.

To close this design (and its project and workspace), select **Close Workspace**  from the **File** menu.

How To

This section contains various "how to" topics to help you learn how to get the most from PSoC Creator. This section is broken down into various categories, as follows:

General PSoC Creator Tasks:

- [Creating a New Project](#)
- [Finding Code Examples](#)
- [Opening an Existing Project](#)
- [Adding a New Workspace/Project Item](#)
- [Adding an Existing Project Item](#)
- [Assigning a Core to File in a Multi-Core Design](#)
- [Archiving a Workspace/Project \(Bundling\)](#)
- [Copying a Project](#)
- [Generating a Project Datasheet](#)
- [Creating a New File](#)
- [Opening an Existing File](#)
- [Creating Folders](#)
- [Copying a Project](#)

General Design Entry Tasks:

- [Design Entry Options](#)
- [Working with Text](#)
- [Using Text Substitution](#)
- [Working with Lines](#)
- [Working with Shapes](#)
- [Zooming](#)
- [Scrolling](#)
- [Selecting a Device](#)
- [Using the Notice List Window](#)

Schematic Editor Tasks:

- [Creating a New Schematic](#)
- [Using the Component Catalog](#)

- [Configuring Components](#)
- [Working with Wires](#)
- [Drawing Buses](#)
- [Working with Schematic Terminals](#)
- [Adding a Schematic Page](#)
- [Updating Components](#)
- [Importing a Component](#)

Text Editor Tasks:

- [Writing Code](#)
- [Using the Text Editor](#)
- [Using Find Replace](#)
- [Using Wildcards](#)
- [Using Regular Expressions](#)
- [Using Go To](#)

Framework:

- [To Float Tool Windows](#)
- [To Dock Tool Windows](#)
- [To Use Tabbed Documents](#)
- [To Move Tool Windows](#)
- [To Auto-Hide Tool Windows](#)

Debugger Tasks:

- [Using the Debugger](#)
- [Using Program/Debug Options](#)
- [Using the Breakpoints Window](#)
- [Using Debugger Menu Commands](#)
- [Selecting a Default Compiler](#)

Symbol Editor Tasks:

- [Creating a Symbol](#)
- [Using the Symbol Wizard](#)
- [Adding a Component Item](#)
- [Working with Component Terminals](#)
- [Creating Symbol Parameters](#)
- [Defining Catalog Placement](#)
- [Exporting a Component](#)

3 Understanding PSoC Creator



PSoC Creator provides a PSoC hardware/software co-design environment, with software development tools, a graphical design editor, a device selector, and various features for project management. There are many concepts referenced within this PSoC Creator help with which you may not be familiar. This section helps you have a deeper understanding of PSoC Creator. It contains the following sub-sections:

- [Concepts](#)
- [General Tasks](#)
- [Framework](#)

Concepts

To help you better understand PSoC Creator, you should become familiar with the following terms and concepts:

- [Workspace/Project](#)
- [Project Types](#)
- [Component/Instance](#)

Workspace/Project

The PSoC Creator integrated development environment (IDE) provides two containers to help you manage items in your designs: workspaces and projects.

- **Workspace** – A workspace is the top-level container within PSoC Creator; it contains one or more projects that you can open, close, and save together. You can only have one workspace for any given PSoC Creator workspace file (.cywrk).
- **Project** – A project contains multiple items that represent your design, such as schematics, design-wide resources, source code, and hex files. The types of items contained within a project vary according to the [project type](#). A project is always part of a workspace. You can create projects in an existing workspace or you can create a new workspace as part of creating a new project.

PSoC Creator provides workspace folders to organize related projects into groups and then perform actions on those groups of projects. You can view and manage your workspace, projects, and their associated items using the [Workspace Explorer](#). A workspace and projects allow you to use the IDE in the following ways:

- Manage settings for your workspace as a whole or for individual projects.

- Use Workspace Explorer to handle the details of file management while you focus on items that make up your design.
- Add items that are useful to multiple projects in the workspace or to the workspace without referencing the item in each project.
- Work on miscellaneous files that are independent from the workspace or projects.

When you create a multi-project workspace, the first design project created becomes the active project, by default. The active project appears in bold font in Workspace Explorer and is the project that runs when you click Start on the Debug menu. You can build either a single project within the workspace or multiple projects in the workspace. You can also specify which workspace projects you wish to exclude from builds. For more information, see [Building a PSoC Creator Project](#).

See Also:

- [Project Types](#)
- [Workspace Explorer](#)
- [Building a PSoC Creator Project](#)

Project Types

A PSoC Creator project contains multiple items, such as schematics, Components, design-wide resources, source code, and hex files. PSoC Creator provides two types of projects: design and library.

- **Design Project** – A design project is used to create and modify designs. With a design project, select and configure the Components for your device in a schematic. Next, set up design-wide resources, such as clocks and interrupts. Then, write the C code for your application. Finally, you build (and debug) the project to generate the hex file and program the device. When you first create a design project, PSoC Creator creates the project/workspace files and directory structure, as well as the top-level schematic, *main.c* shell file, and a design-wide resources file (.cydwr).
- **Library Project** – A library project is a collection of one or more Components and the associated source code. With a library project, you can develop Components that will be elaborated in a design, as well as reused in many designs. Library Projects can also be used to create static libraries that can be linked in to a design. Each library can serve either purpose (or both at the same time). Component development includes creating the graphic symbol, defining parameters, and specifying validation requirements. When you first create a library project, PSoC Creator creates the project/workspace files and directory structure. Library project(s) are included in PSoC Creator as [Dependencies](#) to determine which Components are available for your designs in the [Component Catalog](#).

Project names have a .cyprj extension, such as *ProjectName.cyprj*. In addition, project files must always be located in a directory named either *ProjectName.cydsn* (design) or *ProjectName.cylib* (library).

See Also:

- [Component](#)
- [Component Catalog](#)
- [Dependencies](#)

Component/Instance

A Component is a collection of files, such as a symbol, schematics, APIs, and documentation that defines functionality within the PSoC Device. Examples of Components include a timer, counter, and a mux. An instance is a Component that has been selected from the [Component Catalog](#) and used in a design. You can have multiple copies – or instances – of a Component in a design, as long as the selected device can support it. You can also create and reuse your own Components/instances.

Files that make up a Component/instance include the following:

- **Symbol** – A symbol contains the basic definition of a Component. It contains the top-level picture shown in the Component Catalog, as well as the parameter definitions. There can be only one symbol in a Component.
- **Schematic** – A schematic defines how a Component has been implemented visually. A schematic can be generic for any PSoC device, or it can be architecture, family and/or device specific.
- **API** – Application Programming Interface. APIs define how to interact with a Component using C code. They can be generic for any PSoC device, or they can be architecture, family and/or device specific.
- **Verilog** – Verilog can be used to define the functionality of a Component implemented in Verilog. There will only be one Verilog file in any given level of a Component. Verilog files found at different levels of the Component, such as at an architecture, family and device level, may not refer to each other.
- **Control File** – The control file contains directives to the code generation module. For more information, see [Control File](#) and [Directives](#).
- **Documentation** – The documentation of the Component is generally its datasheet.
- **CyPrimitive** – A CyPrimitive is a basic Component item, such as a logic gate, interrupt, or DMA.

Note A symbol need not be primitive -- it could be a primitive for some device, but implemented out of logic and software for another device.

See Also:

- [Component Catalog](#)
- [Control File](#)
- [Directives](#)

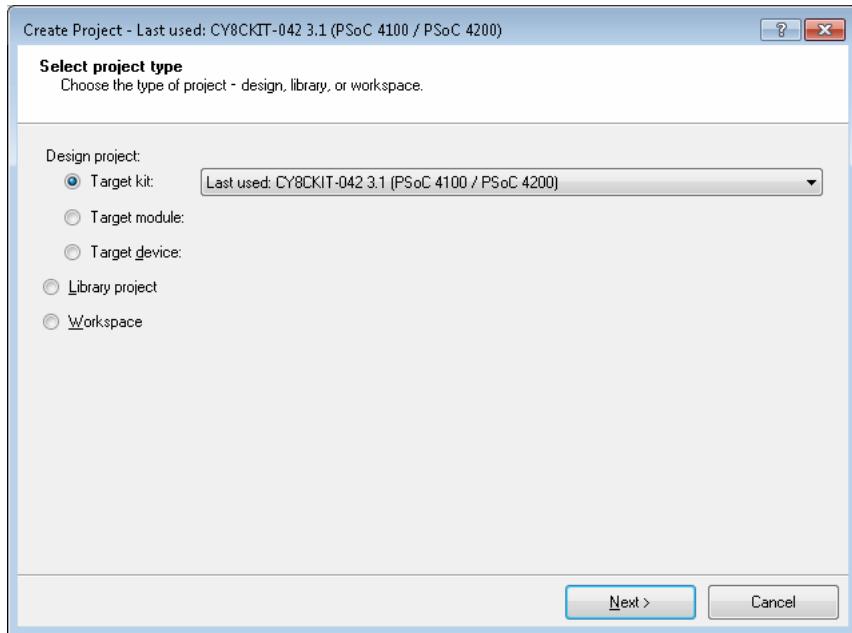
General Tasks

The following are some of the general tasks you will perform with PSoC Creator:

- [Creating a New Project](#)
- [Opening an Existing Project](#)
- [Adding a New Workspace/Project Item](#)
- [Adding an Existing Project Item](#)
- [Assigning a Core to File in a Multi-Core Design](#)
- [Writing Code](#)
- [Archiving a Workspace/Project](#)
- [Saving a Project As](#)
- [Generating a Project Datasheet](#)
- [Generating Description Files](#)
- [Copying a Project](#)
- [Selecting a Default Compiler](#)
- [Creating a New File](#)
- [Opening an Existing File](#)
- [Creating Folders](#)

Creating a New Project

The Create New Project wizard is used to create new PSoC Creator projects.



Use this wizard to:

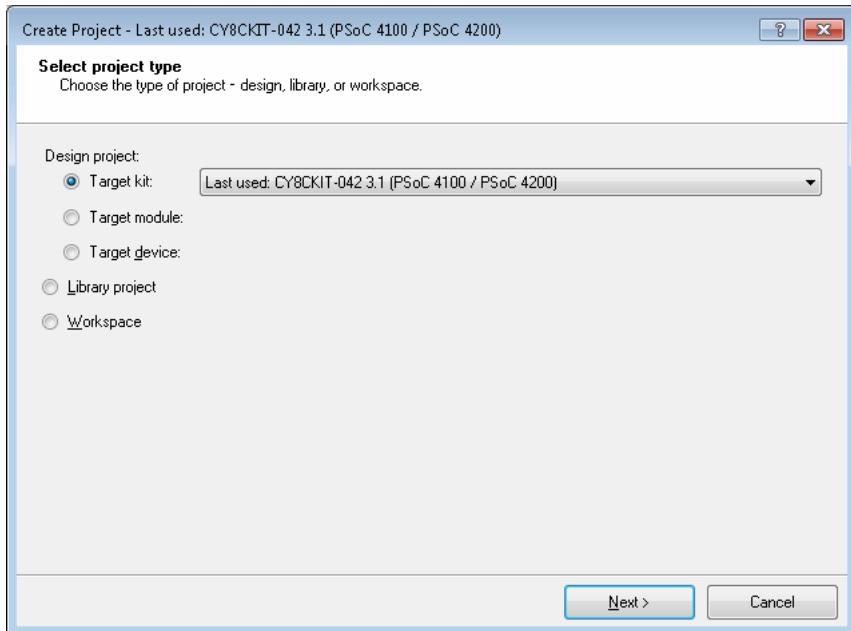
- select the kit, module, or device
- select the type of project to create; see [Project Types](#)
- specify the project name and location
- choose whether to create a new workspace or add to an existing workspace; see [Workspace/Project](#)

To Open this Wizard:

From the **File** menu, select **New > Project...**  or click **Create New Project** on the PSoC Creator Start page.

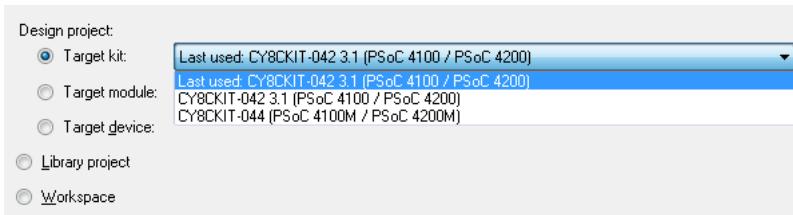
To Create a New Project:

Select project type

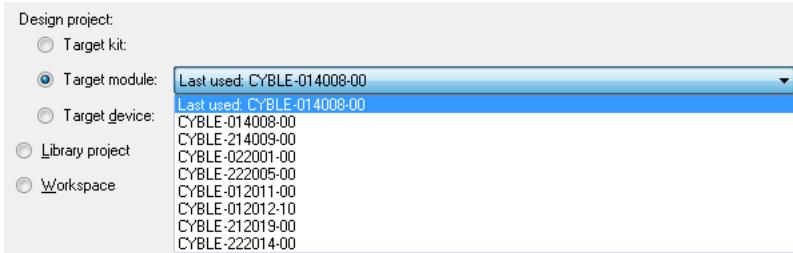


Choose one of the following:

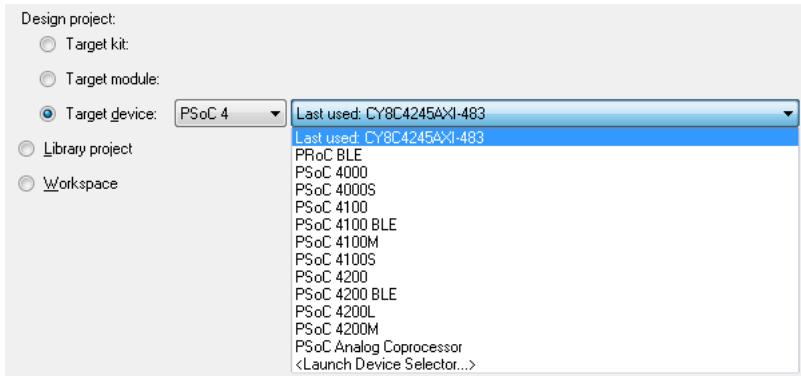
- **Target kit:** Use this option to select a specific kit, or the last used kit.



- **Target module:** Use this option to select a specific module, or the last used module.



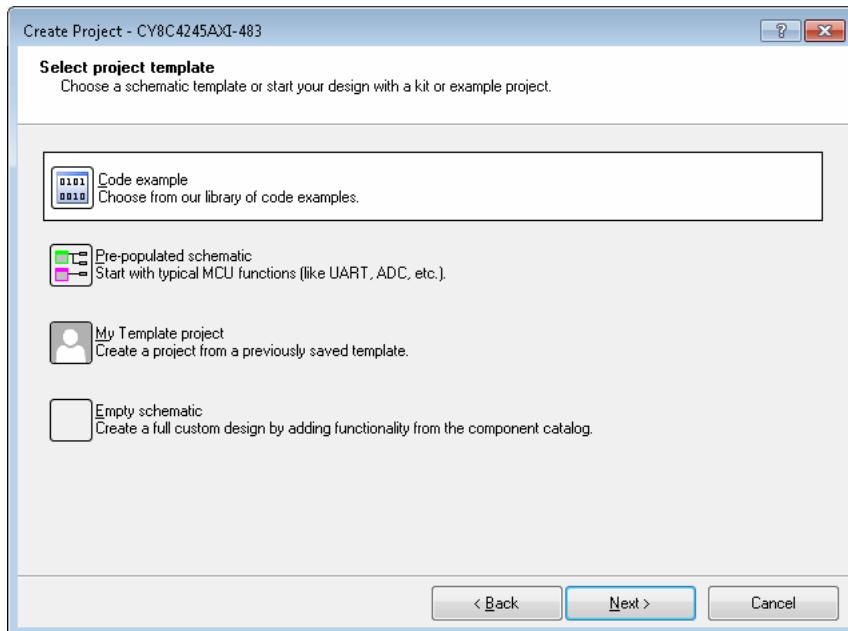
- **Target device:** Use this option to select a device family and series, the last used device, or to launch the [Device Selector](#).



- **Library project:** Selection this option to create a library project.
- **Workspace:** Select this option to create an empty workspace.

Click **Next >**.

Select project template (Design Projects Only)

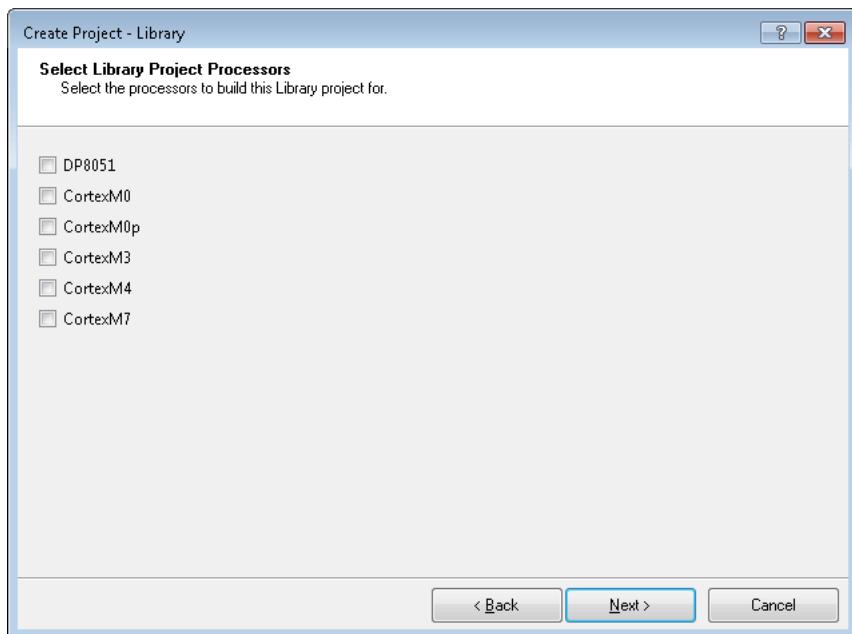


Choose one of the following:

- **Code example:** Choose this option to select a starter project from a list of available examples. See [Code Examples](#).
- **Pre-populated schematic:** If available, select this option to create a design with Components already placed on the design schematic.
- **My Template project:** If you have previously copied a project to My Templates, select this option to create a new project based on a selected My Template project. See [My Templates](#).
- **Empty schematic:** Choose this option to create a blank schematic canvas (not available for PRoC BLE designs).

Click **Next >**.

Select Library Project Processors (Library Projects Only)

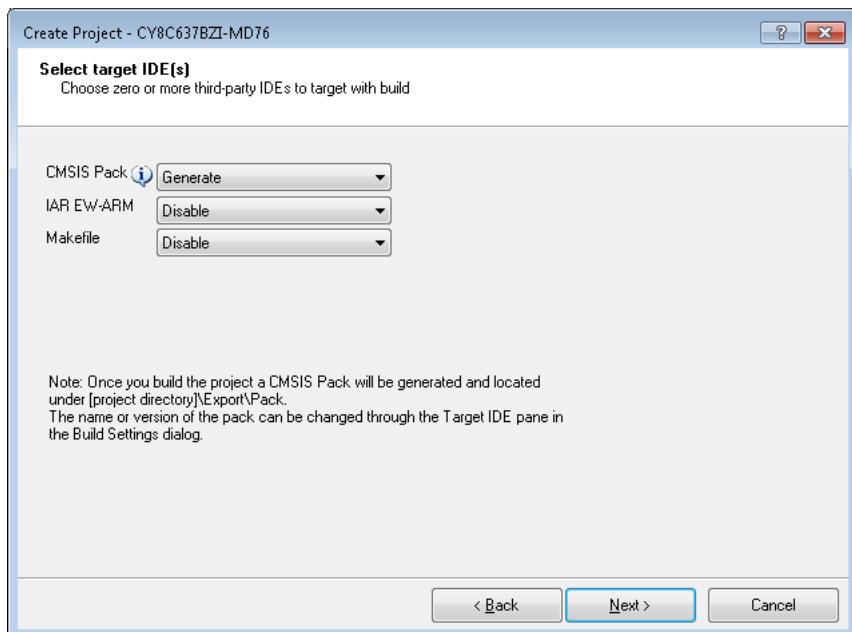


Choose one or more processors for which this library will be built. You can also select more and different processors on the [Build Settings dialog](#) for library projects.

Note If you select DP8051 (for PSoC 3) you cannot select any Cortex processor, and vice-versa. You will need to create separate library projects.

Click **Next >**.

PSoC 6 Only Select Target IDEs

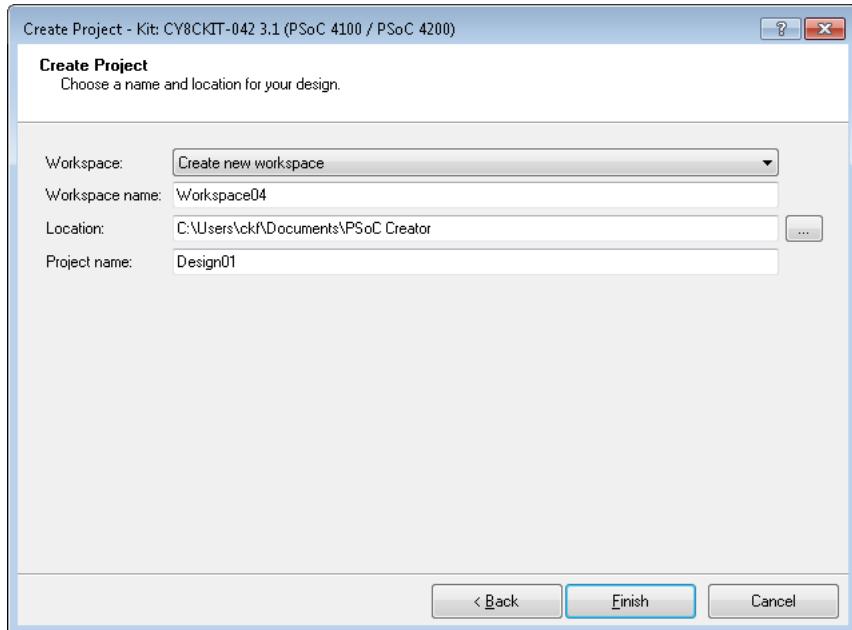


For PSoC 6 devices only, the Select Target IDEs step allows you to select one or more IDEs for which to generate files. If you select to generate files for **CMSIS Pack** (for Eclipse or Arm MDK) a note will display indicating where

the files will be located, as well as information to change the name and version of the pack. See [Build Settings > Target IDEs](#) and [Generating PSoC 6 Files for 3rd Party IDEs](#) for more information.

Click **Next >**.

Create Project (All Projects)



Workspace – Select the appropriate option:

- **Create new workspace:** Use this option to create a new workspace, and specify a location to save the workspace.
- **Add to current workspace:** If creating a new project in a workspace that is already open, select this option to add the new project to the existing workspace.

Workspace name – For a new workspace, type a name for the workspace.

Location – Specify a location for the project/workspace.

Project name – Type a name for the project.

Note The project name cannot exceed 80 characters.

Note If creating a blank workspace, this page will not have a **Project name** option.

Click **Finish**.

- If you create a design project, PSoC Creator will create files and open the [Schematic Editor](#) by default.
- If you create a library project or empty workspace, PSoC Creator will create the library or empty workspace infrastructure in the [Workspace Explorer](#).

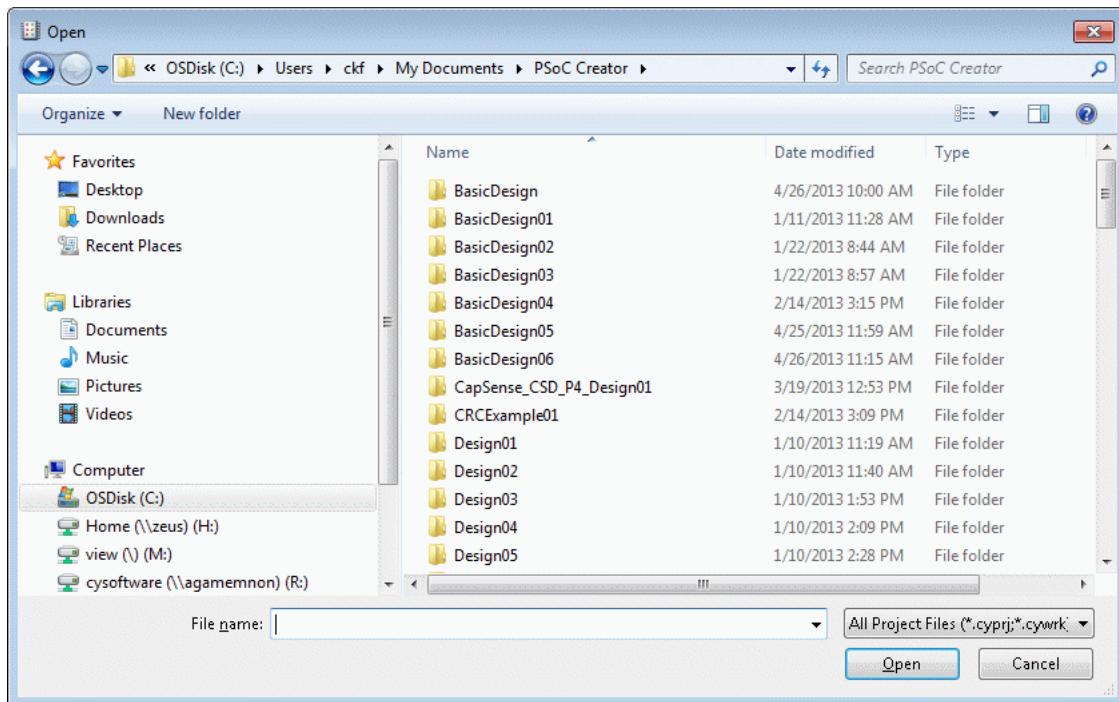
See Also:

- [Project Types](#)

- [Workspace/Project](#)
- [Device Selector](#)
- [Schematic Editor](#)
- [Workspace Explorer](#)
- [Opening an Existing Project](#)

Opening an Existing Project

The Open Project dialog is used to open an existing PSoC Creator workspace/project.



Use this dialog to browse and select the workspace file (.cywrk) or project file (.cyprj) to open. You can open an existing project that you created or one of the many examples provided with PSoC Creator.

To Open this Dialog:

From the **File** menu, select **Open > Project/Workspace...**  or click **Open Existing Project** on the PSoC Creator Start page.

To Open a Project:

1. Navigate to the appropriate directory where the project to open is located.
2. Select the desired project and click **Open**.

The project opens in PSoC Creator, and it is shown in the [Workspace Explorer](#).

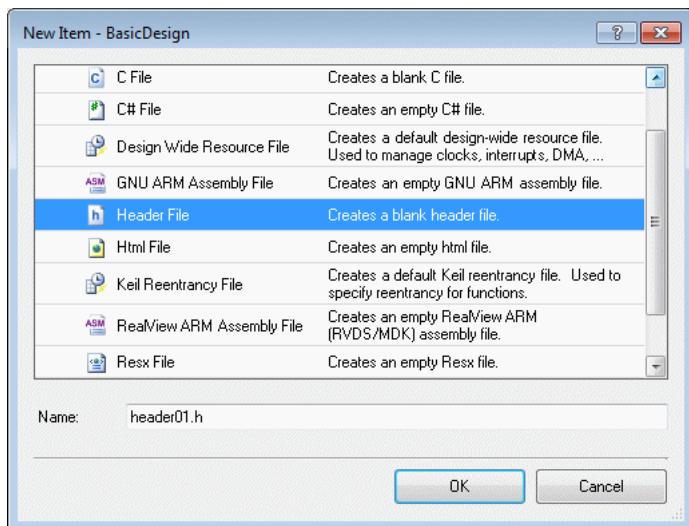
Note You can also open recent projects without this dialog using **Recent Projects** on the Start page or on the **File** menu.

See Also:

- [Workspace/Project](#)
- [Workspace Explorer](#)
- [Creating a New Project](#)

Adding a New Workspace/Project Item

The New Item dialog allows you to add new items to your workspace/project.



The dialog provides templates of the different types of items you can add. Source files will be compiled as part of a build; non-source files will be ignored.

Note You cannot use this dialog to create new items for Components. Instead you must use the [Add Component Item dialog](#).

To Open the Dialog:

1. In the [Workspace Explorer](#), under the **Source** tab, select the level at which you want to add the item: workspace, project, or folder.
2. From the **Project** menu, select **New Item...**

You can also right-click on a workspace, project, or folder and select **Add > New Item** from the context menu.

To Add an Item:

1. In the **Templates** area, select the icon for the type of item to add to your workspace/project. Currently the available templates include:

- **8051 Keil Assembly File** – Used to create an assembly file that will be compiled when you select the Keil tool-chain.
- **C File** – Used to create a standard C source file.
- **C# File** – Used to create a standard C sharp source file.
- **Design Wide Resource File** – Used to create a file for editing design level resources, such as interrupts, clocks, etc.
- **GNU Arm Assembly File** – Used to create an assembly file that will be compiled when you select the GNU Arm tool-chain.
- **Header File** – Used to create a standard C Header file.
- **HTML File** – Used to create an HTML file for documentation purposes.
- **Keil Reentrancy File** – Used to mark APIs as reentrant. See [Reentrant Code in PSoC 3](#).
- **RealView Arm Assemly File** – Used to create an assembly file that will be compiled when you select the RealView Arm tool-chain.
- **Resx File** – Used to create a resource file.
- **Text File** – Used to create an empty text file.
- **XML File** – Used to create an empty XML file for whatever you want it for.

There are two buttons on the top right side of the dialog to change the size of the icons. Below the **Templates** area is a text box that displays a brief description for each Component item.

2. Specify a file name for the item in the **Name** field.
3. Click **OK**.

The item is added at the location you selected in the Workspace Explorer, and an empty file of the type you added opens in the [Text Editor](#) as a tabbed document.

See Also:

- [Adding an Existing Project Item](#)
- [Workspace/Project](#)
- [Adding a New Component Item](#)
- [Workspace Explorer](#)
- [Text Editor](#)

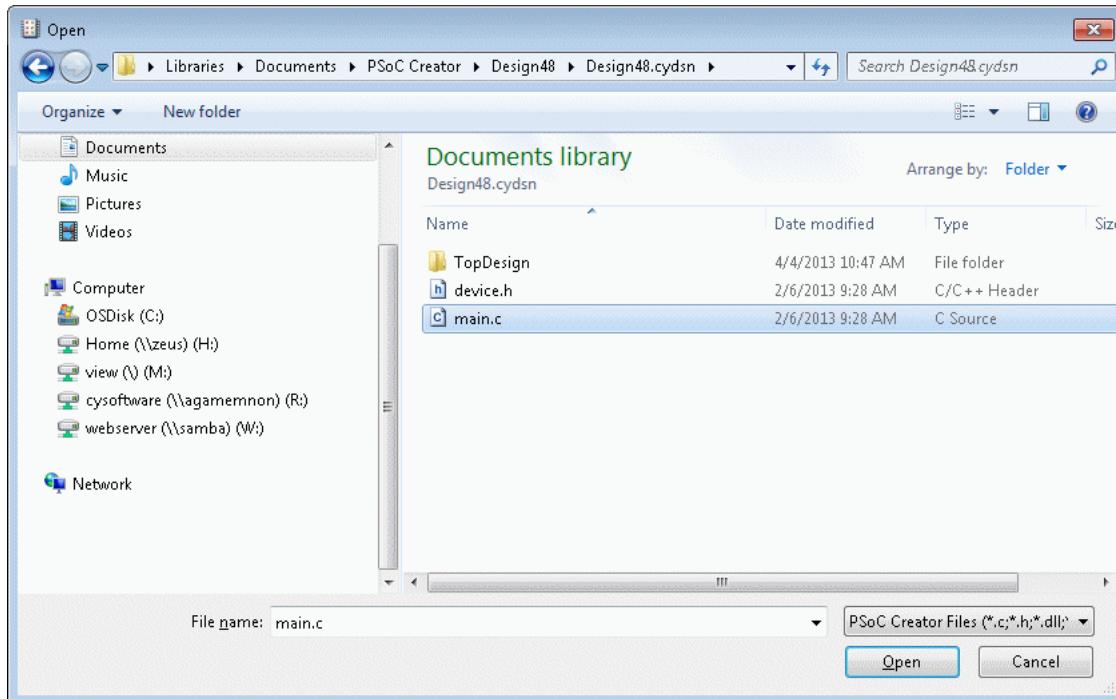
Adding an Existing Project Item

PSoC Creator provides the ability to add an existing project item, such as an assembly or header file to a current project. This could be a file used in other projects, or a file that another person created as part of a design team.

To Add an Existing Item:

Right-click on a project in the Workspace Explorer and select **Add > Existing Item**. You can also select **Existing Item** from the **Project** menu.

Either method opens the Open dialog.

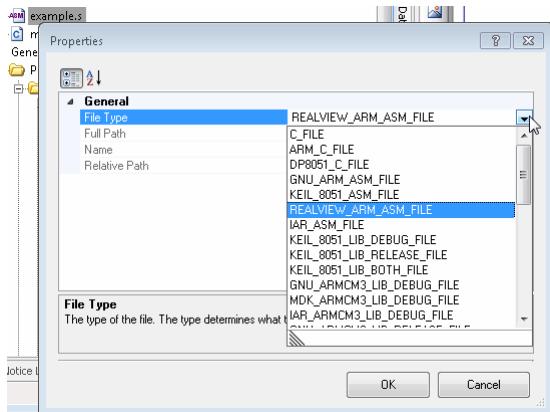


Use this dialog to browse and select item to add to the current project, and then click **Open**.

Notes

Some file types, such as *.cyre, are copied from their existing location to the current project's cydsn folder; other file types, such as header files are not. Be aware of which files are not copied for portability issues.

If you add an existing assembly file, you may need to select the correct file type from the drop-down menu in the Properties dialog.

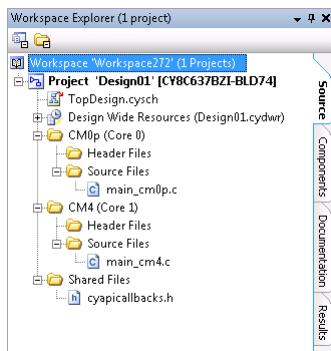


See Also:

- [Adding a New Workspace/Project Item](#)
- [Workspace/Project](#)
- [Adding a New Component Item](#)
- [Workspace Explorer](#)

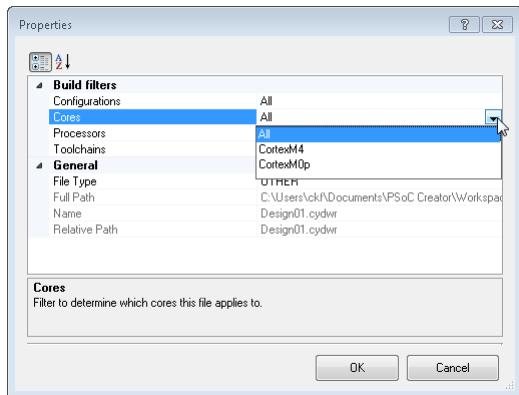
Assigning a Core in a Multi-Core Design

When you create a new multi-core PSoC 6 project, the initial project structure in the [Workspace Explorer](#) includes a folder for each core (CM0p and CM4), as well as a folder for source files that are to be built for all cores on the device (Shared Files). By default, these folders and any files in them will be built for the specific cores: CM0p for CortexM0p, CM4 for CortexM4, and Shared for both.



Cores Property

All files and folders for multi-core designs have a category of properties named **Build filters**. Among these properties is one named **Cores**, and it controls which source files are built for each core. If you open a folder's [Properties](#) dialog, you can modify the **Cores** property value. Setting the property value to "All" will cause all source in that folder to be built for all available device cores.



Notes:

- When adding a new file to a folder, that file's **Cores** property value will default to its parent folder's value.
- When adding a new folder at the project level, that new folder's **Cores** property value will default to a value of "All."
- When adding a new folder at another level, that new folder's **Cores** property value will default to its parent container's value.
- When moving a file from one folder to another, that file's **Cores** property will take on the value of its new parent folder.

Processor Property

Another **Build filters** property is named **Processors**, and it is used to differentiate between multiple cores of the same type (for example, a device with multiple CM4 cores, when you want a file to be built for only one of those multiple CM4 cores).

See Also:

- [Workspace Explorer](#)
- [Properties](#)
- [Building a PSoC Creator Project](#)

Writing Code

PSoC Creator provides a [Code Editor](#) to write C code for your designs. You can use this tool or any other preferred code editor you prefer. If you use the PSoC Creator Code Editor, files you create, edit, and save as part of your design will be integrated into the Save and Build processes automatically. Files you edit externally can still be included in the Build process, but if you make changes external to PSoC Creator, make sure you save those changes using your preferred tool.

Embedded Programming with C - Beginners Resources:

See the following link for information and links to resources for writing C code:

- <http://www.cypress.com/blog/psoc-creator-news-and-information/matts-tips-embedded-programming-c-beginners-resources>

Including Code in Generated Source

As described in the [Building a PSoC Creator Project](#) section, PSoC Creator generates code when you build your project/workspace. Sometimes, you may want to run or include your own code with that generated code. PSoC Creator provides two methods to do this:

- Macro Callbacks (preferred)
- Merge Regions (legacy)

Macro Callbacks

Macro Callbacks is a term defined in PSoC Creator to call user code from macros specified in a Component's generated code. These macros can be used by defining them in the user-defined header file named *cyapicallbacks.h*. This file will be included in all generated source files that offer callbacks.

A callback requires you to complete the following:

- Define a macro to signal the presence of a callback (in *cyapicallbacks.h*).
- Write the function declaration (in *cyapicallbacks.h*).
- Write the function implementation (in any user file).

To complete the example, the *cyapicallbacks.h* file would include this code:

```
#define SimpleComp_1_START_CALLBACK  
void SimpleComp_1_Start_Callback( void );
```

In any other user file, you could include *cyapicallbacks.h* and write the SimpleComp_1_Start_Callback() function.

Merge Regions

Merge Regions provide another method to insert user code, through the use of specially marked sections in generated code, such as:

```
/* `#START isr_Interrupt` */  
/* `#END` */
```

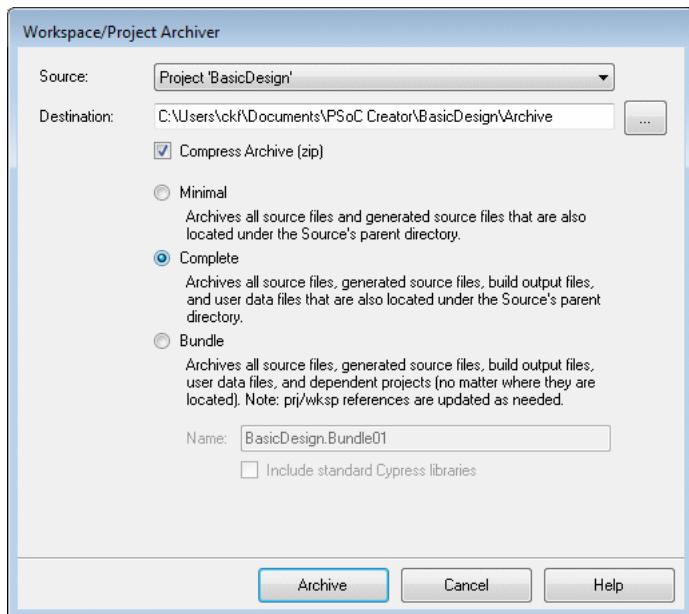
Anything you place in this region will be preserved in subsequent updates of the file. If a subsequent version of the file does not contain the same named region, the entire region from the previous file will be copied to the end of the file and placed in comments.

See Also:

- [Code Editor](#)
- [Building a PSoC Creator Project](#)

Archiving a Workspace/Project

The Workspace/Project Archiver dialog allows you to archive an entire workspace or a project from the current workspace. You can also use it to bundle an entire workspace, including dependent projects. The archive can either be zipped or not. Only the files registered with the workspace/project and any file that PSoC Creator generates can be archived.



To Open the Dialog:

1. Open the PSoC Creator workspace/project you want to archive.
2. Use one of the following methods, as applicable:
 - Select **Archive Workspace/Project...** from the Project menu
 - Right-click on a project or workspace in the **Source** tab of the [Workspace Explorer](#) and select **Archive Workspace/Project...**
 - Select **Create Workspace Bundle...** from the File menu.

Note If there are any modified files in the workspace, you will be prompted to save them before the dialog will open.

To Archive a Project/Workspace:

1. Select the **Source** workspace or project to archive.

2. Specify the **Destination** where to create the archive. Either type the path, or click [...] and browse to the appropriate directory.
 3. Select the **Compress Archive** check box to create a zip file that contains the archive; de-select to choose not to zip the archive.
 4. Select one of the following levels as desired:
 - Minimal** – This level includes project source files and generated source files. Only non-external files are archived. Non-external files are those files located under the archiving source's parent directory on disk.
 - Complete** – This level includes project source files, generated source files, all derived files (build output files), and user data files. Only non-external files are archived. Non-external files are those files that are located under the archiving source's parent directory on disk.
 - Bundle** – This level includes project source files, generated source files, all derived files (build output files), user data files, and dependant projects. External files **are** included in this level. Also, project/workspace files have their dependencies/links updated to reference the archived copies of the projects/files. This is to achieve the goal of being able to open a bundle from anywhere and it will always behave the exact same (that is, have all its references with it).
- If you choose the bundle option, a workspace will always be archived. If a workspace is selected as a source, it will be archived. If, however a project is selected, a new workspace will be created that contains only the project to archive and it will be archived. This is done because there are project dependencies that are stored on the workspace which need to be included.
- **Name** – Enter a name to rename the archived workspace. If zipped, it will also be used as the zipped file name.
 - **Include standard Cypress libraries** – Select this check box to include a copy of all Cypress libraries (CyPrimitives and CyComponentLibrary) in the archive. This will add dependencies to the archived copies, and they will be used prior to the standard Cypress libraries installed on the machine.
5. Click **Archive**. The dialog shows the progress and reports success or failure.
 - If successful, an **Open archive in Windows Explorer** check box displays to open the archive location upon clicking **OK**.
 - If not, an error message will be displayed explaining the cause of the failure.
 6. Click **OK** to close the dialog.

See Also:

- [Workspace/Project](#)
- [Saving a Project As](#)
- [Copying a Project](#)

Saving a Project As

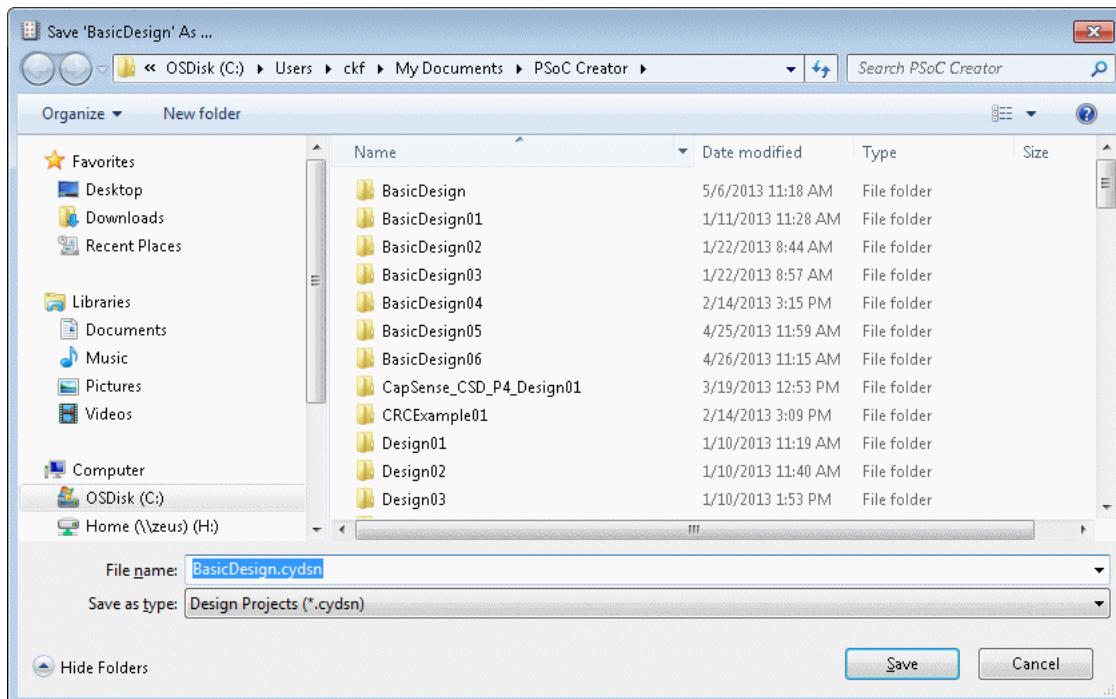
PSoC Creator allows you to save a project with another name in the same location or with any name in a different location. This is useful when you want to have a back-up project and you do not wish to use the [Archiving tool](#).

To Save a Project As:

Select the project and select **Save <project> As...** from the **File** menu.

You can also right-click on a project in the [Workspace Explorer](#) and select **Save <project> As...**.

The standard Windows Save As dialog opens to select a location and enter a name for the project to save.



See Also:

- [Archiving a Project](#)
- [Copying a Project](#)

Generating a Project Datasheet

A project datasheet is a PDF document generated from your PSoC Creator project. It includes:

- Overview of your selected PSoC chip architecture
- List of chip resources used
- Pins
- Design-wide resource settings (clocks, interrupts, DMA, flash security, etc.)
- Schematic sheets
- Components and parameter settings

The project datasheet acts as a snapshot summary of your completed design. It is useful as a hand-off document from the engineering team, who configure the hardware portion of a PSoC design, to the firmware team who are programming it. This is helpful if the firmware team is using an IDE other than PSoC Creator, without direct access to PSoC design configuration data.

To Generate a Project Datasheet:

1. The project for the datasheet to be generated must be the active project. If necessary, right-click on the project under the **Source** tab in the [Workspace Explorer](#), and select **Set as Active Project**.
2. If the project has not been built, or if there were changes made since it was last built, build the project at this time.
3. Click the Build menu and select **Generate Project Datasheet**.

If you attempt to generate a project datasheet without updating the build first, PSoC Creator will display a message and allow you to rebuild the project first.

When the datasheet has been generated, it will be listed under the **Documentation** tab in the Workspace Explorer. Double-click the file to open in the same manner as any other document.

See Also:

- [Workspace Explorer](#)
- [Building a PSoC Creator Project](#)

Generating Description Files

PSoC Creator provides a feature to generate two XML-based files that describe the contents of various design entry files (schematic, schematic macro, symbol, UDB, and sheet template). One file (.cysem) contains all the semantically meaningful data from the source. The other file (.cyvis) includes the cosmetic information from the source. There can be several sets of these .cysem and .cyvis files in a project, depending on the types of design entry files you create in a project.

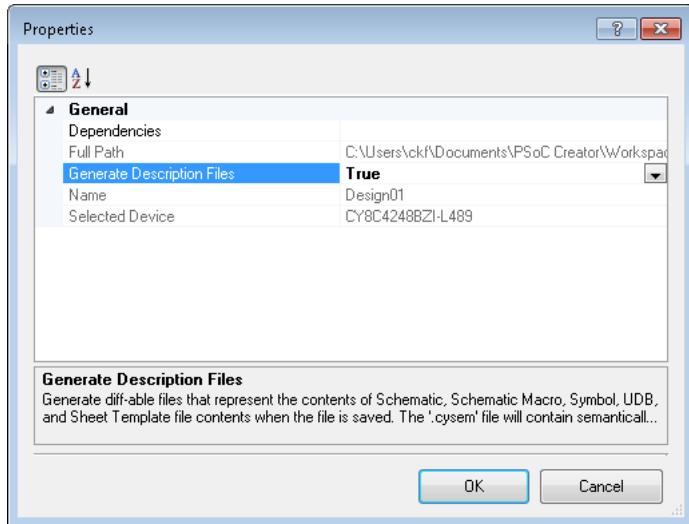
The generated files can be used as documentation for the design. You may also compare these XML-based files from different projects to determine the differences between them, using any text diff utility program. You may include these files in revision control repositories because they enable a reader to determine what changed in a particular PSoC Creator file. However, this is not a revision control feature per se.

Note This feature is turned off by default for every project you create.

To Enable the Feature (Per Project):

In the [Workspace Explorer](#), right-click on a project and select **Properties**. to open the [Properties](#) dialog.

Select **True** for the **Generate Description Files** property, and click **OK** to close the dialog.



To Generate Description Files:

The description files are not automatically generated when the option is enabled. They will not be created for a file until you save a change to that file. After making a save, the files are saved next to the particular design entry file on disk from which it was generated.

Notes:

- If either file is a set as read-only file, a dialog will display requesting to make it writable.
- If you disable the feature, any previously generated files will remain on disk.
- The generated files will be included in all levels of [project archiving](#), as long as the feature is enabled.

Command Line

Command line options have been added to help facilitate this feature, including:

- `-generateDescFiles`: For all specified projects that have 'Generate description files' enabled, generates the description files.
- `-verifyDescFileEnabled`: Verifies that all specified projects have 'Generate description files' enabled.
- `-verifyDescFileContents`: Verifies that all specified projects (that have 'Generate description files' enabled) have generated files that are in-sync with the current version of their source files.

See [CyPrjMgr Command Line Tool](#) for more information.

See Also:

- [Workspace Explorer](#)
- [Properties](#)
- [Source Code Control](#)
- [Archiving a Workspace/Project](#)

Copying a Project

PSoC Creator allows you to copy a project within a workspace, as well as from workspace to another.

To Copy a Project:

Right-click on the project in the Workspace Explorer and select **Copy**.

To Paste a Project:

Right-click on a project or workspace in the Workspace Explorer and select **Paste**.

The copied project is added to the workspace with "_Copy_01" appended to the project name.

See Also:

- [Archiving a Project](#)
- [Saving a Project As](#)

Selecting a Default Compiler

For all target devices, PSoC Creator supports multiple toolchains. While there is a preferred default for each family, you may change the tools used on a per-project basis using the option under [Build Settings](#). However, if you have installed an external compiler, you can specify the default to use for every new project under [Project Management Options](#). To select a default compiler:

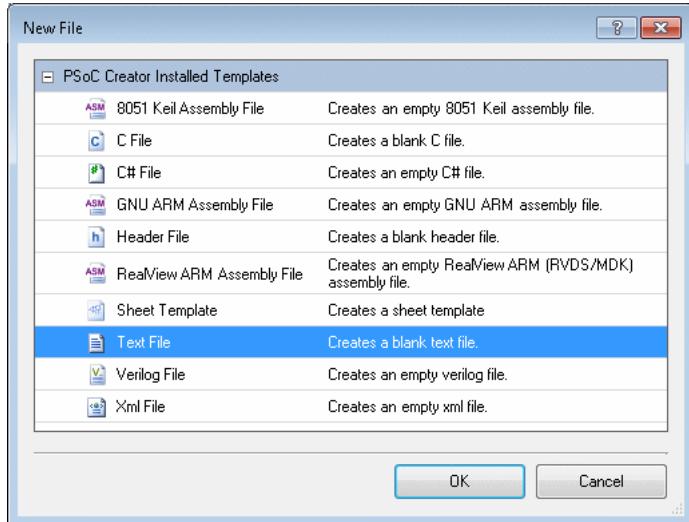
1. Select **Options** from the **Tools** menu.
2. Expand Project Management, and select either DP8051 Toolchains or Arm Toolchains.
3. Select the **Default Toolchain** pull-down menu and select the appropriate option.
For the selected toolchain, a warning indicator will display.
4. Click the **Browse [...]** button, navigate to the location where the binary file is stored, select it, and click **OK**.
5. Click **OK** to close the Options dialog.

See Also:

- [Options Dialog](#)
- [Build Settings](#)

Creating a New File

The New File dialog is used to create new files with PSoC Creator. The dialog provides templates of the different types of files you can create. You may want to create a file for many reasons, such as having a file to copy and paste code.



Creating a new file is **not** the same as adding a file to a project or adding a Component item. This process merely creates an empty file of the type you specify. To add a new file to your project, see [Adding a New Workspace/Project Item](#). To add a Component item to your project see [Adding a Component Item](#).

To Open this Dialog:

From the **File** menu, select **New > File...** 

To Create a New File:

Select the icon for the type of file to create and click **OK**.

An empty file of the type you created opens in the [Text Editor](#) as a tabbed document with a default name.

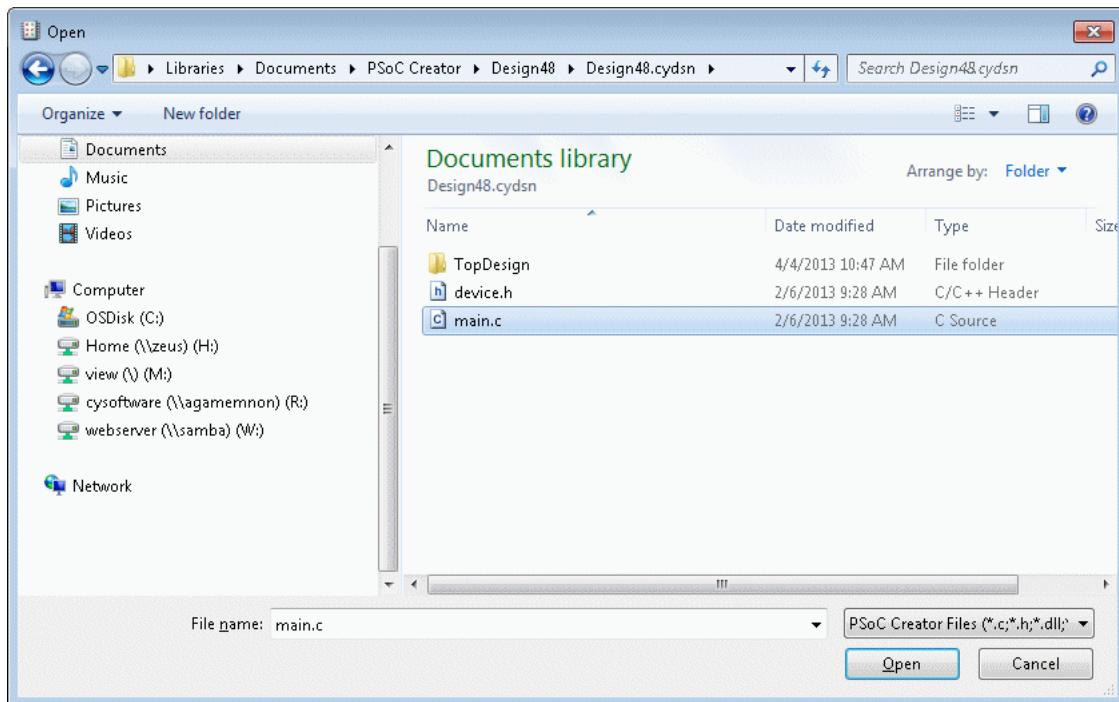
Note The file is not saved to disk until you specify a name and location to save it.

See Also:

- [Text Editor](#)
- [Workspace/Project](#)

Opening an Existing File

The Open File dialog is used to open an existing file within PSoC Creator. You may want to open a file for many reasons, such as having a file to copy and paste code.



Use this dialog to browse and select files you wish to open, such as source code and text files. Opening a file using this dialog merely opens the file in PSoC Creator. That file is not part of any project you may have open. To add a file to an existing project, see [Adding a New Workspace/Project Item](#).

To Open this Dialog:

From the **File** menu, select **Open > File...** .

To Open a File:

1. Navigate to the appropriate directory where the file to open is located.
2. Select the desired file and click **Open**.

The file opens in the [Text Editor](#) as a tabbed document.

See Also:

- [Text Editor](#)
- [Workspace/Project](#)
- [Creating a New File](#)
- [Opening an Existing Project](#)

Creating Folders

In PSoC Creator, there are two different types of folders: physical and virtual. Physical folders are created on disk by PSoC Creator as part of creating projects, Components, and devices/families/architectures. Virtual folders do not exist on disk. You can create virtual folders in your workspaces and projects to organize things as you need them, but you will not see those folders using Windows Explorer.

Filters:

Each folder has a set of associated filters. When a new file is added to a project containing one or more folders, that file will be added to the first folder with a filter matching the extension of the new file. This only occurs when you add the file to the project directly.

To Create Physical Folders:

Use the PSoC Creator interface to create a new project, add Components, or generate files from a build. These folders are created for you by PSoC Creator.

To Create Virtual Folders:

1. In the [Workspace Explorer](#) under the **Source** tab, right-click on a workspace, project, or folder and select **Add > New Folder** or **Add > New Workspace Folder**.

A new folder is added to the selected item and given a default name.

2. Type a name for the folder and press **[Enter]**.

See Also:

- [Workspace Explorer](#)
- [Creating a New Project](#)
- [Creating a Symbol](#)
- [Building a PSoC Creator Project](#)

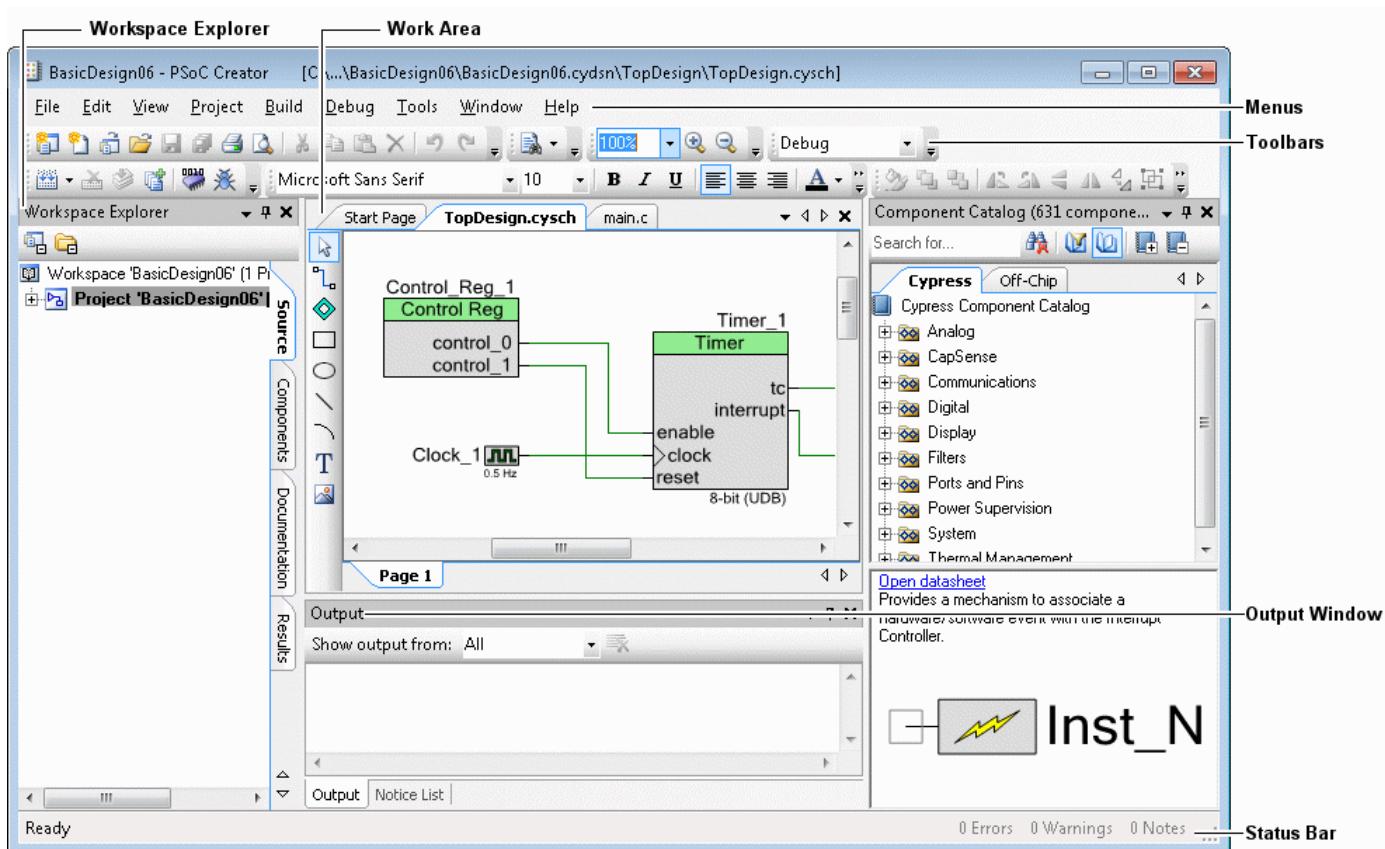
PSoC Creator Framework

This section contains the following topics:

- [Framework Description](#)
- [Window Types](#)
- [Framework Interface Components](#)
- [Customizing the Framework](#)

Framework Description

The PSoC Creator framework provides numerous features to help organize your designs and complete projects faster.



When you first open PSoC Creator, the framework displays with a Workspace Explorer, document work area, and Output window. The framework also contains a menu and a status bar, as well as various toolbars that will change depending on the type of file you are working with.

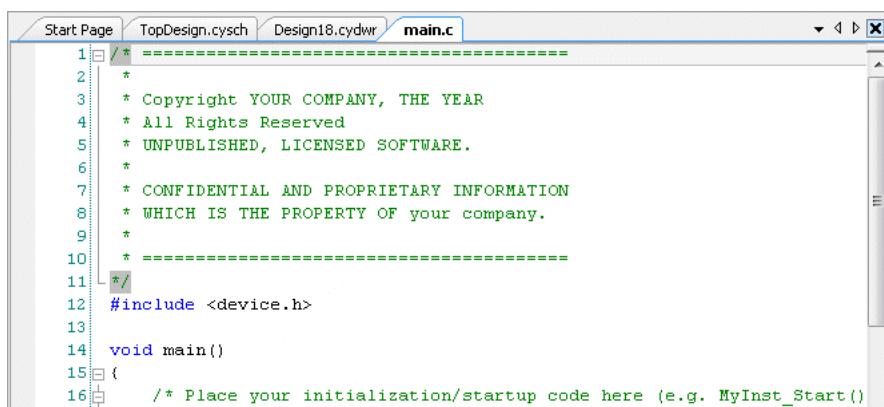
Workspace Explorer:

The Workspace Explorer is a docked tool window, which displays your project files in a similar manner to Windows Explorer. If you double-click a file in the Workspace Explorer, that file will display in the work area. For more information, see [Workspace Explorer](#).

Document Work Area:

The document work area contains various files you have opened for your project. Depending on the type of file displayed, different tools and commands become available. For example, if you open a source code file, you will see toolbar commands for source code editing; if you open a schematic file, you will see commands for editing schematics, such as the Component Catalog used for adding items to your schematic.

All files open in the work area display as tabbed documents, with the active document on top and the other documents behind. For more information, see [Document Windows](#).



```

Start Page TopDesign.cysch Design18.cydwr main.c
1  /* ===== */
2  *
3  * Copyright YOUR COMPANY, THE YEAR
4  * All Rights Reserved
5  * UNPUBLISHED, LICENSED SOFTWARE.
6  *
7  * CONFIDENTIAL AND PROPRIETARY INFORMATION
8  * WHICH IS THE PROPERTY OF your company.
9  *
10 * =====
11 */
12 #include <device.h>
13
14 void main()
15 {
16     /* Place your initialization/startup code here (e.g. MyInst_Start())

```

Start Page

The Start page is a general tabbed document window that is not part of any project. It provides links to create new projects, open existing projects, as well as links to information and news about PSoC devices. It displays in the same work area where other document windows will be opened as you work with your design.

You can configure whether or not to show the Start page on startup; see [Environment Options](#).

Output Window:

The Output window is another docked tool window that shows various system messages. For more information, see [Output Window](#).

Status Bar

The status bar displays informational messages regarding the status of your design. If there are Components that need to be updated, there will be an indication (see [Component Update](#)). The status bar also shows the number of errors, warnings, and notes that are contained in the [Notice List window](#).

See Also:

- [Window Types](#)
- [Document Windows](#)
- [Tool Windows](#)

- [Framework Interface Components](#)

Window Types

PSoC Creator has two types of windows: tool windows and document windows. These two window types behave in slightly different ways.

Tool windows:

Tool windows are listed on the **View** menu and are defined by the current application and its add-ins. They include the [Workspace Explorer](#), [Output Window](#), and [Component Catalog](#). You can configure tool windows to:

- Show or hide automatically
- Tab link with other tool windows
- Dock against the edges of the framework
- Float over

See [Tool Windows](#) for more information.

Document windows:

Document windows are dynamically created when you open or create files or other items. The list of open document windows appears on the **Window** menu, with the top-most window listed first. These can include source code files, schematic files, symbol files, and design-wide resources files. See [Document Windows](#) for more information.

Arranging Windows:

You can increase the viewing and editing space for code, depending on how you arrange the windows. You have several options for arranging windows, including the following:

- Tab-dock document windows
- Dock tool windows to the edge of a frame
- Minimize tool windows along the edge of the frame
- Tile document windows

See [Customizing the Framework](#) for more information.

See Also:

- [Tool Windows](#)
- [Document Windows](#)

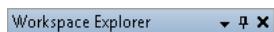
Tool Windows

Tool windows are listed on the [View menu](#) and they include some of the following:

- [Workspace Explorer](#)
- [Output Window](#)
- [Component Catalog](#)

These windows provide different types of functionality within PSoC Creator; however, they all can be arranged within the framework using the same techniques.

Tool Window Toolbar:



Every tool window contains the following toolbar commands:

- **Window Mode** – This pull down menu allows you to toggle the window to different modes, including:
- **Floating** – Sets the tool window as a floating window not attached to the PSoC Creator framework.
- **Dockable** – Sets the tool window as dockable to the PSoC Creator framework.
- **Tabbed Document** – Sets the tool window as a tabbed [document window](#).
- **Auto-Hide** – Sets the tool window to slide to the edge of the framework when not in use.
- **Hide** – Closes the tool window.

These are the same commands available from the tool window right-click context menu.

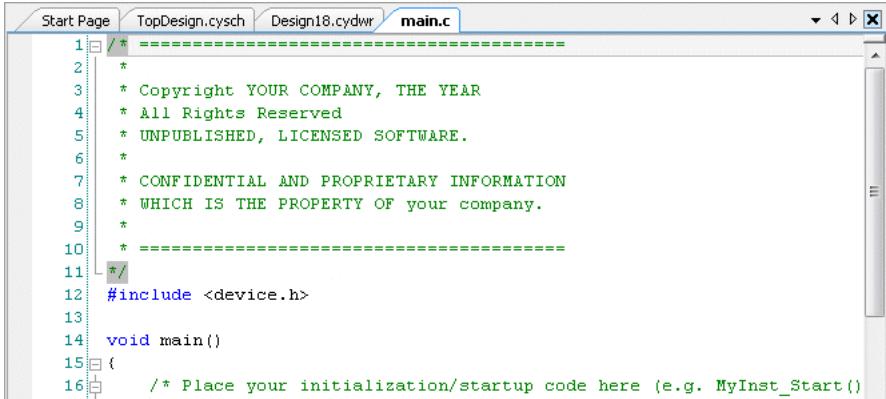
- **Auto-Hide Pushpin** – This command toggles the auto-hide feature on and off.
- **Close** – This command closes the tool window.

See Also:

- [To Auto-Hide Tool Windows](#)
- [To Move Tool Windows](#)
- [To Dock Tool Windows](#)
- [To Float Tool Windows](#)

Document Windows

Document windows are dynamically created when you open or create files or other items. They display in the document work area as tabbed documents, with the active document on top and the other documents behind.



```

1  / ****
2  *
3  * Copyright YOUR COMPANY, THE YEAR
4  * All Rights Reserved
5  * UNPUBLISHED, LICENSED SOFTWARE.
6  *
7  * CONFIDENTIAL AND PROPRIETARY INFORMATION
8  * WHICH IS THE PROPERTY OF your company.
9  *
10 * =====
11 */
12 #include <device.h>
13
14 void main()
15 {
16     /* Place your initialization/startup code here (e.g. MyInst_Start())

```

The following sections describe various aspects of displaying and working with document windows.

Document Work Area Toolbar:



The toolbar at the top right of the document work area contains the following commands:

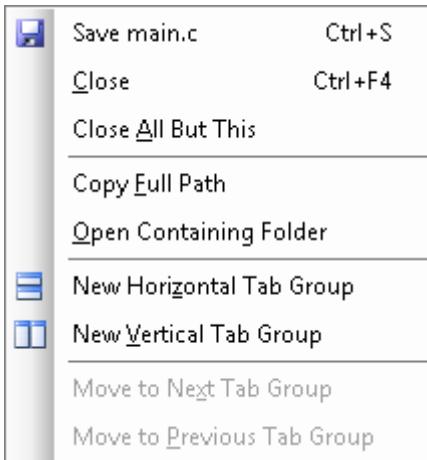
- **Select Document** – This pull down menu displays a list of open document windows in the work area according to the tab order. You can use this menu to select another open document to display.
- **Scroll Left/Right** – The left/right scroll arrows are available if there are more open documents to display to the left or right of the visible work area.
- **Close** – This command closes the active document

Document Tab Context Menus:

You can right-click on a tab to display a context menu. There are two different types of context menus: one for project files and another for non-project files.

Project File Context Menu

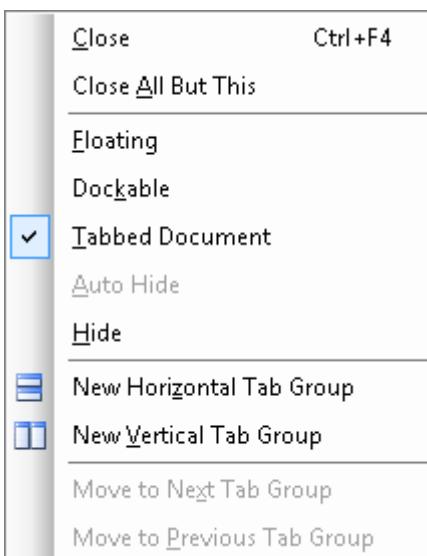
Project files, such as a source code file or a schematic file, can only be displayed as a tabbed document in the document work area. When you right click on a project file tab, the following commands will be available depending on the context of the selected project file:



- **Save** – Saves selected file.
- **Close** – Closes selected file.
- **Close All But This** – Closes all open tabbed documents except the selected file.
- **Copy Full Path** – Copies the full path of the file to the clipboard for pasting into a text file.
- **Open Containing Folder** – Opens the folder in which the file is located.
- **New Horizontal Tab Group** – Creates a new horizontal tab group and moves the selected file to that group.
- **New Vertical Tab Group** – Creates a new vertical tab group and moves the selected file to that group.
- **Move to Next Tab Group** – Moves the selected file to the next tab group.
- **Move to Previous Tab Group** – Moves the selected file to the previous tab group.

Non-Project File Context Menu

Non-project files, such as the Start page, can be displayed as a tabbed document or as a floating or docked [tool window](#). The context menu for these types of files contains the following commands:



- **Close** – Closes selected file.
- **Close All But This** – Closes all open tabbed documents except the selected file.
- **Floating/Dockable/Tabbed Document** – Toggles the display of the file between a floating window, dockable window, or tabbed document.
- **Auto-Hide** – Only active for a dockable window; allows the window to collapse to the edge of the framework.
- **Hide** – Closes selected file/window.
- **New Horizontal Tab Group** – Creates a new horizontal tab group and moves the selected file to that group.
- **New Vertical Tab Group** – Creates a new vertical tab group and moves the selected file to that group.
- **Move to Next Tab Group** – Moves the selected file to the next tab group.

- **Move to Previous Tab Group** – Moves the selected file to the previous tab group.

To Select Document Windows:

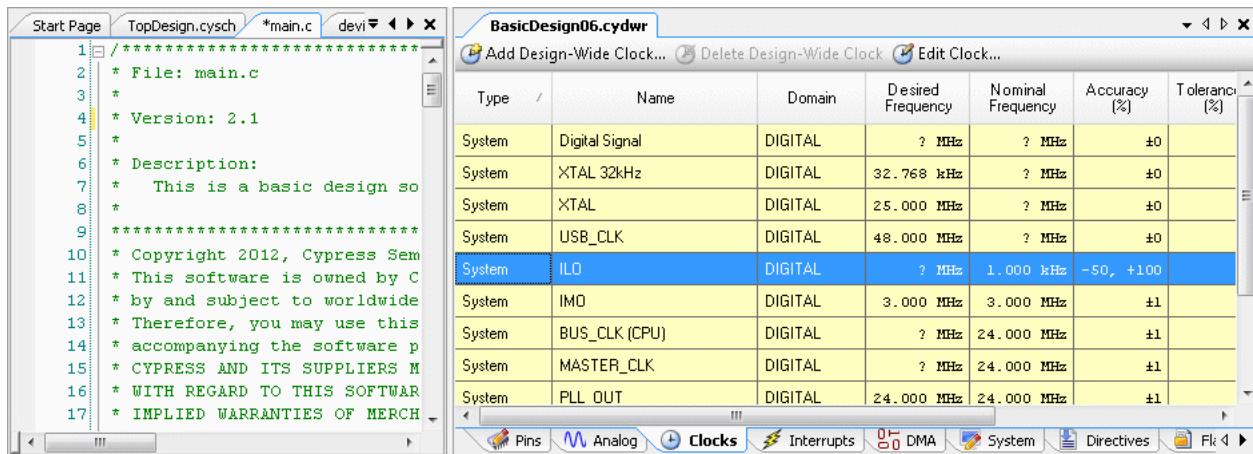
When you have Tabbed Documents, document windows are placed side-by-side on tabbed panes as they are opened. This makes it simple to cycle through the documents you are editing.

To move through open documents in order of use:

- Press **[Ctrl] + [Tab]** to activate open documents in the order that they were most recently touched.
- Press **[Ctrl] + [Shift] + [Tab]** to activate open documents in the reverse order.

Tab Groups:

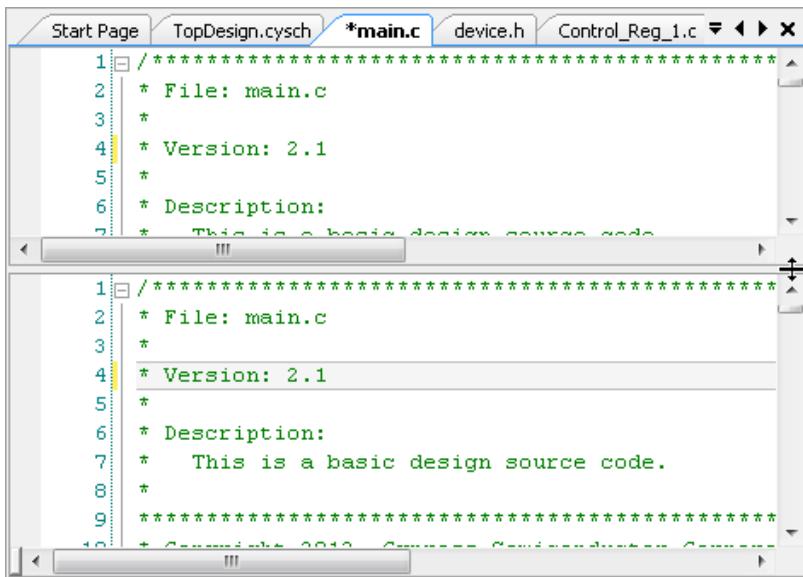
Tab Groups allow you to manage limited workspace while working with two or more open documents. You can organize multiple document windows into either vertical or horizontal Tab Groups and easily shuffle documents from one Tab Group to another.



Split Windows:

Some types of files, such as source code, allow you to view or edit two locations of the same file at once.

- To divide your document into two independently scrolling sections, select the **Split** icon located above the scroll bar and drag to the desired location.



- To remove the split, drag from the split back above the scroll bar.

See Also:

- [Tool Windows](#)

Framework Interface Components

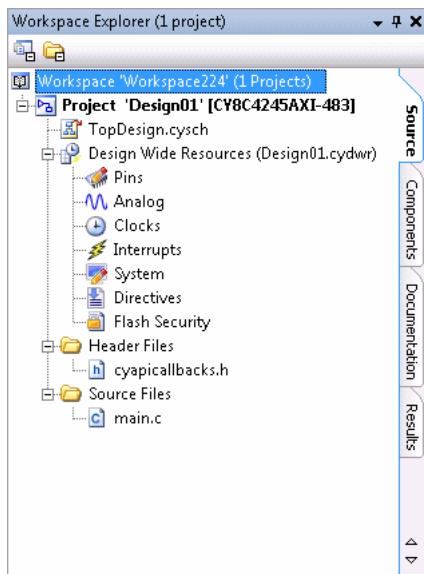
The major PSoC Creator framework Components include:

- [Workspace Explorer](#)
- [Output Window](#)
- [Notice List Window](#)
- [Resource Meter](#)
- [File Menu](#)
- [Edit Menu](#)
- [View Menu](#)
- [Project Menu](#)
- [Build Menu](#)
- [Debugger Menus](#)
- [Tools Menu](#)
- [Window Menu](#)
- [Help Menu](#)
- [Standard Toolbar](#)

- [Keyboard Shortcuts](#)

Workspace Explorer

The Workspace Explorer is PSoC Creator's method for displaying the contents of a workspace.



The contents are displayed in a tree format, similar to Windows Explorer. There are also different tabs along the side of the tool window that allow you to view different types of files by category.

Toolbar Commands:

The Workspace Explorer has two commands:

- **Collapse to Project** – Collapses the selected project tree to the top-level project.
- **Collapse to Parent** – Collapse the selected tree to the top-level node.

Note When you right-click on most items in the Workspace Explorer tree, you can access context-menu commands relevant to the item selected. Most of these commands are described under [Project Menu](#) or [Build Menu](#).

To Open a File:

Double-click any file in the Workspace Explorer to open it in the document work area as a tabbed document.

Source Tab:

The **Source** tab displays by default if no other tab was selected. PSoC Creator only shows files you added or that were generated from the Code Generation step of a PSoC Creator build. The view is updated whenever a build completes or PSoC Creator is given focus.

When in the **Source** tab, you will only be able to add new folders and files to projects, or create new projects. You will not be able to manipulate Components. If you wish to add new items to Components, or new Components entirely, you must use the **Components** tab.

Generated Source

In a design project, the Generated Source directory contains the source from the Code Generation step. Under the Generated Source directory is a sub-directory for each architecture that has been built in the past. Each architecture directory is further subdivided with virtual folders for each Component instance that provided an API. In addition to the instance directories, the architecture directory contains other generated files, such as a *project.h*, boot file, etc., with each project being a node in the tree. Each project in the workspace also displays all of its contents. See [Generated Files](#).

Components Tab:

The **Components** tab filters the contents of the workspace, showing only the Components belonging to each project in the workspace, as well as the file and folder contents of all Components. You may add new Components and items to Components, and you may add new projects while in this tab. You may add files and folders outside of the Components, but only using the **Source** tab for those operations. For more information about Components, refer to the [Component Author Guide](#).

Documentation Tab:

The **Documentation** tab displays and provides quick access to documentation, including Technical Reference Manuals (TRMs), as well as device and Component datasheets currently used in your design. As you add and remove Components, or change devices, the list of available documents will update accordingly.

If you double-click a file, it will open in your default PDF reader tool.

Results Tab:

The **Results** tab displays a dynamic listing of interesting files from the most recent build of each platform. A platform will select a subset of build files to display, and there may be differences between what is displayed from one platform to another.

If a platform or configuration has not been built, or if the folders created by that build have been deleted, the folder for that platform will not be displayed. In addition, if platform and configuration directories are found, but an expected file is missing, that file will appear grayed out, indicating it was not found on disk when expected.

The minimum set of files to be displayed for a Design build is as follows:

- Programming file
- Debugging file (if different from Programming file)
- Code Generation Report file (if Code Generation was run)
- Device File (when implemented)

Additional files that may be included are list files and map files.

For a Library build, the resulting library file will be displayed.

Files in the **Results** tab can be double-clicked to open an appropriate editor for the file, such as the Text Editor for a report file. Other actions that may be available include programming a device or debugging, as appropriate. The projects displayed in the **Results** tab cannot be modified in any way. The tab is simply a display of the results of builds.

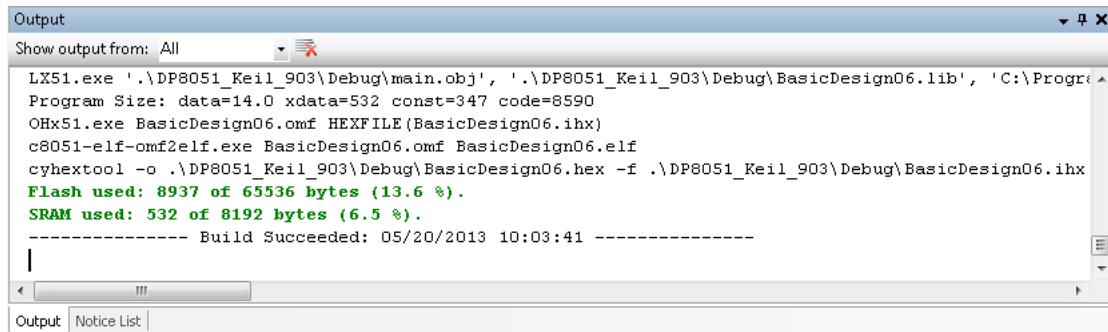
See Also:

- [Workspace/Project](#)
- [Framework Description](#)

- Building a PSoC Creator Project
 - Generated Files

Output Window

The Output window displays status messages for various features in PSoC Creator, including the builder and debugger. This window is usually located at the bottom of the [PSoC Creator framework](#). It is often in the same window group as the [Notice List Window](#).



Note When building a bootloader project, the flash size represents the number of bytes to store in flash (the same way it works for normal projects). When building a bootloadable project, the bootloader size represents the size of the flash rows consumed by the bootloader (the bootloader is padded to a multiple of the row size).

To Display the Output Window:

The Output window usually displays by default when you launch PSoC Creator. If the window was closed, you can open it again by selecting **Output Window** from the **View** menu.

Toolbar:

Depending on which tool you are using, the Output window will have various toolbar commands for you to use. The following are the most common commands:

Show output from

Displays one or more output panes to view. Several panes of information might be available, depending upon which tools have used the Output window to deliver messages.

[Clear all](#)

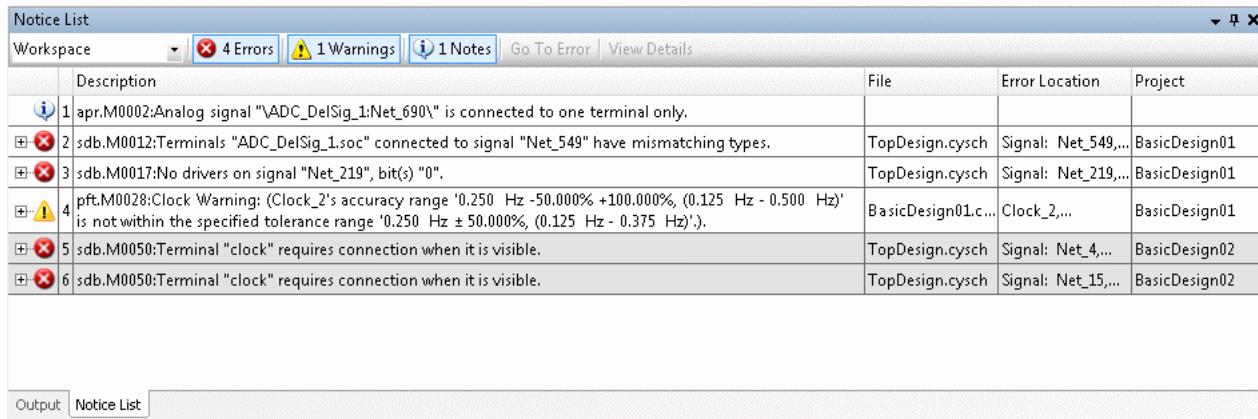
Clears all text from the Output pane.

See Also:

- Framework Description
 - Notice List Window

Notice List Window

The Notice List window combines notices (errors, warnings, and notes) from many places in your design into one centralized list. If a file and/or error location is shown, you can double-click the entry to show the error or warning. There are also buttons to **Go To Error** or **View Details**. This window is usually located at the bottom of the [PSoC Creator framework](#). It is often in the same window group as the [Output window](#).



If your design's Notice List window contains notices, they are displayed in numbered rows. Above the rows are a set of filters to show or hide notices, as follows:

- **Workspace/Active Project** – Use this combo box to show notices from all the projects (and their dependencies) in the workspace or to show notices only from the active project (and its dependencies). This filter only applies to notices associated with a project.
- **Errors**  – These indicate there is at least one problem that must be addressed before you can build your application. Typical errors could include: compiler build errors, dynamic connectivity errors in schematics, and Design Rule Checker (DRC) errors. Errors from the build process remain in the list until the next build.
- **Warnings**  – These report unusual conditions that might indicate a problem, although you can usually build the application regardless.
- **Notes**  – These are informational messages from the system to indicate something occurred.

Note Gray rows are notices not associated with the active project or one of its dependencies. They will not be shown if the **Active Project** filter is selected from the combo box.

The Notice List contains the following columns:

- **Icon** – Displays the icons for the error, warning, or note. A specific row may also contain a tree control containing individual parts of the overall message.
- **Number** – Displays the number of the notice.
- **Description** – Displays a brief description of the notice.
- **File** – Displays the file name where the notice originated,
- **Error Location** – Displays the specific line number or other location of the message, when applicable.

- **Project** – Displays the name of the project that contains the notice. White rows are for the Active project; grey rows are for inactive projects.

Design Rule Checker (DRC):

The DRC evaluates your design based on a collection of pre-determined rules in the project database. The DRC points out potential errors or "rule" violations in your project that might pose problems. It displays various messages in the Notice List window.

Some connectivity and dynamic errors update as soon as you make changes to your design, while other errors update on load and save operations.

To Open the Notice List Window:

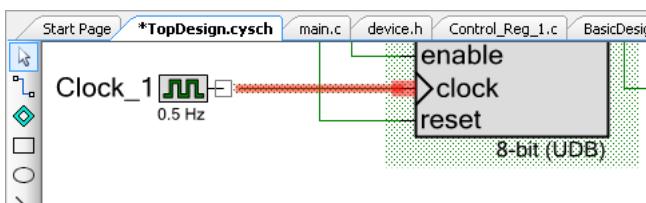
The Notice List window is displayed by default. If you close the window, you can open it again by selecting **View > Other Windows > Notice List**.

You can also re-open the window by selecting **Windows > Reset Layout**. However, this option will reset all windows to the default positions.

To Display Errors in Tools:

Click on a notice in the list. If there is a link to the error, click the **Go To Error** button. You may also double-click any warning or error in the Notice List window to show that error or warning in the appropriate tool. There is also a command available to go to the error if you right-click on a notice.

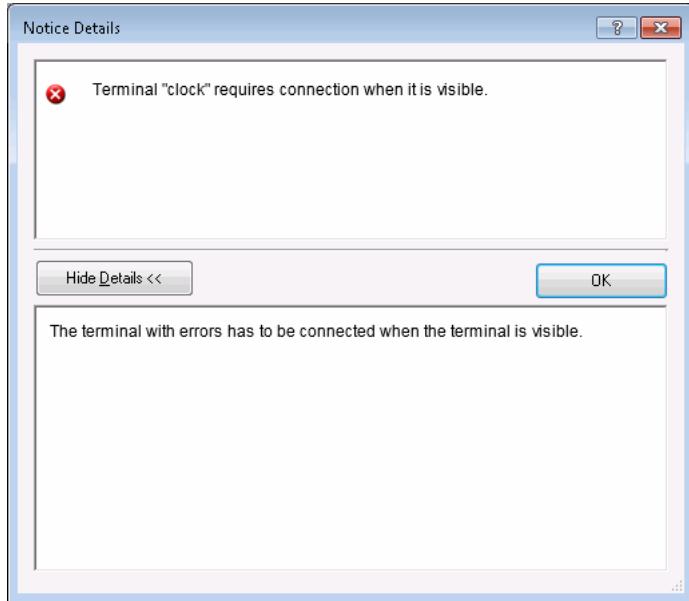
- In a schematic, a translucent red highlight shows the source(s) of the error. You can turn highlighting on or off using the right-click menu on a notice.



- In a code file, the specific line of code with the error or warning will be highlighted.

To View the Message Description:

Click the  button next to the message or click the **View Details** button to open the [Notice Details dialog](#). There is also a command available to view details if you right-click on a notice. The Notice Details dialog will display the entire message, as well as additional information if available.



Context Menu:

As described previously, there is also a context menu if you right-click on a notice. The menu items available include:

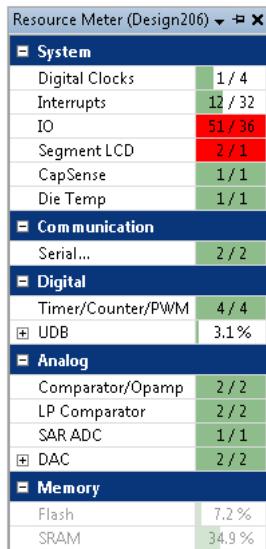
- **Show/Hide Error Highlighting:** Turns on/off highlight of the error.
- **Go to the Error Location:** Moves the display to the error, if applicable.
- **More Information:** Displays the Notice Details dialog for more information.

See Also:

- [Framework Description](#)
- [Output Window](#)
- [Notice Details](#)

Resource Meter

The Resource Meter is a graphical display of the resources used in the currently [active project](#) of your workspace. It updates after every build. This keeps you notified of possible resource overuse and helps identify which resource is the cause of the problem.



Description

Each resource contains a color-coded bar graph that shows the number or percentage of resources used. The following table shows the colors used and indicate their meaning:

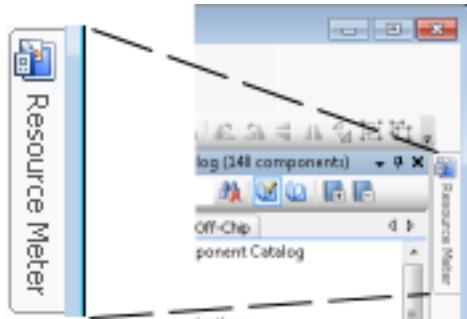
Color	Description
Green	Resource fits in the current design.
Red	Resource is overused in the current design; build failed.
White	Resource is not used.

If the build fails or only a partial build is performed (that is, Generate Application), any resources that did not have their usage recalculated will be displayed as washed out, as it is potentially stale.

Note Any previous usage calculations will still be displayed; they just might not be accurate due to the fact that they weren't recalculated.

To Open the Resource Meter:

The Resource Meter is available after you [open a project](#) or [create a new project](#). By default, this [tool window](#) is set to [auto-hide](#) on the right side of the PSoC Creator [framework](#). To display it, click the **Resource Meter** tab:



If the window is closed, you can open it again by selecting **Resource Meter** from the **View** menu.

File Menu

The File menu contains the following commands:

Menu Item		Icon	Shortcut	Description	See Also
New >				Displays a dialog to create a new project.	Creating a New Project
				Displays a dialog to create a new file.	Creating Files
Open >				Displays a dialog to open an existing project/workspace.	Opening an Existing Project
			[Ctrl] + [O]	Displays a dialog to open an existing file.	Opening an Existing File
Code Example				Displays a dialog to open a code example.	Find Code Example
Add >	New Project			Adds a new project to the workspace.	
	Existing Project			Adds an existing project to the workspace.	
Close			[Ctrl] + [F4]	Closes the active document window.	Window Types
Close Workspace				Closes the workspace.	
Save			[Ctrl] + [S]	Saves the active document.	
Save <Project / File> As				Displays a dialog to save the selected project or file as another project or file.	Saving a Project As
Save All			[Ctrl]+ [shift] + [S]	Saves all the files in the workspace.	
Create Workspace Bundle				Opens the Workspace/Project Archiver wizard to create workspace bundle.	Archiving a Workspace/Project
Page Setup				Opens the Page Setup dialog to select printing layout options.	
Print			[Ctrl] + [P]	Prints the active document.	
Print Preview				Opens the Print Preview dialog for the selected file.	Print Preview
Recent Files				Provides access to previously opened files.	
Recent Projects				Provides access to previously opened projects.	
Exit				Exits PSoC Creator.	

Edit Menu

The Edit menu contains the following commands:

Menu Item	Icon	Shortcut	Description	See Also
Undo		[Ctrl] + [Z]	Undoes the last action; can be used repeatedly to undo multiple actions.	
Redo		[Ctrl] + [Y]	Redoes the last action.	

Menu Item	Icon	Shortcut	Description	See Also
Cut		[Ctrl] + [X]	Cuts the selected element(s).	
Copy		[Ctrl] + [C]	Copies the selected element(s).	
Paste		[Ctrl] + [V]	Pastes the cut or copied elements from the clipboard.	
Delete		[Delete]	Deletes the selected element(s).	
Select All		[Ctrl] + [A]	Selects all elements in active window.	
Find and Replace >				
Find		[Ctrl] + [F]	Opens a dialog to find text.	Find Replace
Replace		[Ctrl] + [H]	Opens a dialog to find and replace text.	Find Replace
Find in Files		[Ctrl] + [Shift] + [F]	Opens a dialog to find text in multiple files.	Find in Files
Replace in Files		[Ctrl] + [Shift] + [H]	Opens a dialog to find and replace text in multiple files.	Replace in Files
Go To		[Ctrl] + [G]	Opens the Go To dialog.	Go To Line
Advanced >				
Tabify Selected Text			Converts spaces to tabs for the selected text. The number of spaces per tab is defined in the Text Editor Options Dialog .	Text Editor
UnTabify Selected Text			Converts tabs to spaces for the selected text.	
Make Uppercase		[Ctrl] + [Shift] + [U]	Changes selected text to uppercase letters.	
Make Lowercase		[Ctrl] + [U]	Changes selected text to lowercase letters.	
View White Space		[Ctrl] + [E], [S]	Toggles on or off to show hidden characters, such as spaces or tabs. Press and hold [Ctrl] and [E] keys, release, and then press [S] key.	
Incremental Search		[Ctrl] + [I]	Find first instance of matching pattern after pressing shortcut keys. Press [Ctrl] + [I] keys and the icon displays. Then type letters to find.	
Comment Selection		[Ctrl] + [E], [C]	Comments selected line(s) of text. Inserts // at the beginning of the line. Press and hold [Ctrl] and [E] keys, release, and then press [C] key.	
Uncomment Selection		[Ctrl] + [E], [U]	Uncomments selected line(s) of text. Removes // at the beginning of the line. Press and hold [Ctrl] and [E] keys, release, and then press [U] key.	
Increase Line Indent			Indents the selected line(s) of text.	
Decrease Line Indent			Outdents the selected line(s) of text.	
Toggle Bookmark		[Ctrl] + [K], [Ctrl] + [K]	Inserts or removes a bookmark from the Indicator Margin next to the line where the cursor is located. Press and hold [Ctrl] and [K] keys, release, and then press [Ctrl] and [K] keys again.	
Outlining >				
Toggle Outlining Expansion		[Ctrl] + [M], [M]	If Start Automatic Outlining has been selected, this command expands or contracts the outlining for selected section of code. Press and hold [Ctrl] and [M] keys, release, and then press [M] key again.	Text Editor

Menu Item	Icon	Shortcut	Description	See Also
Toggle All Outlining		[Ctrl] + [M], [L]	If Start Automatic Outlining has been selected, this command expands or contracts all outlining in the file. Press and hold [Ctrl] and [M] keys, release, and then press [L] key.	
Stop Automatic Outlining		[Ctrl] + [M], [P]	Disables automatic outlining. Press and hold [Ctrl] and [M] keys, release, and then press [P] key.	
Start Automatic Outlining		[Ctrl] + [M], [P]	Enables automatic outlining.	
Snippets >				
Insert Snippet...		[Ctrl] + [K], [X]	Opens a menu to select code snippets, if available.	
Surround with...		[Ctrl] + [K], [S]	Opens a menu to insert surround with snippets, if available.	

View Menu

The View menu contains the following commands:

Menu Item	Icon	Shortcut	Description	See Also
Output Window			Open or display the Output window.	Output Window
Workspace Explorer			Open or display the Workspace Explorer.	Workspace Explorer
Code Explorer			Open the Code Explorer window for the Code Editor.	Code Explorer Window
Component Catalog			Open the Component Catalog for the Schematic Editor.	Component Catalog
Resource Meter			Open the Resource Meter for the currently active project.	Resource Meter
Other Windows >				
Notice List			Open or display the Notice List window.	Notice List Window
Start Page			Open or display the Start Page.	Framework Description
Find Results			Open or display Find Results windows.	Find and Replace
Zoom In/Out/To		[Ctrl] + [+] or [Ctrl] + [-]	Adjusts size of the page according to option	

Project Menu

The Project menu contains the following commands:

Menu Item	Icon	Description	See Also
New Item		Open New Item dialog to select a new item to add to your project/workspace.	New Item
Add Component Item		Open Add Component Item dialog to add a new item to your Component.	Add Component Item
Existing Item		Open the Open dialog to add an existing item to your project/workspace.	Opening an Existing File
Import Component		Opens the Import Component dialog.	Import Component
Update Components		Opens the Update Component Tool dialog to update selected Components.	
Remove from <Project/Workspace>		Remove selected file/project from project/workspace.	
Unload/Reload Project		Alternately unloads and reloads the selected project in the Workspace.	
New Folder		Create a new virtual folder in the Workspace Explorer but not on disk.	
Show All Files		Open all files in a directory.	
Set As Active Project		Set the selected project as the active project.	
Set As Top Component		Set the selected Component as the top Component.	
Dependencies		Open the Dependencies dialog.	Dependencies
Build Order		Opens the Dependencies dialog to the Build Order tab.	Dependencies
Device Selector		Open the Device Selector.	Device Selector
Archive Workspace/Project		Opens the Archive Project dialog to archive the workspace/project.	Archiving a Workspace/Project
Export to IDE		Opens a dialog to export the PSoC Creator design to an external IDE. Note This option is disabled for PSoC 6 projects. Refer to Build Settings > Target IDEs .	Exporting a Design to 3rd Party IDE
<Project> Resources		Opens the Design-Wide Resources file for the project.	Design-Wide Resources
Build Settings		Open the Build Settings dialog.	Build Settings
Properties		Opens the Properties dialog.	Properties

Build Menu

The Build menu contains the following commands:

Menu Item	Icon	Shortcut	Description	See Also
Build All Projects		[F6]	Build all projects in the workspace.	Building a PSoC Creator Project
Clean All Projects			Clean all projects in the workspace. Note During the clean process, PSoC Creator cleans the build output of only those files that are still part of the project. Any output files generated using source files that are no longer part of the project (deleted or simply removed from project), will be left untouched.	
Clean and Build All Projects			Clean and build all projects in the workspace.	
Build (Named) Project		[Shift] + [F6]	Build the selected project.	
Clean (Named) Project			Clean the selected project.	
Clean and Build (Named) Project			Rebuild the selected project.	
Cancel Build		[Ctrl] + [Break]	Cancel the build process.	
Compile File		[Ctrl] + [F6]	Compile the selected file.	
Generate Application			Generate API source code files.	Generated Files
Generate Project Datasheet			Generate a project datasheet.	Generating a Project Datasheet

See Also:

- [Build Toolbar Commands](#)

Debugger Menu Commands

The Debug menus provide access to all of the functionality and information available to the current debugger. There are two modes for the menu: inactive and active.

Inactive Mode Debug Menu:

When the debugger is not running, the Debug menu will be in the inactive state. In this state, only the functions that make sense are available on the menu. For instance, the Halt function is unnecessary in the inactive state. In the inactive mode, the Debug Windows submenu provides access to a limited subset of the available debug windows.

Command	Icon	Shortcut	Description
Windows >			Provides access to the various debugger windows. See Debugger Windows .
Breakpoints		[Ctrl] + [D], [B]	Opens the Breakpoints window to display all of the breakpoints that have been set in the workspace.

Command	Icon	Shortcut	Description
Output		[Ctrl] + [D], [O]	Opens the Output window .
Program		[Ctrl] + [F5]	Provides a one click means of programming the selected debug target with the code generated from the selected project. <ul style="list-style-type: none"> • Automatically starts a build if the project is out-of-date • Updates the status bar's message to indicate that programming is taking place. • Launches PSoC Programmer behind the scenes to perform the programming.
Select Debug Target			Opens the Select Debug Target dialog to manually select the debug target to use.
Debug		[F5]	Starts the debugger.
Debug without Programming		[Alt] + [F5]	Starts the debugger without programming the device.
Attach to Running Target			Opens the Attach to Target dialog to connect to an already programmed target device. Applicable to PSoC 3 and PSoC 5LP only.
Toggle Breakpoint		[F9]	Alternately inserts and removes a breakpoint in the current line of code. This is the same as clicking in the Indicator Margin of the Code Editor . You can use the [F9] key as a shortcut for this command.
New Breakpoint >			Provides access to the following breakpoint windows. See Breakpoints Window .
Address Breakpoint		[Ctrl] + [D], [A]	Opens the Address Breakpoint window.
File Line Breakpoint		[Ctrl] + [D], [F]	Opens the File/Line Breakpoint window.
Function Breakpoint		[Ctrl] + [D], [U]	Opens the Function Breakpoint window.
Variable Watchpoint		[Ctrl] + [D], [V]	Opens the Variable Watchpoints window.
Memory Watchpoint		[Ctrl] + [D], [E]	Opens the Memory Watchpoint window.
Delete All Breakpoints		[Ctrl] + [Shift] + [F9]	Deletes all breakpoints in the workspace instead of having to remove each one individually. This is useful if there are multiple breakpoints set but you just want the processor to run.
Enable All Breakpoints			Enables all breakpoints

Active Debug Windows Menu:

The active debug mode indicates that you have started a debug session; that the active PSoC Creator subsystem is the debugger. In the active mode, the debugger is running and you have the full range of available functions; the Debug Windows submenu provides access to all of the available debug windows.

Command	Icon	Shortcut	Description
Windows >			Provides access to the various debugger windows. See Debugger Windows .
Breakpoints		[Ctrl] + [D], [B]	Opens the Breakpoints window to display all of the breakpoints that have been set in the workspace.
Output		[Ctrl] + [D], [O]	Opens the Output window .

Command	Icon	Shortcut	Description
Watch >			
1		[Ctrl] + [D], [W]	Opens one of four Watch windows to evaluate and display variables, registers, or expressions.
2		[Ctrl] + [Alt] + [W], [2]	
3		[Ctrl] + [Alt] + [W], [3]	
4		[Ctrl] + [Alt] + [W], [4]	
Locals		[Ctrl] + [D], [L]	Opens the Locals window to view and modify all of the local variables in the current debug frame.
Components		[Ctrl] + [D], [P]	Opens the Component Debug Window to view debug information about Components in your design.
Call Stack		[Ctrl] + [D], [C]	Opens the Call Stack window to track the order that different functions are called by the target program.
Memory >			
1		[Ctrl] + [D], [M]	Opens one of four Memory windows to display the values stored in the memory of the processor.
2		[Ctrl] + [Alt] + [M], [2]	
3		[Ctrl] + [Alt] + [M], [3]	
4		[Ctrl] + [Alt] + [M], [4]	
Disassembly		[Ctrl] + [Alt] + [D]	Opens the Disassembly window to display the basic instructions created for your source code.
Registers		[Ctrl] + [R], [R]	Opens the Registers window to display the core CPU registers and their values
Program		[Ctrl] + [F5]	<p>Provides a one click means of programming the selected debug target with the code generated from the selected project.</p> <ul style="list-style-type: none"> • Automatically starts a build if the project is out-of-date • Updates the status bar's message to indicate that programming is taking place. • Launches PSoC Programmer behind the scenes to perform the programming.
Select Debug Target			Opens the Select Debug Target dialog to manually select the debug target to use.
Show Current Line			Displays the line of code that is or will be executed.
Resume Execution		[F5]	Continues the debugger. Starts the debug target running again after a Halt or a breakpoint. Use this function to have the program continue running to the next breakpoint.
Halt Execution		[Ctrl] + [Alt] + [Break]	Pauses the debugger.
Stop Debugging		[Shift] + [F5]	Stops the debugging session.
Rebuild and Run		[Ctrl] + [Shift] + [F5]	Rebuilds the program and restarts the debugger.
Reset		[Ctrl] + [Alt] + [F5]	Resets the debugger.
Step Into		[F11]	Executes a single line of code. If the line is a function call, the debugger will break at the first instruction in the function. If the line is not a function call, the debugger will break at the following line of code. Use this to verify that a line of code is doing what is expected.

Command	Icon	Shortcut	Description
			This function temporarily allows the processor to run until it finishes processing the instructions that make up the current line of code.
Step Over		[F10]	Executes a single line of code. The debugger will break at the following line of code. If the current line of code is a function call, the function will be executed without stopping. The debugger will then stop on the next line after the function call. Use this to verify that a line of code is doing what is expected. This function temporarily allows the processor to run until it finishes processing the instructions that make up the current line of code.
Step Out		[Shift] + [F11]	Finishes executing the current function. The processor is allowed to run until the current function has finished. It will halt again at the first instruction after the function call. Use this to exit the current function and return to the calling method. This function temporarily allows the processor to run until it finishes processing the instructions that make up the current function.
Toggle Breakpoint		[F9]	Alternately inserts and removes a breakpoint in the current line of code. This is the same as clicking in the Indicator Margin of the Code Editor . You can use the [F9] key as a shortcut for this command.
New Breakpoint >			Provides access to the following breakpoint windows. See Breakpoints Window .
Address Breakpoint		[Ctrl] + [D], [A]	Opens the Address Breakpoint window.
File Line Breakpoint		[Ctrl] + [D], [F]	Opens the File/Line Breakpoint window.
Function Breakpoint		[Ctrl] + [D], [U]	Opens the Function Breakpoint window.
Variable Watchpoint		[Ctrl] + [D], [V]	Opens the Variable Watchpoints window.
Memory Watchpoint		[Ctrl] + [D], [E]	Opens the Memory Watchpoint window.
Delete All Breakpoints		[Ctrl] + [Shift] + [F9]	Deletes all breakpoints in the workspace instead of having to remove each one individually. This is useful if there are multiple breakpoints set but you just want the processor to run.
Enable All Breakpoints			Enables all breakpoints.
Refresh			Refreshes the debugger.
Enable/Disable Global Interrupt			Enable/Disable (depending on current state) Global Interrupt

See Also:

- [Using the Debugger](#)
- [Debugger Toolbar Commands](#)
- [Text Editor Context Menu Commands](#)

Tools Menu

The Tools menu contains the following commands:

Menu Item	Description	See Also
Find new Components	Opens the Component Installer dialog to find and install new and updated Components from the Cypress web site.	Component Installer
Find new devices	Opens the Device Update Installer dialog to find and install new devices from the Cypress web site.	Device Update Installer
Install drivers for µVision	Opens the Installation Wizard to install MiniProg3 drivers for the µVision IDE.	Installing MiniProg3 Drivers
Datapath Config Tool	Launches the Datapath Configuration Tool for use when creating Components implemented with Verilog.	Component Author Guide
DMA Wizard	Opens the DMA Wizard.	DMA Wizard
Component Tuners	Allows direct access to tuner dialogs if included in your design.	Tuner Communication Setup
Bootloader Host...	Launches the stand-alone Bootloader Host application.	Bootloader Host Help file (Press [F1] within the application.)
Options...	Opens the Options dialog.	Options Dialog

Window Menu

The Window menu contains the following commands:

Menu Item	Description
New Horizontal Tab Group	Creates a new horizontal Tab Group .
New Vertical Tab Group	Creates a new vertical Tab Group .
Move to Previous Tab Group	Move to Previous Tab Group .
Move to Next Tab Group	Move to Next Tab Group .
Auto Hide All	Sets all Tool Windows to the Auto Hide setting.
Close All Documents	Closes all open Document Windows .

Help Menu

The Help menu contains the following commands:

Menu Item	Description
PSoC Creator Help Topics	Opens this Help to the Welcome topic.
Document Manager	Opens the Document Manager to access various Cypress documents. If Cypress Document Manager is not installed, this opens a link on the Cypress web site to learn more about it and download it.
System Reference Guides	Opens an HTML page containing information and links to various system-wide and design-wide Components.
Update Manager	Launches the Cypress Update Manager to check for and install program updates.
PSoC Creator Training	Opens the Cypress PSoC Creator Training web page.
Cypress Dev Community	Open the Cypress Development Community web page.

Menu Item	Description	
Documentation >	<ul style="list-style-type: none"> Quick Start Guide - Opens the PSoC Creator Quick Start document. PSoC Creator User Guide Opens the Keyboard Shortcuts Help topic. Release Notes - Opens the Release Notes document. Known Problems & Solutions - Opens a Cypress web page to access the Known Problems and Solutions document. Device Datasheets - Opens a Cypress web page to access PSoC device datasheets. PSoC Technical Reference Manuals - Opens a Cypress web page to access PSoC TRM documents. 	
	<ul style="list-style-type: none"> Japanese Documentation > Chinese Documentation > 	<ul style="list-style-type: none"> Component Datasheets - Opens a web page to translated copies of various Component datasheets. Component Datasheets - Opens a web page to translated copies of various Component datasheets. PSoC Creator User Guide (Chinese)
	<ul style="list-style-type: none"> Component Author Guide Peripheral Driver Library - Opens the documentation for the appropriate PDL version for your project, based on the path set in the Options Dialog. Datapath Configure Tool User Guide UDB Editor Guide Customizer API Reference Guide Warp Verilog Reference Guide Tuner API Reference Guide 	
	<ul style="list-style-type: none"> GCC Documentation Keil 	
Register >	<ul style="list-style-type: none"> PSoC Creator Keil 	
Support >	<ul style="list-style-type: none"> Knowledge Base - Opens the Cypress Knowledge Base web page. Create a Support Case - Opens the Cypress support web page to create a new support case. View Your Support Cases - Opens the Cypress support web page to your home page. 	
Order Samples	Opens the Cypress Store web page to order samples, kits, cables, etc.	
Contact Us	Opens Help topic to provide ways to contact Cypress .	
About	Opens the About PSoC Creator dialog, which provides build and plug-in information.	

Standard Toolbar



The PSoC Creator standard toolbar contains the following commands:

- [New Project/Workspace](#)
- [New File](#)
- [Open Project/Workspace](#)
- [Open File](#)

- Save
- Save All
- Print
- Print Preview
- Cut
- Copy
- Paste
- Delete
- Undo
- Redo

Keyboard Shortcuts

The following table lists many of the various keyboard shortcuts available in PSoC Creator.

Location	Command	Shortcut
File Menu	Open	[Ctrl] + [O]
	Close active file	[Ctrl] + [F4]
	Save	[Ctrl] + [S]
	Save All	[Ctrl] + [Shift] + [s]
	Print	[Ctrl] + [P]
Edit Menu	Undo	[Ctrl] + [Z]
	Redo	[Ctrl] + [Y]
	Cut	[Ctrl] + [X]
	Copy	[Ctrl] + [C]
	Paste	[Ctrl] + [V]
	Delete	[Del]
	Select All	[Ctrl] + [A]
	Find	[Ctrl] + [F]
	Replace	[Ctrl] + [H]
	Find in Files	[Ctrl] + [Shift] + [F]
	Replace in Files	[Ctrl] + [Shift] + [H]
	Go To	[Ctrl] + [G]
	Make Uppercase	[Ctrl] + [Shift] + [U]
	Make Lowercase	[Ctrl] + [U]

Location	Command	Shortcut
	View White Space	[Ctrl] + [E], [S]
	Incremental Search	[Ctrl] + [I]
	Comment Selection	[Ctrl] + [E], [C]
	Uncomment Selection	[Ctrl] + [E], [U]
	Toggle Bookmark	[Ctrl] + [K], [Ctrl] + [K]
	Toggle All Outlining	[Ctrl] + [M], [L]
	Toggle Outlining Expansion	[Ctrl] + [M], [M]
	Start Automatic Outlining	[Ctrl] + [M], [O]
	Stop Automatic Outlining	[Ctrl] + [M], [P]
View Menu	Zoom in	[Ctrl] + [+]
	Zoom out	[Ctrl] + [-]
Debug Menu	Breakpoints	[Ctrl] + [D], [B]
	Output	[Ctrl] + [D], [O]
	Watch 1	[Ctrl] + [D], [W]
	Watch 2	[Ctrl] + [Alt] + [W], [2]
	Watch 3	[Ctrl] + [Alt] + [W], [3]
	Watch 4	[Ctrl] + [Alt] + [W], [4]
	Locals	[Ctrl] + [D], [L]
	Components	[Ctrl] + [D], [P]
	Call Stack	[Ctrl] + [D], [C]
	Memory 1	[Ctrl] + [D], [M]
	Memory 2	[Ctrl] + [Alt] + [M], [2]
	Memory 3	[Ctrl] + [Alt] + [M], [3]
	Memory 4	[Ctrl] + [Alt] + [M], [4]
	Disassembly	[Ctrl] + [Alt], [D]
	Registers	[Ctrl] + [D], [R]
	Program	[Ctrl] + [F5]
	Execute Code (Debug) / Resume Execution	[F5]
	Debug without Programming	[Alt] + [F5]
	Halt Execution	[Ctrl] + [Alt] + [Break]
	Stop Debugging	[Shift] + [F5]
	Rebuild and Run	[Ctrl] + [Shift] + [F5]

Location	Command	Shortcut
	Reset	[Shift] + [Alt] + [F5]
	Step Into	[F11]
	Step Over	[F10]
	Step Out	[Shift] + [F11]
	Toggle Breakpoint	[F9]
	Address Breakpoint	[Ctrl] + [D], [A]
	File Line Breakpoint	[Ctrl] + [D], [F]
	Function Breakpoint	[Ctrl] + [D], [U]
	Variable Watchpoint	[Ctrl] + [D], [V]
	Memory Watchpoint	[Ctrl] + [D], [E]
	Delete all breakpoints	[Ctrl] + [Shift] + F9
Build Menu	Build All Projects	[F6]
	Build Active Project	[Shift] + [F6]
	Cancel Build	[Ctrl] + [Break]
	Compile File	[Ctrl] + [F6]
Help Menu	Help Topics	[F1]
Design Elements Palette	Cancel current operation	[Esc]
	Rectangle Tool	[r]
	Ellipse Tool	[e]
	Line Tool	[l]
	Text Tool	[t]
	Arc Tool	[c]
	Image Tool	[m]
	Wire Tool	[w]
	Sheet Connector	[s]
	Digital Input	[i]
	Digital Output	[o]
	Digital Inout	[b]
	Analog Terminal	[a]
	External Terminal	[x]
Sheet / Page	Move item 1 grid unit or Scroll 1 grid unit	[Up], [Down], [Left], or [Right] *
	Move selected shape 1 point	[Shift] + [Up], [Down], [Left], or [Right]

Location	Command	Shortcut
	Move selected shapes off grid **	[Shift] + Left Click (hold) + Drag
	Disable/enable rubber-banding	[Ctrl] + Left Click (hold) on one or more objects + Drag, or [Ctrl] + [Up], [Down], [Left], or [Right]
	Zoom to selection	[Ctrl] + Left Click (hold) on white space + Drag
	Select multiple items	[Ctrl] + Left Click (successive actions)
	Pan	[Alt] + Left Click + Drag
	Scroll up / down	Mouse Wheel Up / Down
	Zoom in / out	[Ctrl] + Mouse Wheel Up / Down
	Scroll left / right	[Shift] + Mouse Wheel Up / Down
	Large scroll up / down	[Page Up] / [Page Down]
	Large scroll left / right	[Shift] + [Page Up] / [Page Down]
Navigation	Switch between schematic pages.	[Ctrl] + [F6] (Use [Shift] for reverse direction.)
	Switch between open project files or open tabbed documents.	[Ctrl] + [Tab] (Use [Shift] for reverse direction.)
	Close a tabbed document.	[Ctrl] + [w]
	Go back to previous cursor location in Code Editor .	[Ctrl] + [-] (Use [Shift] for reverse direction.)
Code Editor	Autocomplete	[Ctrl] + [Space]
	Find References	[Ctrl] + [Shift] + [R]
	Go To Declaration	[F12]
	Go To Definition	[Ctrl] + [F12]

* If items are selected, using the arrow keys moves items 1 grid unit in the specified direction. If sheet/page is selected, using the arrow keys scrolls instead.

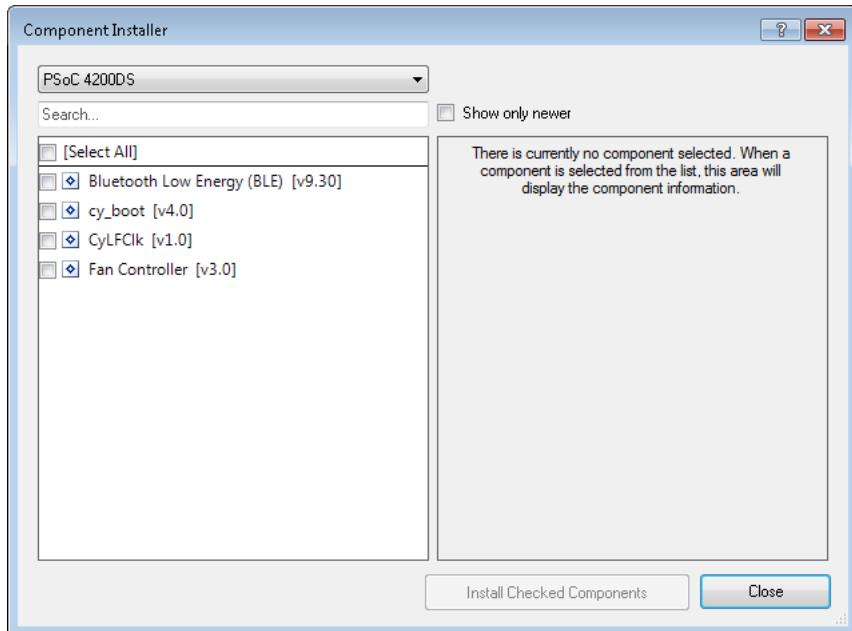
** Components, wires, and other design-specific objects will always snap to grip. Other shapes not part of the actual design can be moved freely.

See Also:

- [File Menu](#)
- [Edit Menu](#)
- [Debugger Menu Commands](#)
- [Build Menu](#)
- [Design Elements Palette](#)
- [Customize Dialog](#)

Component Installer

The Component Installer dialog is used to install new and updated Components from the Cypress web site. It shows all new Components and updated Component versions available online. You can use the pull-down menu to filter by device family/series, as well as a search box to further limit the Components displayed.



To Open the Dialog:

Select **Find new Components** from the **Tools** menu or click the **Find new Components** button  on the [Component Catalog](#).

If you have an open project, the default filter will be derived from the selected device of the active project. If there is no open project, the filter will default to "All."

Use the **Show only newer** check box to limit the Components shown to only those newer than what you already have installed.

To Select a Component:

You can select one or more Components through the tri-state check box.

If you select a Component, the description, image, and datasheet link are downloaded from the web content store and shown on the right hand side of dialog.

To Install a Component:

After selecting one or more Components, click the **Install Checked Components** button.

A download progress bar will display, indicating how many parcel files need to be downloaded in order to satisfy required dependencies. Parcel files will be on the order of 10 MB or less (e.g., BLE_v3_0 compresses to 8 MB), so the progress bar will simply reflect the number of parcel files downloaded.

- While the download/installation is in progress, the button will display as **Cancel** to cancel the process.
- When the download/installation is completed, the button will display as **OK** to close the dialog.

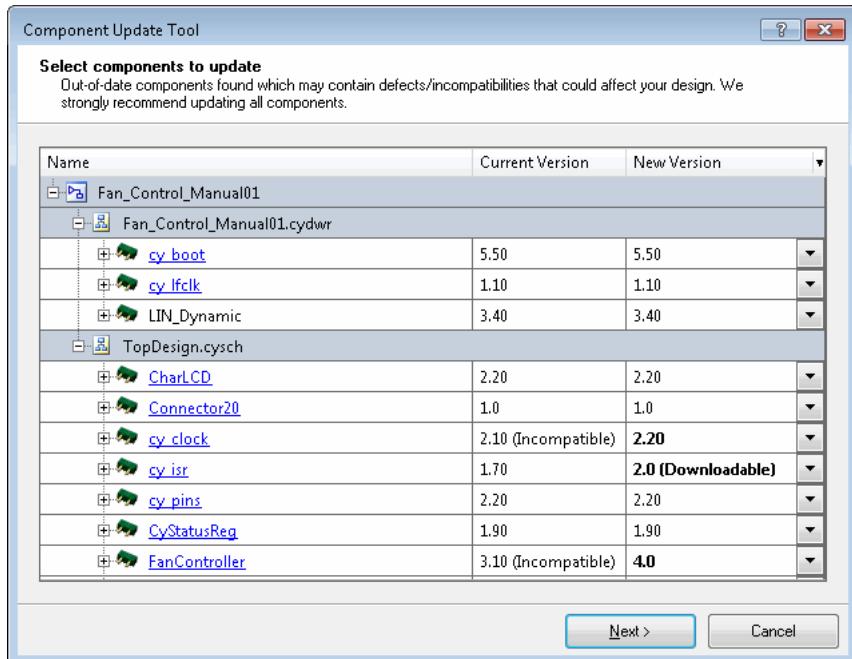
- If there were errors during the download process, they appear in the output window.

See Also:

- [Tools Menu](#)
- [Component Catalog](#)

Component Update

The Component Update Tool dialog allows you to update versions of Components in your designs.



Under the **New Version** column, bold items indicate Components that will have their version updated. By default, PSoC Creator selects the most current version of a Component (including online Components) to be updated.

After a Component update operation has been completed, the tool generates a *ComponentUpdateLog.txt* file, which provides details of the update. This file is stored as in the corresponding project folder.

To Open the Dialog:

1. Open a PSoC Creator design project with one or more Components.
2. Select Update Components from the Project menu.

Notes

- If newer versions of Components are available (including online) in a design, an icon will be added to the status bar as follows:



Click the icon to open the Component Update Tool dialog.

- You can also right-click on the project in the **Source** tab of the [Workspace Explorer](#) and select **Update Components**.

Component Version States:

The Component Update Tool reports the state of various Component versions, under the **Current Version** and **New Version** columns.

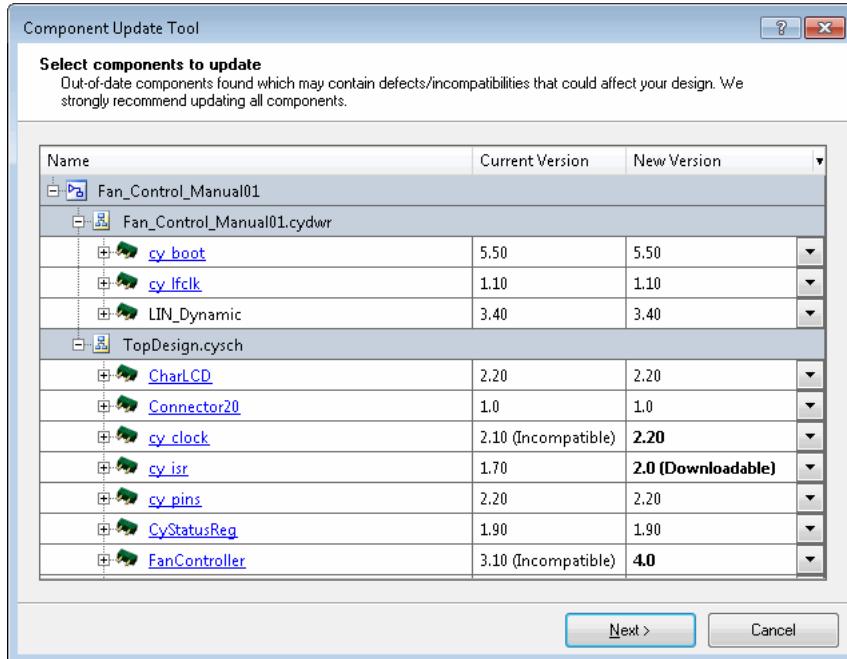
- **Obsolete** – An obsolete Component version should be updated to a newer version. PSoC Creator will display a warning or error about this Component version in the [Notice List Window](#).
- **Prototype** – A pilot Component version is typically a working example only. It is not characterized and not regression tested. PSoC Creator will most often display a note about this Component version in the Notice List Window. In some cases, it will display a warning.
- **Incompatible** – An incompatible Component version means that the selected Component and version is not compatible with the selected device for your design. PSoC Creator will display a warning or error about this Component version in the [Notice List Window](#).
- **Downloadable** – This version of the Component is available online. Selecting it will cause it to be downloaded and installed as a part of the update process.

To View Datasheets:

Click the Component link in the **Name** column at the top of each set of Component instances to display the latest version of the Component's datasheet.

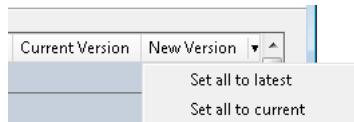
To Update Components:

The Components to be updated are selected automatically and shown in bold under the **New Version** column.

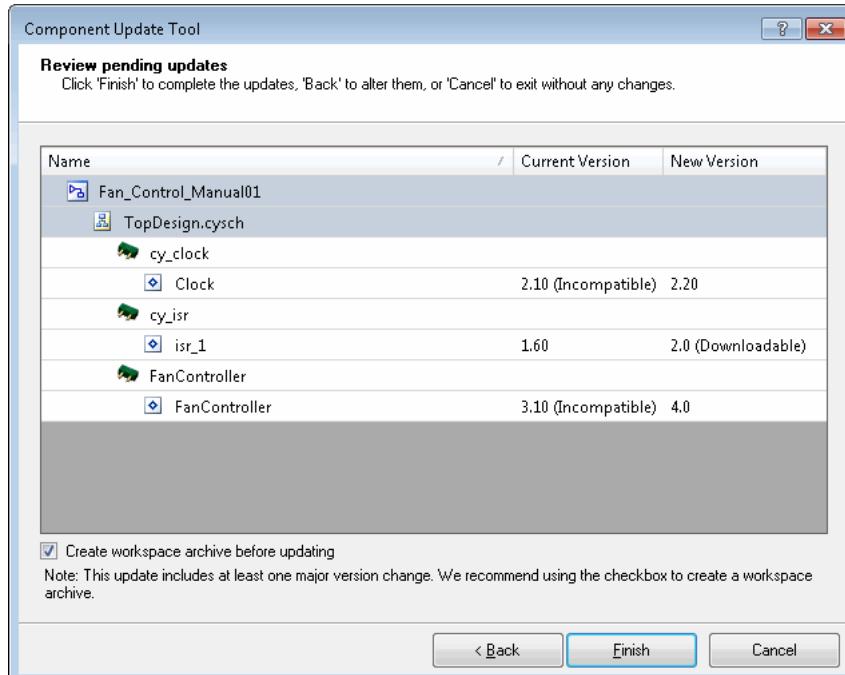


1. As desired, pull down the menu under **New Version** for each Component and select the version you wish to use. You might use this to downgrade a Component to a previous version, for example.

Note If desired, use the menu next to **New Version** to set all Components to the latest version (default) or to the current version.



2. Click **Next**.
3. Review the information on the Review Pending Updates step.



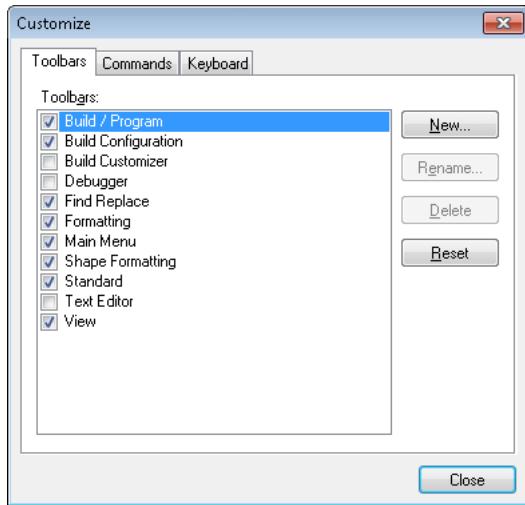
4. The **Create workspace archive before updating** check box will create an [archive of your project](#). Leave selected or deselect to skip the archiving step.
5. Click **Finish** to update the Component(s) or click **Cancel** to cancel.

See Also:

- [Component/Instance](#)
- [Workspace Explorer](#)
- [Notice List Window](#)
- [Environment Options](#)

Customize

The Customize dialog is used to display various toolbars, as well as customize which commands appear on toolbars.



This dialog has three tabs:

- [Toolbars](#)
- [Commands](#)
- Keyboard – This tab and its functionality should not be used.

To Open this Dialog:

Right-click in the PSoC Creator menu/toolbar area and select **Customize**.

Toolbars Tab

The **Toolbars** tab allows you to create, rename, remove, and reset toolbars. It displays the toolbars available and any toolbars you create. When a toolbar is displayed, a check mark appears to the left of it in this dialog box.

- **New** – Displays the New Toolbar dialog box, which allows you to create and name a custom toolbar.
- **Rename** – Displays the Rename Toolbar dialog box, which allows you to change the name of a custom toolbar only.
- **Delete** – Deletes the custom toolbar selected in the Toolbars list.
- **Reset** – Removes any changes to the predefined toolbar selected in the Toolbars list and resets it to its original state. Available only if you select a built-in toolbar.

Note You can only delete and rename toolbars that you created.

Commands Tab

The **Commands** tab allows you to add and remove commands on toolbars and menus.

- **Categories** – Specifies the set of commands that are displayed in the Commands list box. The categories of commands are based on menu titles provided by the tools and designers that the environment is currently supporting. This list of titles is dynamic so that the order of categories and the menu titles change, depending

on the tools and the designer, as well as any customizations made to them. Therefore, it is possible for two menus from different designers to have the same name, so the same title can appear twice but offer different command sets.

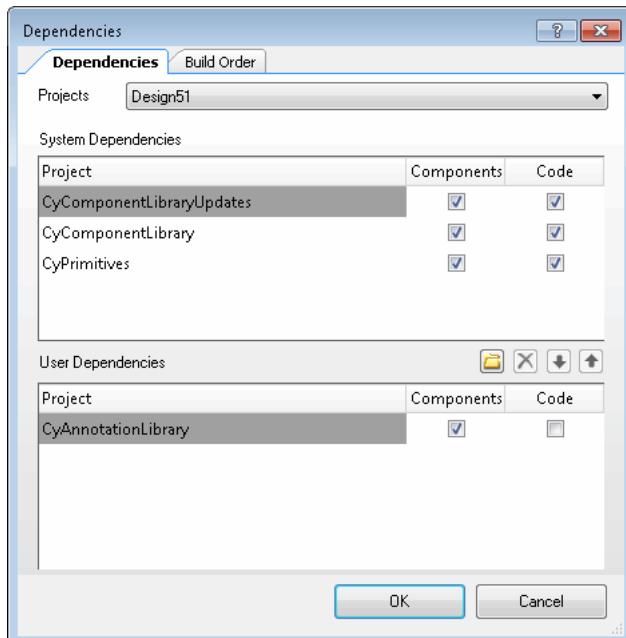
- **Commands** – Displays the commands and command images based on the category you selected. You can drag a command onto the toolbar you want to customize.
- **Modify Selection** – Displays a list of commands you can use to customize the display of the button on the toolbar. For example, you can change the image or accelerator keys, as well as specify whether to display text instead of an image for the command. This button is available after you select a command button on a toolbar you want to customize.

See Also:

- [Framework Description](#)
- [Standard Toolbar](#)

Dependencies

The Dependencies dialog is used to show and select one or more user-level project dependency. The order of the dependencies determines the order that PSoC Creator will use to search for Components and code.



To Open the Dialog:

Select **Dependencies...** from the **Project** menu.

Projects:

This pull down allows you to select the project for which you want to include a dependency. This only applies if there are multiple projects in your workspace.

System Dependencies:

This area shows the system-level dependencies for your project.

For each dependency there are two check boxes: **Components** and **Code**. **Components** determines whether PSoC Creator searches the project for Components to include in the Component Catalog. **Code** determines whether a C static library is linked in to the project.

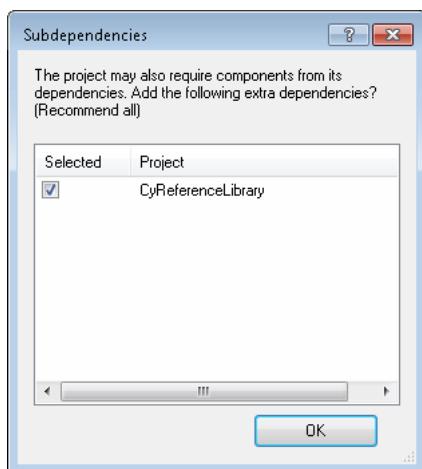
User Dependencies:

This area is where you can add/remove other dependencies to your project. It contains four buttons: **Add**, **Remove**, **Move Up** and **Move Down**. This section also contains **Components** and **Code** check boxes described under [System Dependencies](#).

Note If there were any default dependencies added to the [Project Management Options](#) dialog under Default Dependencies, those dependencies will be shown in this area for projects created after the dependencies were added.

Subdependencies

If you have multiple projects in your workspace, there are cases where one or more projects use Components defined in another project in the same workspace. In this case, if you select the **Components** check box next to the project where the Component is defined, the Subdependencies dialog will display.



Project dependencies are a list. For example, suppose a workspace has "Project A" that depends solely on "Project B" that depends solely on "Project C."

- Project A will not use Components or code from Project C, unless Project A has its own direct dependency on Project C.
- If modifying dependencies for Project A, selecting Subdependencies for Project B would show Project C.
- If adding a new dependency for Project A, the Subdependencies dialog makes it easy to add any other dependencies that may be necessary.
- If you do not add Project B Subdependencies for Project A **may** cause DRC errors or link errors in Project A.

Build Order Tab:

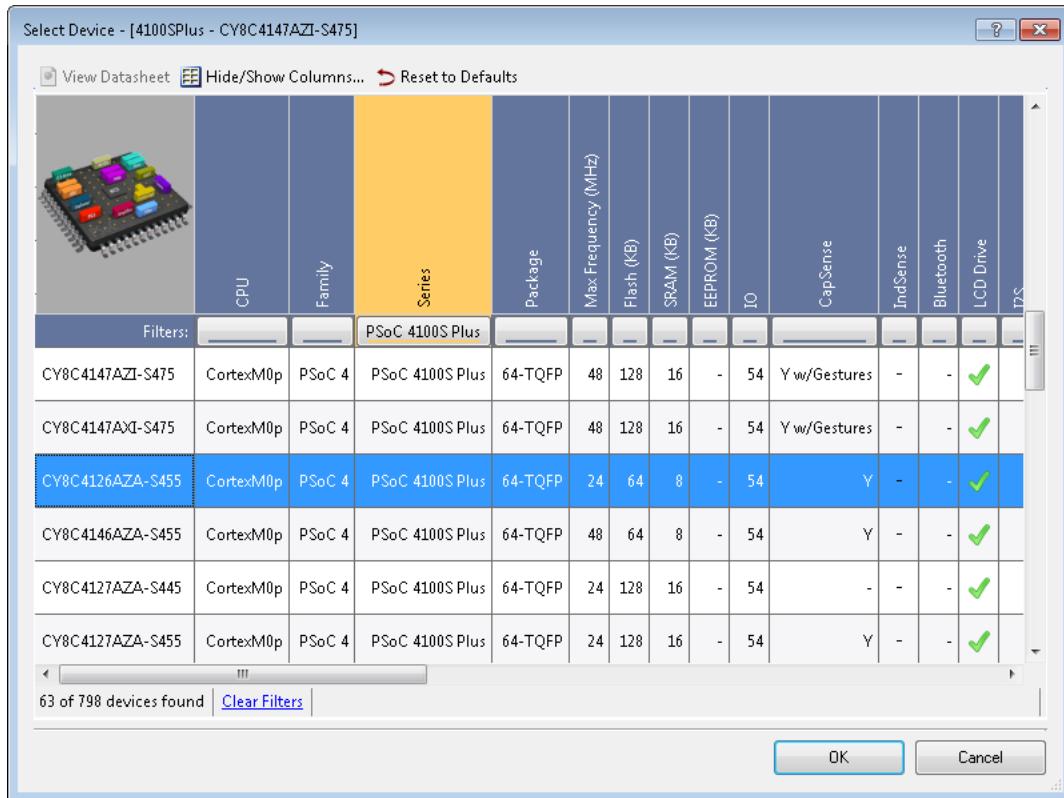
The **Build Order** tab is a read-only list of the order in which your projects will build.

See Also:

- [Basic Hierarchical Design](#)
- [Project Management Options](#)

Device Selector

Use the Device Selector dialog to choose which device to use in your design.



The dialog provides a list of available devices, as well as various characteristics for each device. Using the dialog, you can:

- change the selection at any time during the design process
- customize how/what data is displayed for the devices
- filter which columns should be displayed/hidden

Notes

- The title bar of the dialog displays the name of the project that the device is being set for and the name of the currently selected device.
- The Device Selector layout is saved on a per project basis. When you re-open the Device Selector for a particular project it will retain all the information from the previous session.
- The Device Selector allows for targeting a project to a specific Silicon Revision. This will cause changes in the generated code such that it can only be programmed on a target device that has a matching Silicon Revision.

To Open the Dialog:

Select the design project for which you want to change the selected device in the [Workspace Explorer](#) and do one of the following:

- Select Project > Device Selector..., or
- Right-click on the project and select **Device Selector...**

Note You can also open the Device Selector from the [New Project dialog](#) **Device** pull down menu when creating a new design project. The major difference is that when opening the Device Selector for a new design, none of Auto-Select features are applicable and you will only see devices for the project type you have selected.

To Select a Device:

To select a device, do one of the following:

- Double-click a device row in the device table, or
- Click a device row and click **OK**.

To quickly select the default device for a specific architecture, right-click anywhere in the device selector table area and select **Select Default Device**. Then pick your desired architecture.

To Sort a Device Category:

You can sort by any column in ascending or descending order by clicking on the header for that column. Clicking again will toggle the sort direction. When a column has a sort applied to it, an arrow will be displayed under the text in the column header.

By default, the part number column in the device selector has the sort applied to it. To re-sort on the part number, right-click anywhere in the device selector table area, select **Sort by Part Number**, and pick your desired sort direction. This menu also provides a way to automatically scroll to whatever device is currently selected.

To View Device Datasheet:

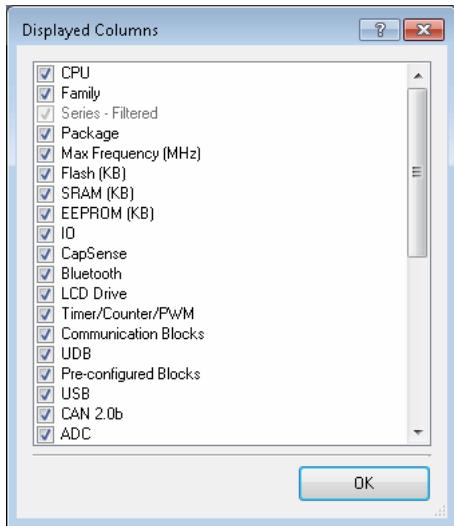
Click **View Datasheet**.

This opens a web page to the selected device's datasheet.

To Show/Hide Columns:

Click Hide/Show Columns.

This is used to hide/show in the device table. This opens Display Columns dialog that lists all the available columns.



You can then select/unselect columns to alter what is displayed in the device table.

Note These changes are made live as you change the state of the check boxes. Columns that have filters applied cannot be hidden. They are disabled in the dialog and are followed by the text “- Filtered” so you know why they cannot be hidden.

Status information displays above the device table, showing how many columns are hidden. If no columns are hidden, no status information is provided.

To Filter the Device Table:

Each column in the Device Selector has a drop-down button used for filtering on the column. When pressed, a drop-down window displays a list of check boxes, one for each of the different values possible in that column. You can unselect as many of the values as desired, removing devices with those values from the set of displayed devices. These changes are not applied to the device table until you click off the window or press the [**Enter**] key (pressing [**Esc**] will close the window without applying any of the changes). By default all values are checked meaning that no filtering has been done.

When filtering has been applied on a column, the column header (and the accent color on the button) changes to make it very obvious which columns have filters applied. The button also displays the value that has been left “on”. If more than one value has been left “on,” only the first value will be displayed on the button. The value will be followed by “...” indicating that more values are being displayed. In this case hovering the mouse over the button will produce a tool tip that displays all the values that are currently “on”.

There is a status bar just below the device table that shows how many devices are currently being displayed (i.e. how many devices passed all the filters) versus how many devices exist. There is also a link here that clears all the filters from the table making all the devices visible again.

To Reset to Defaults:

Click **Reset to Defaults**.

This resets all the settings back to their defaults (removes filters, re-orders columns, resets which columns are displayed, etc.).

See Also:

- [Workspace Explorer](#)
- [Creating a New Project](#)

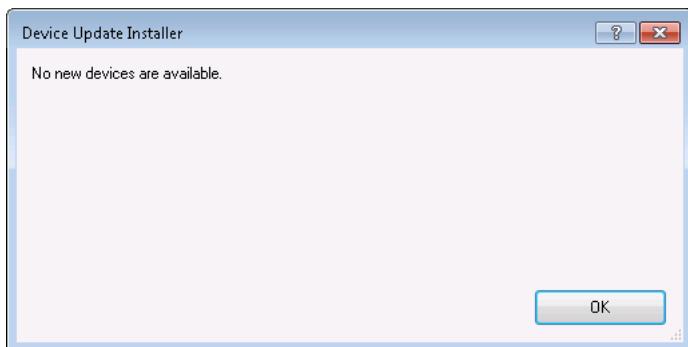
Device Update Installer

The Device Update Installer dialog is used to download and install new device support available on the Cypress web page.

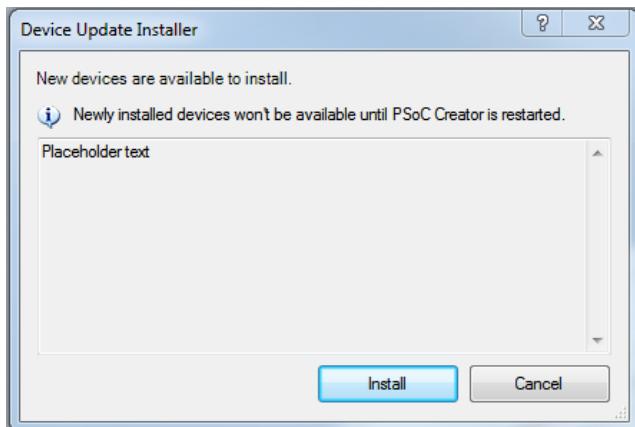
To Open the Dialog:

Select **Find new Devices** from the **Tools** menu.

If no new device support is available, a "No new devices are available." message displays.



If new device support is available, the following dialog will display.

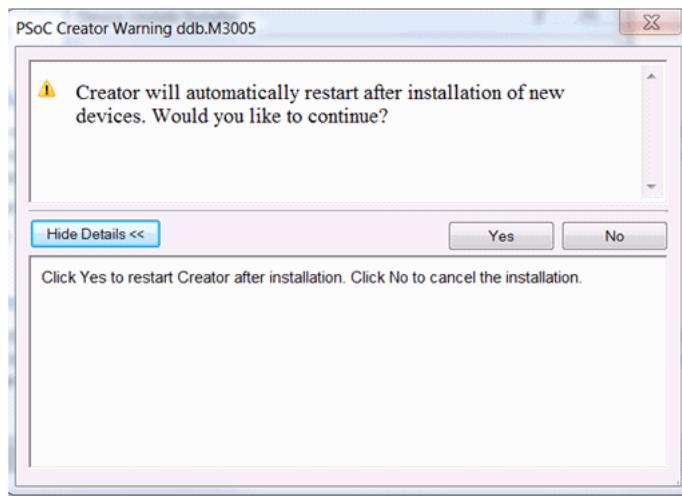


To Install/Download a Device:

Select one or more devices and click the **Install** button.

Once the download is completed, click **OK** to close the dialog.

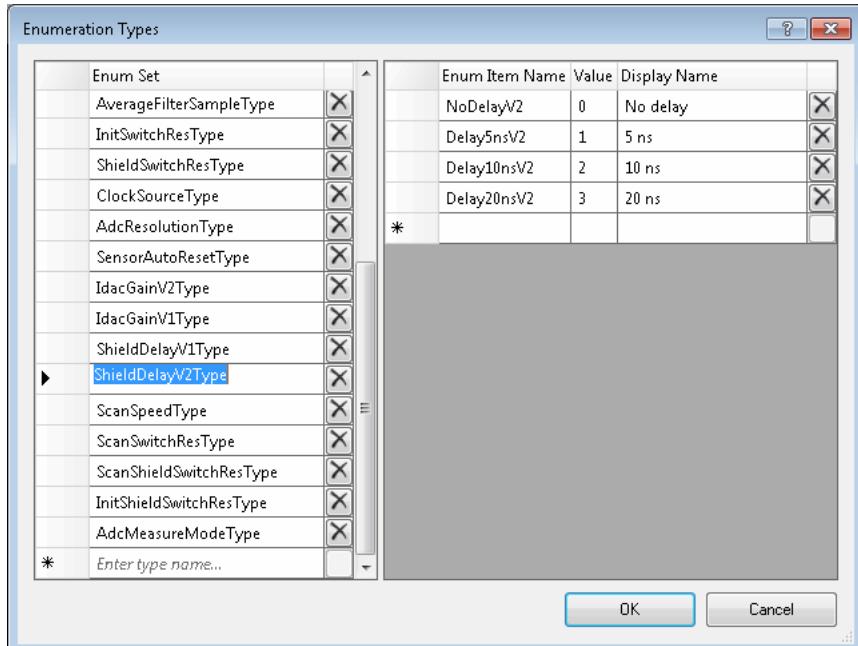
A pop up message will display to indicate that PSoC Creator needs to restart.



Click **Yes** to restart; click **No** to cancel the installation.

Enumeration Types

The Enumeration Types dialog allows you to create and edit enumerations (or user-defined types) for your Components. This dialog is used in conjunction with [defining parameters for a Component](#). The process of creating Components can be complex. This topic is provided as a help if you press [F1] for this dialog. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).



A Component can contain multiple types but each type name must be unique and each key in the types must be unique across all other types in that Component. The types defined for each Component in a given project are aggregated on the project (the project type cache) and the project type caches from all projects on a design's search path are aggregated into a design type cache.

- Internal to the symbol, the type and the type keys are accessible by their "short name."
- Outside of the symbol, the type and keys are identified using the "long name," which is the Component name followed by the actual type or key name (using "__" to separate the two).

When evaluating a type or key for an expression, the evaluation system first looks in the enclosing symbol for the identifier. If it is not found there, the system queries the design type cache. If the design type cache does not recognize the identifier, it is reported as an unknown identifier.

To Open the Dialog:

Click the **Types...** button on the [Parameters Definition dialog](#).

To Add an Enumeration Type:

1. Enter a name at the bottom of the 'Enum Set' table (where it says 'Enter type name...').
2. In the Enum Item table, click in the empty row under **Enum Item Name** and type a name for the 1st name/value pair of the enumerated type; type a value under **Value** or accept the default.
3. Optionally, enter a string in **Display Name** that will display in the Component's Configure dialog pull down menu for that parameter.
4. Enter as many enum sets as needed and click **OK** to close the Enumeration Types dialog..

To Delete an Enumeration Type:

Click **Delete**  next to the item to delete.

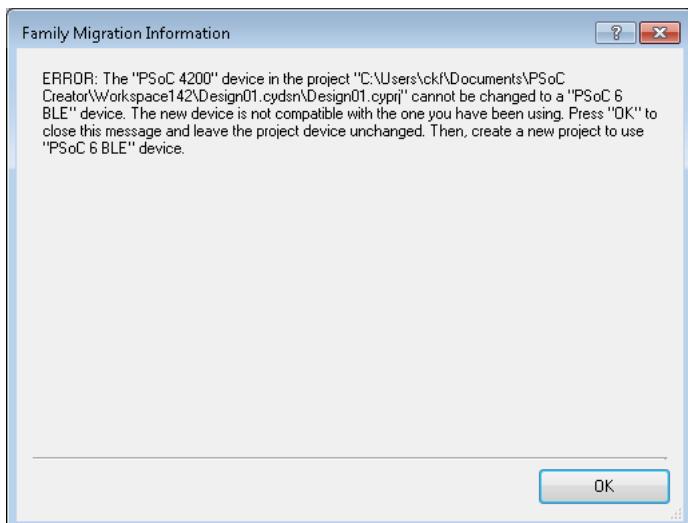
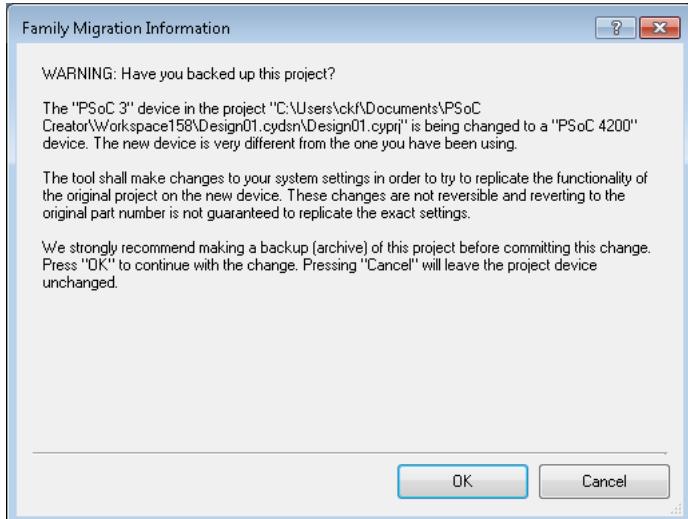
See Also:

- [Symbol Editor](#)
- [Parameters Definition dialog](#)
- [Component Author Guide](#)

Family Migration Information

The Family Migration Information dialog displays when you try to change a project from one device family to another family and vice versa. The dialog warns you that the device change being made is likely to require manual edits to the design, or that the migration is not supported.

Note This dialog text will change depending the devices involved. Here are a couple examples.



To Open this Dialog:

This dialog opens when you try to change your project to or from a PSoC 4 device using the [Device Selector](#).

To Use this Dialog:

Click **OK** to close the dialog and proceed with changing the device.

Click **Cancel** to close the dialog and leave the project device unchanged.

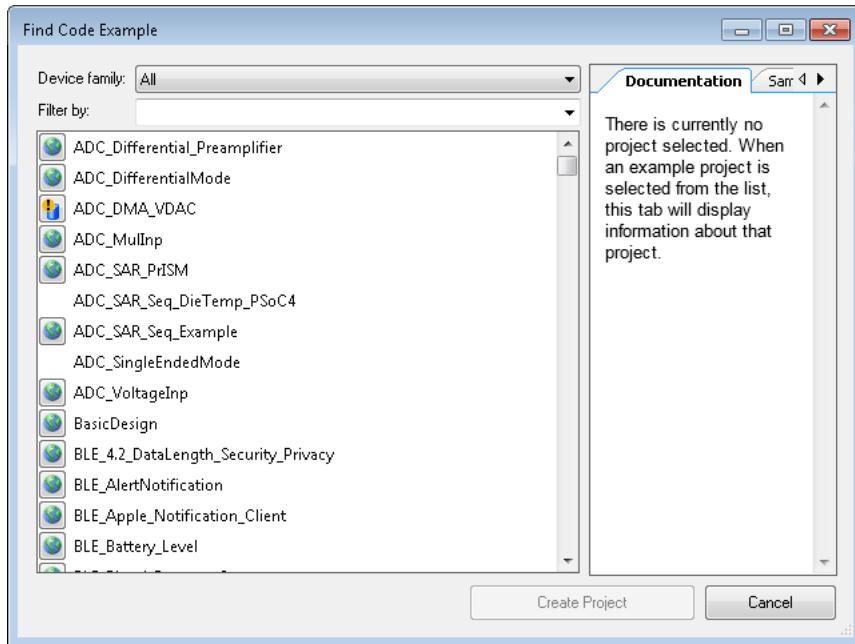
See Also:

- [Device Selector](#)

- [Options Dialog](#)

Find Code Example

The Find Code Example dialog allows you to search for and open code examples installed on disk with PSoC Creator, as well as download and install code examples from the Cypress web site. Code examples show how various Components can be configured, and they include sample code to provide a better understanding of how a Component can be used.



Most code examples have a description document that provides setup and configuration information. This document is displayed under the **Documentation** tab when you select a project from the list. Also available is example code located under the **Sample Code** tab.

Code examples shown in the list will have different indicators, as follows:

- **Install** – These are code examples that are online only and need to be installed before they can be used.
- **Update** – These are code examples that have newer versions online than the code example already installed on disk.
- **None** – These are code examples installed on disk with no updates available.

To Open the Find Code Example Dialog:

All Examples

To open the dialog with all code examples available, select **Find Code Example...** on the Start page, or select **Code Example...** under the **File** menu.

Component-Specific Examples

To open the dialog with a list of code examples associated with a corresponding Component, right-click on a Component in the [Component Catalog](#) or on the design canvas, and select **Find Code Example...**

To Install/Update a Code Example:

Click the **Install** or **Update** button, as appropriate.

A download progress dialog displays. When completed, click **OK** to close the download progress dialog. This will also update the status of applicable code examples in the Find Code Example dialog.

Also, if a new/updated code example includes new or updated versions of Components than those already installed on disk, then the new Components and Component versions will also be installed.

To Select a Code Example:

1. As needed, choose filters under **Device Family** or **Filter by** to narrow the list of available projects.
 - Select the **Device Family** for the project. For example, All, PSoC 3, PSoC 4000, PSoC 4200 BLE, PSoC 5LP, etc.
 - If desired, also select a keyword from the **Filter by** pull-down menu or type the project name or other words. This could be the name of a Component or different keywords used by code example developers to distinguish projects.
 2. Select a project or workspace from the list matching the given filtering criteria.
- Note** It is possible that multiple projects will be contained in an example workspace. If any of the projects in a workspace matches the filtering criteria, the workspace and all of its projects will be shown.
3. View the documentation and/or code for the project by clicking the appropriate tab, if desired.
 4. Click either **Create Project** or **Create Workspace**, as appropriate. The [New Project wizard](#) opens to complete the project/workspace creation process.

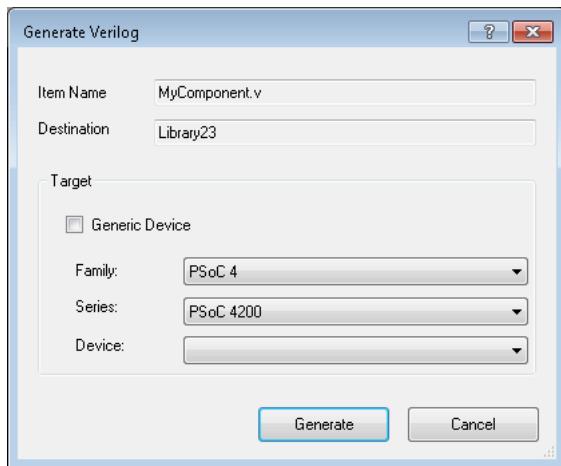
Note If an online only code example has not been installed, an error message will display indicating that you must install it before it can be created. If a code example update has not been installed, the previously installed version of the code example

See Also:

- [Creating a New Project](#)
- [Workspace/Project](#)
- [Component/Instance](#)

Generate Verilog

The Generate Verilog dialog allows you to choose the architecture, family, and/or device for the Verilog file being generated for your Component symbol. This dialog is used as part of creating a Component. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).



To Open this Dialog:

This dialog opens automatically when you select the **Generate Verilog** command from the [Symbol canvas context menu](#).

To Choose Architecture/Family/Device:

The **Target** options may be used to specify a particular architecture, family, and/or device to which the Verilog file applies.

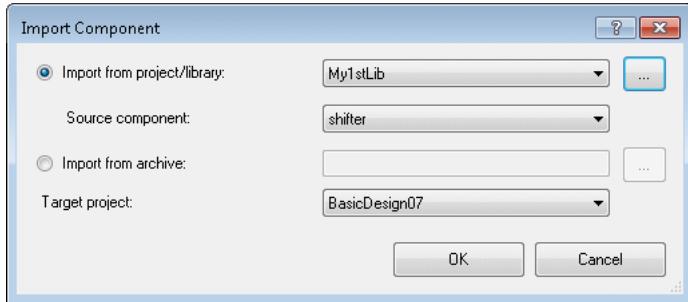
- Leave the default setting **Generic Device** selected to allow the Verilog file to apply to all devices. Deselect the check box to enable the **Family**, **Series**, and **Device** pull down menus.
- Choose a **Family** to create the Verilog file in a subfolder for a family.
- Choose a device **Series** to create the Verilog file in a subfolder for a series of devices.
- Choose a specific **Device** part number to create the Verilog file in a subfolder for a specific device.

See Also:

- [Component Author Guide](#)
- [Symbol Editor Context Menus](#)
- [Adding a Component Item](#)

Import Component

The Import Component dialog is used to import a Component from another project. It is also used to import one or more Components from a Component archive file.



Note Imported Components retain all of the symbol and Component properties of the original Component. This includes [Component Catalog Placement](#). If there are duplicate Components from different [Dependencies](#) in your project, the [Component Catalog](#) will display the Component from the dependency with the highest precedence.

The dialog has the following options:

- **Import from project/library** – Used to choose the project/library that contains the Component to import.
 - **Source Component** – When importing from a project/library, this option is used to specify which Component from the project/library to import.
- **Import from archive** – Used to select the Component archive that contains the Component(s) to import. For more information, see [Exporting a Component](#). All Components in the archive file will be imported.
- **Target Project** – Used to select the specific project for which the Component will be imported.

To Open this Dialog:

1. Select the **Components** tab in the [Workspace Explorer](#).
2. Right-click on a project and select **Import Component**.

To Import a Component:

3. Select either Import from project/library or Import from archive.
 - If you selected **Import from project/library**, select the project that contains the Component to import. If necessary, click the browse button [...] and navigate to the folder containing the Component to import. In **Source Component**, select the Component to import from the project/library.
 - If you selected **Import from archive**, select the appropriate Component archive (.cycomp file). If necessary, click the browse button [...] and navigate to the folder containing the Component archive.

4. In **Target Project**, select the project where the Component will be placed.
5. Click **OK**.

PSoC Creator adds the imported Component(s) to the selected project.

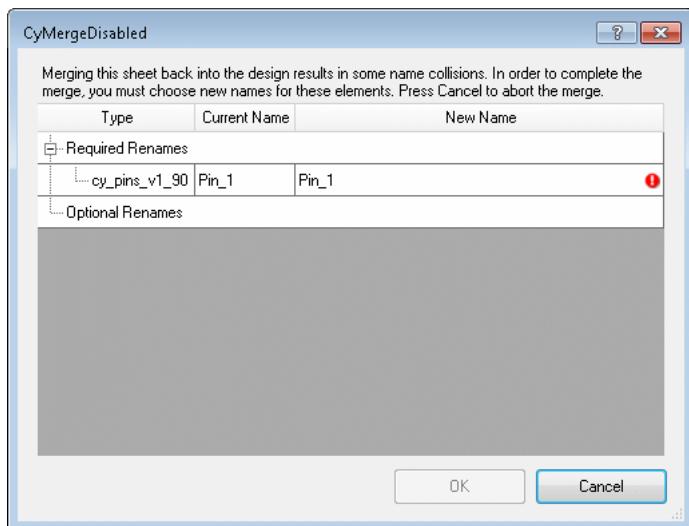
See Also:

- [Dependencies](#)

- [Component Catalog](#)
- [Defining Catalog Placement](#)
- [Exporting a Component](#)
- [Workspace Explorer](#)

Merge Dialog

The Merge dialog displays when you try to enable a [Disabled Schematic page](#) and there are one or more Component and/or wire instances that have the same name.



To use the dialog:

Select the instance in the **New Name** field, type a different name for the instance, and click **OK**.

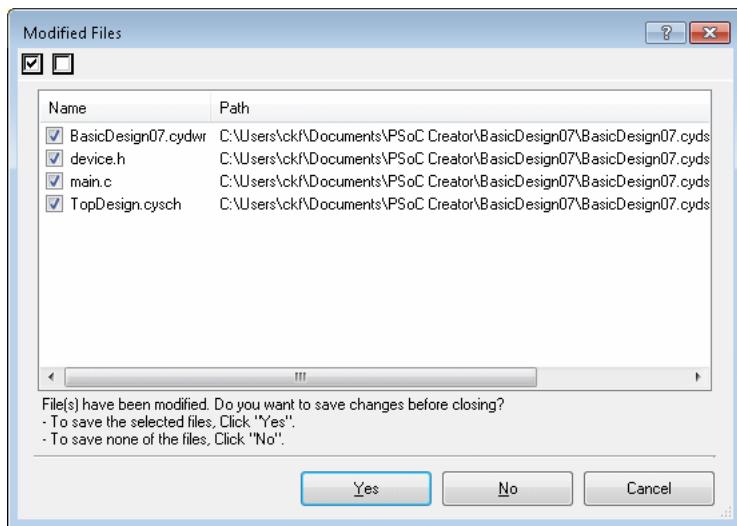
Note Component instances must be renamed in order to resolve the name conflict. Some wire and signal instances do not have to be renamed. The **OK** button will not be enabled until all mandatory conflicts have been resolved.

See Also:

- [Using Disabled Schematic Pages](#)

Modified Files

The Modified Files dialog allows you to selectively save files when closing a project or workspace.



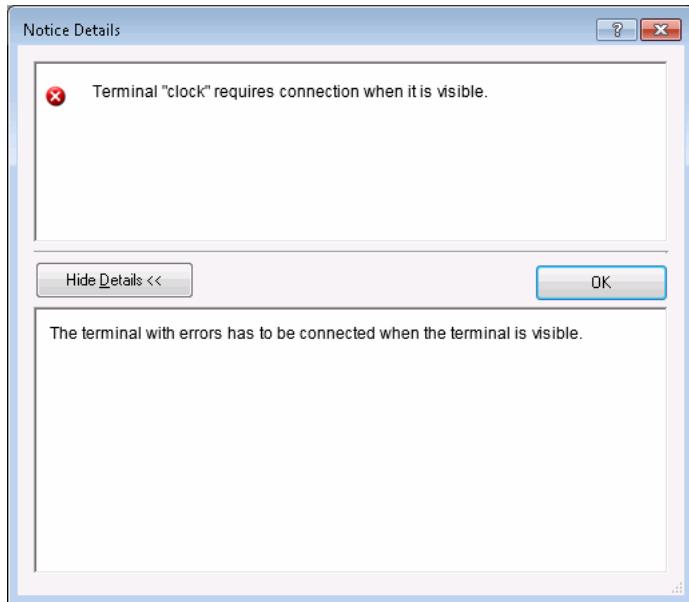
This dialog displays automatically if you attempt to close a file or workspace/project that needs to be saved.

To Use this Dialog:

- Click **Yes** to save the selected files.
- Click **No** to save none of the files.
- Click **Cancel** to close the dialog and leave the files open.

Notice Details

The Notice Details dialog displays expanded information for messages in the [Notice List window](#). The Notice Details dialog will display the entire message, as well as additional information if available.



To Open this Dialog:

On the Notice List window, click the  button next to the message or click the **View Details** button.

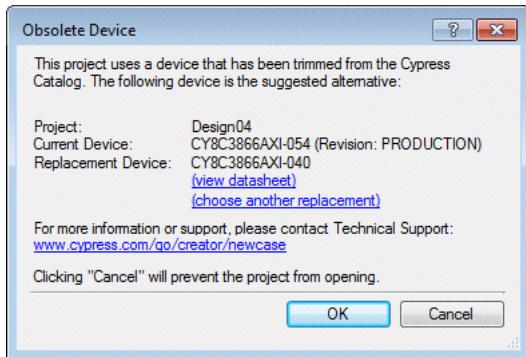
There is also a command available to view details if you right-click on a notice.

See Also:

- [Notice List Window](#)

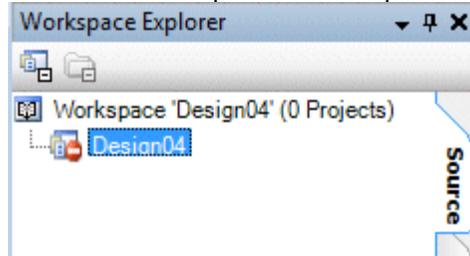
Obsolete Device

The Obsolete Device dialog displays when you open a project with a device that is no longer in the Cypress device catalog. This dialog suggests the next best alternative for your device, and provides a link to the suggested device datasheet for more information.



This dialog provides you with three options:

- Click **OK** to open your project with the suggested device.
- Click **choose another replacement** to open the [Device Selector](#) and choose a different device.
- Click **Cancel** to open the Workspace with the project unloaded.



Note The project will also be unloaded if you close the Device Selector without selecting a device.

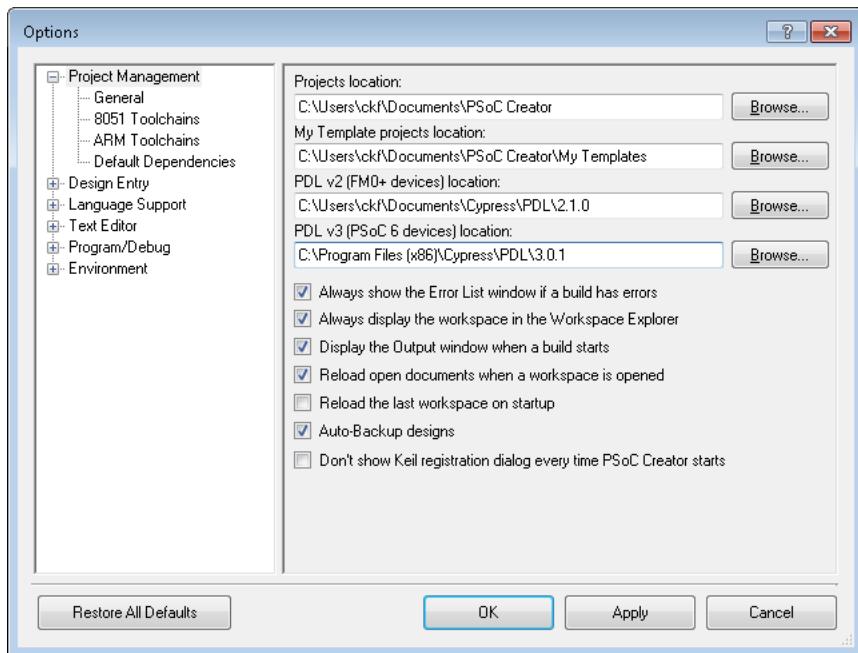
If you reload the unloaded project, PSoC Creator will select the default device for your project. You can use the Device Selector to change it, if needed.

See Also:

- [Device Selector](#)
- [Opening an Existing Project](#)

Options Dialog

The Options dialog allows you to specify various settings for PSoC Creator, such as where projects are stored or the color of wires.



The dialog is divided into the following categories:

- [Project Management](#) (this topic)
- [Design Entry](#)
- [Language Support](#)
- [Text Editor](#)
- [Program/Debug](#)
- [Environment](#)

To Open the Options Dialog:

Select **Options** from the **Tools** menu.

To Restore Defaults:

Click **Restore Defaults**.

All options revert to the default configuration.

Project Management Options:

The Project Management category of the Options dialog is used to set various options for project management.

This category includes the following sections:

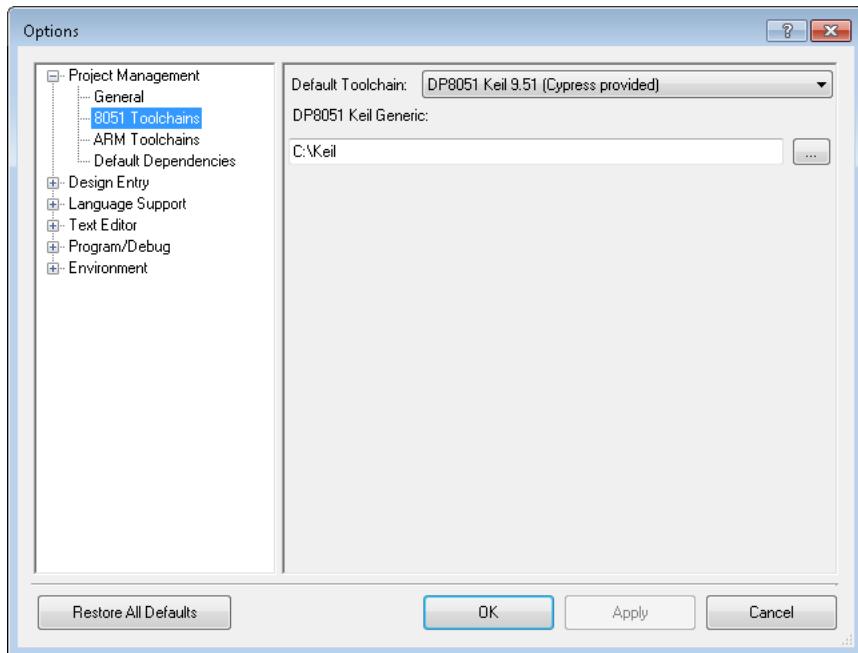
General:

The General area contains various project management options.

- Projects location – Used to specify the default location to save your projects.
- My Template projects location – Used to specify the default location to save your [My Template projects](#).
- PDL v2 (FM0+ devices) location – Used to specify the default location where projects will look for the Peripheral Driver Library (PDL) version 2, for FM0+ devices. You can also specify a custom PDL path per project on the [Peripheral Driver Library Build Settings](#) dialog.
- PDL v3 (PSoC 6 devices) location – Used to specify the default location where projects will look for the PDL version 3 for PSoC 6 devices. You can also specify a custom PDL path per project on the [Peripheral Driver Library Build Settings](#) dialog.
- Always show the Error List window if a build has errors – yes (default) or no.
- Always display the workspace in the Workspace Explorer – yes (default) or no.
- Display the Output window when a build starts – yes (default) or no.
- Reload open documents when a workspace is opened – yes (default) or no.
- Reload the last workspace on startup – yes or no (default).
- Auto-Backup Designs – yes (default) or no. If this option is selected, when you open designs created with previous versions of PSoC Creator, the older designs will be backed up to an archive (located in [Workspace Folder]/Backup). This backup “copy” will contain the version of the tool that was last used to save it (that is, the older version it can be opened with) in the file name.
- Don't show Keil registration ... – yes or no (default). If selected, the [Keil Compiler Registration](#) dialog will not display every time you start PSoC Creator.

8051 Toolchains:

This section is used to set the default 8051 toolchain.



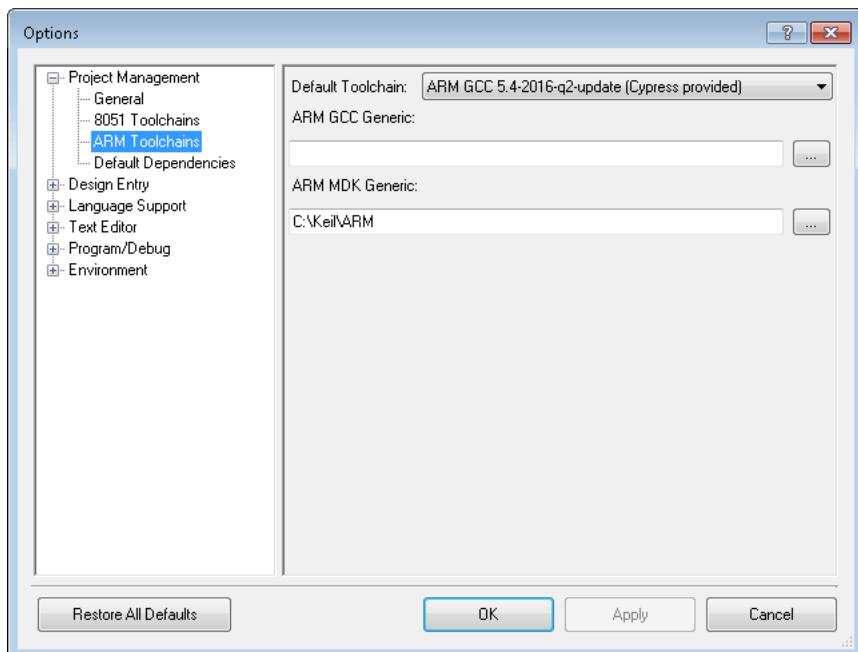
This option also allows you to specify paths to the binaries for each specific toolchain.

- **DP8051 Keil Generic** – The default install location is C:\Keil\C51\BIN\

Example files that should be in the selected folder: *A51.EXE, C51.EXE*

Arm Toolchains:

This section is used to set the default Arm toolchain.



This option also allows you to specify paths to the binaries for each specific toolchain.

- **Arm GCC Generic** – There is no default install location. It can be in whatever folder you extract the files.

Example files that should be in the selected folder: *arm-none-eabi-addr2line.exe*, *arm-none-eabi.ar.exe*

- **Arm MDK Generic** – The default install location is C:\Keil\Arm\BIN40\

Example files that should be in the selected folder: *armar.exe*, *armasm.exe*

Note Some toolchains require environment variables to find its bin, include, and library paths. These variables are set by the toolchain installer. You may have to restart PSoC Creator in order to use the new environment variables.

Default Dependencies:

This option allows you to specify default libraries that appear as user dependencies on the [Dependencies](#) dialog for all newly created projects. Default dependencies apply to all new projects on a per user basis, so they will be the included for all your new projects.

Note Projects created with PSoC Creator 1.0 and initially opened in the current version of PSoC Creator will be updated to include all default dependencies defined at that time. Existing projects for the current version of PSoC Creator will **not** be updated to include any default dependencies added in the future.

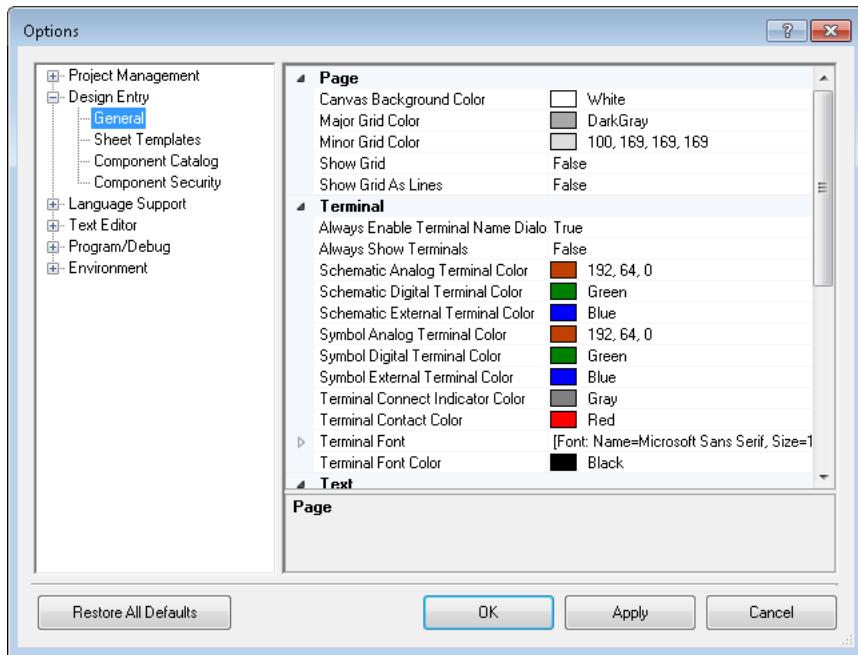
This area contains four buttons: **Add**, **Remove**, **Move Up** and **Move Down**. For each dependency there are two check boxes: **Components** and **Code** to specify whether or not there is a dependency on the search path, code, both, or neither.

See Also:

- [Keil Compiler](#)
- [Dependencies](#)
- [Toolchain Documentation](#)

Design Entry Options

The Design Entry category of the Options dialog is used to set various options for the design entry tools.



This category includes the following sections:

General:

The General section contains numerous fields to specify preferences for colors, widths, sizes, and fonts for most of the objects in the design entry tools. This section is divided as follows:

- **Page** – settings for the display of the canvas, including grid type and color.
- **Terminal** – settings for terminals including colors and fonts.
- **Text** – settings for labels including whether to display expressions and the color or owner lines.
- **Wire** – settings for wires including colors, widths, and fonts. There is also a setting for rubber-banding. When this option is turned on, wires will automatically be redrawn when you move objects. If rubber-banding is turned off, wires will not be redrawn and connections will be broken. Press the **[Ctrl]** key while moving objects to temporarily disable or enable rubber-banding, respectively.

Sheet Templates:

This section allows you to specify locations where PSoC Creator will look for sheet template files to be used with design entry tools. See [Sheet Template Editor](#) for information about creating sheet templates.

Use the **Add** button to add more paths to locations for sheet templates; use the **Remove** button to remove paths.

Component Catalog:

This section contains Component options:

- **Show Hidden Components** – Selecting this check box displays all the hidden Components in the [Component Catalog](#), including primitives. Primitives are basic Components that are used as building blocks in the typical Components found in the catalog.

- **Enable Param Edit Views** – Selecting this check box allows you to view the Expression View in each Component's [Configure dialog](#) by right-clicking on the different tabs in that dialog.

Warning Switching to the Expression View is an advanced feature. It requires a thorough knowledge of the valid parameter settings for the Component. Using this view makes it possible to create combinations of parameters that are not valid. Therefore, you may need to cancel any changes and restart the process if you cannot find valid parameters.

- **Remember Dialog Sizes** – Selecting this check box makes it so that Components' Configure dialogs will save/restore their size on a per Component basis. Unchecking this box will cause the dialogs to open as their default size instead.

Note If the saved size is larger than the current screen, the size will be adjusted to fit the screen.

- **Reset Sizes** – Clicking this button will cause all the currently saved dialog sizes to be reset to the default size for their Components. This operation cannot be undone.

Component Security:

Allows you to add and remove paths to allowed third party customizers that can be used within PSoC Creator. PSoC Creator will generate errors for third party customizers not included in this section. When you try to open an external project, PSoC Creator displays the following:

"Project XXX contains Components that include special code that runs on your machine. Do you trust the source of the project and wish to use the projects Components."

If you select Yes, then the path of the project is added to the relevant setting. After this, whenever you open this particular project, no messages will be shown and the project will load and build normally.

If you select No, then the project will load with an error:

"Component YYY is not available. Component Security is disabled for the project XXX. If you trust the author of the project and wish to allow this Component to run code on your computer during the design process, select Tools > Options > Design Entry > Component Security and add the project folder to the approved list."

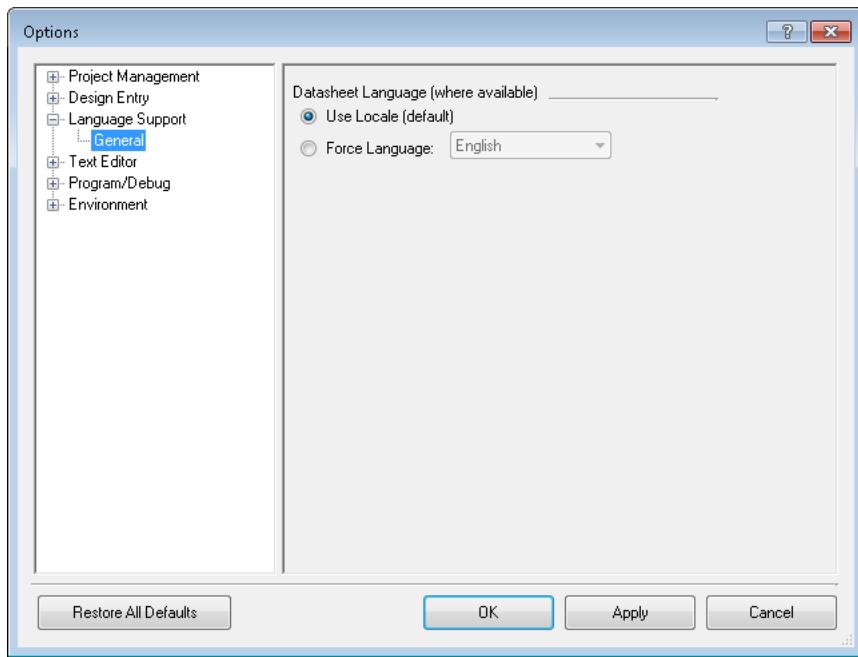
After this, whenever you open this project it will show this error until you add the path.

See Also:

- [Options Dialog](#)
- [Using Design Entry Tools](#)
- [Sheet Template Editor](#)
- [Component Catalog](#)
- [Configure dialog](#)

Language Support Options

The Language Support section of the Options dialog contains settings for using an installed datasheet language pack.



General Options:

The following options are available under the General (default) section:

Datasheet Language

Specify whether to **Use Locale** or to **Force Language**. If a datasheet language pack is installed, the **Force Language** option will enable a pull-down menu with different languages to select.

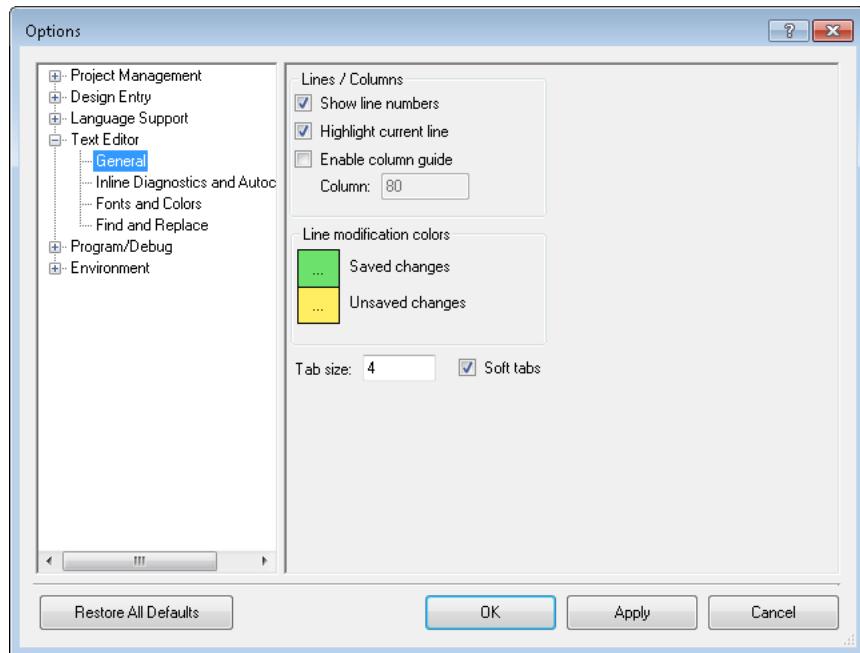
When opening a datasheet, if the specific language and version is not available, the [Select Datasheet](#) dialog will display to select an appropriate datasheet.

See Also:

- [Options Dialog](#)
- [Select Datasheet](#)

Text Editor Options

The Text Editor category of the Options dialog is used to set various options for the code editor.



This category includes the following sections:

- [General](#)
- [Inline Diagonistics and Autocomplete](#)
- [Fonts and Colors](#)
- [Find and Replace](#)

General:

This section provides options to change various editor behaviors.

Lines/Columns

- **Show Line Numbers** – This check box allows you to control whether or not to display the line numbers.
- **Highlight Current Line** – This check box allows you to enable and disable the current line indicator, which shades the current line.
- **Enable Column Guide** – This check box allows you to enable the column length guide. If enabled, you can specify the column length in characters.

Line Modification Colors

These options allow you to change the color shown in the margins for **Saved Changes** and **Unsaved Changes**.

Click the colored box to display the Color selection dialog.

Tab size

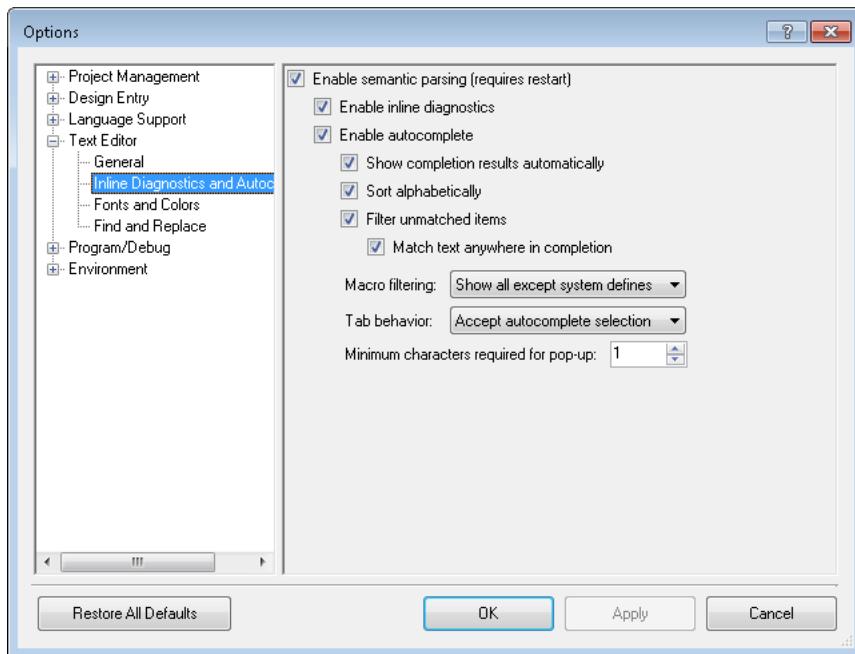
The Tab Size field allows you to specify the tab length in characters.

- If you enable the **Soft Tabs** feature and then press the [Tab] key, the text editor uses the specified number of spaces instead of a tab key stroke.

Note Turning the **Soft Tabs** feature on or off does not affect previously inserted tabs.

Inline Diagnostics and Autocomplete:

This section provides options to change settings for inline diagnostics and autocomplete:



Enable semantic parsing

This controls whether the advanced editor features such as autocomplete, inline diagnostics, Go To definition, and the code explorer tool window are enabled.

- Enable inline diagnostics** – If the **Enable semantic parsing** check box is selected, this check box is enabled to turn on and off the [inline diagnostic](#) feature.
- Enable autocomplete** – If the **Enable semantic parsing** check box above is selected, this check box is enabled to turn on and off the [Autocomplete](#) feature.
 - Show completion results automatically** – If the **Enable autocomplete** check box is selected, this check box is enabled to show the completion results. If checked, items will be shown; otherwise, they won't.
 - Sort alphabetically** – If the **Enable autocomplete** check box is selected, this check box is enabled to turn on and off the sorting mechanism for autocomplete. If checked, items will be arranged alphabetically; otherwise, they will be arranged in order of anticipated value.
 - Filter unmatched items** – This option causes non-matching strings to be removed from the popup. It causes the set of choices to be reduced as you types. By default, this option is enabled.

- If the **Filter unmatched items** check box is enabled, the **Match text anywhere in completion** check box becomes available to all the search to return matches in the middle of strings; not just from the beginning. For example, the string “PWM” will return matches for “PWM_1” and “MyPWM”. By default, this option is enabled when the **Filter unmatched items** check box is enabled.
- **Macro Filtering** – This option is used to specify that Macros (#define) can be omitted from the popup list with the following options:
 - Show all (can be slow)
 - Show all (except system defines) (default)
 - Show macro-functions only
 - Show none

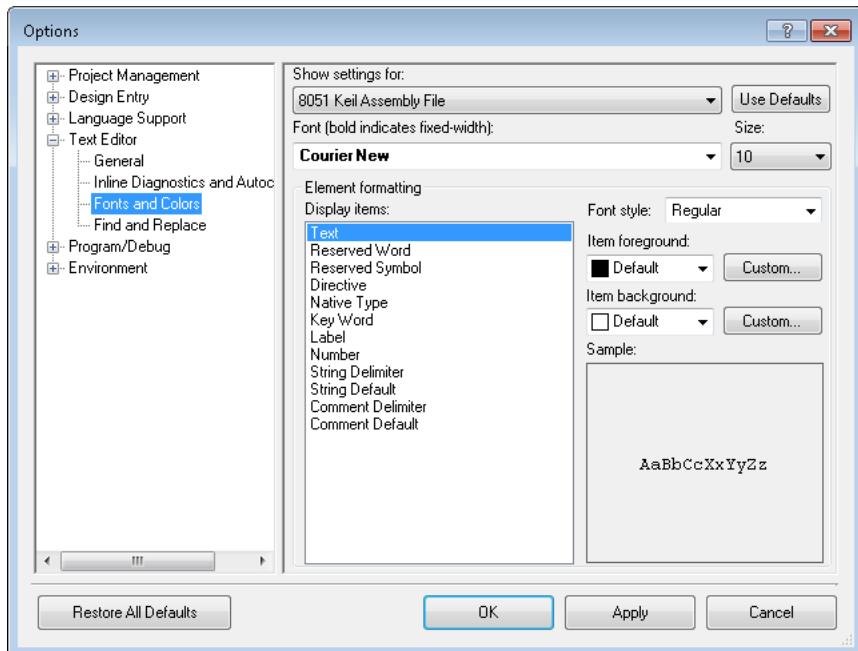
Note These options apply a filter to what macros are included in the auto-complete window for the All and Macros only displays. You can still toggle between All, Macros only, and Non-macros by pressing **[Ctrl] + [Space]**.

- **Tab behavior** – This option is used to allow how the **[Tab]** key is used while the popup is displayed, as follows:
 - Accept autocomplete (default)
 - Close autocomplete window
 - Perform UNIX-style Tab-completion (The **[Tab]** key adds as many characters as possible to the word being typed such that the result is still a match for the set of matches prior to the completion.)
- **Minimum characters required for popup** – This option is used to specify how many characters to type before the autocomplete popup displays. The valid range is 1-10. The default value is 1.

Note When the popup is already open this option is ignored.

Fonts and Colors:

This section provides options to change the various fonts and colors:



- **Show settings for** – Pull-down to select options for different types of files to use in the editor.
- **Use defaults** – This will restore the settings for the currently selected file type back to their original values provided by PSoC Creator.
- **Font** – Pull-down to select a font type for the selected **Display items**.
- **Size** – Pull-down to select a font size for the selected **Display items**.
- **Display items** – List to select the various items in the editor that can be changed.
- **Font style** – Pull-down to select different styles for the font: bold, italic, strikeout, and underline.
- **Item foreground/background** – Pull-downs to select colors for the selected **Display items** and background.
 - **Custom** – Buttons to open custom color chooser.

Find and Replace:

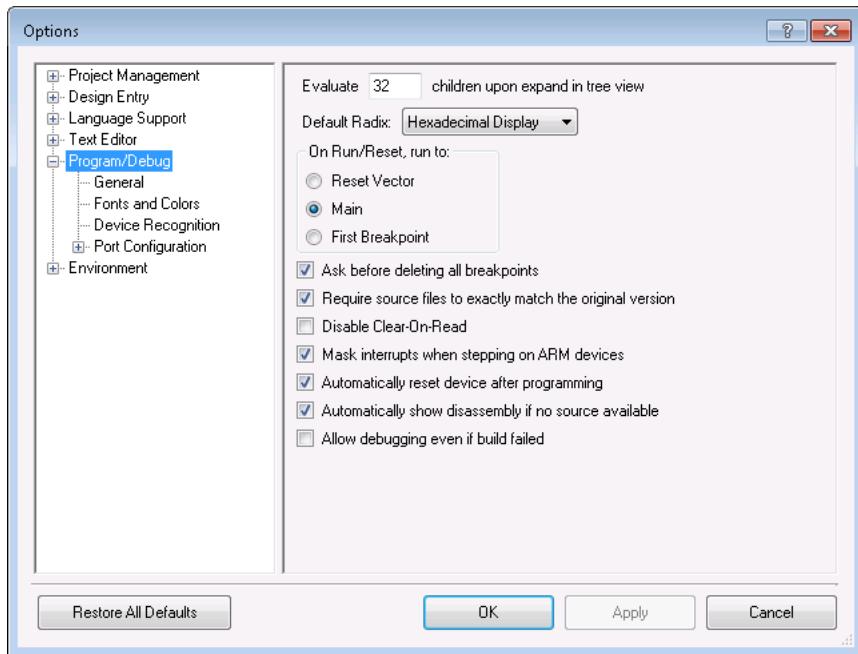
- **Display informational messages** – Selecting this check box will display informational messages related to Find & Replace in PSoC Creator.
- **Automatically populate Find What ...** – Selecting this check box populates the **Find What** field with the highlighted word(s) in the Text Editor.

See Also:

- [Options Dialog](#)
- [Code Editor](#)
- [Find Replace](#)

Program/Debug Options

The Program/Debug section of the Options dialog contains various options for configuring the device, debugger, MiniProg3, and kits.



General:

Under the General category, you can configure the following:

- Evaluate** – Text box to specify the number of children retrieved at a time in the variable view.
- Default Radix** – Pull down menu to specify the default Radix display in various windows.
- On Run/Reset, run to** – Radio option to select the "run to" point: Reset Vector, Main, or First Breakpoint.
- Ask before deleting all breakpoints** – Check box to indicate if PSoC Creator will ask you before deleting breakpoints.
- Require source files to exactly match the original version** – Check box to indicate if source files should match the original version. This option will allow you to edit code while debugging.

Note Changes will not take affect until after you recompile.

- Disable Clear-On-Read** – Allows the debugger to read the CLR registers without actually causing the data to be cleared. This applies to PSoC 3 and PSoC 5LP devices only.
- Mask interrupts when stepping on Arm devices** – This option masks interrupts from triggering while stepping through code in the debugger. This prevents the debugger from unexpectedly stepping into interrupt handler code. This applies to all devices except PSoC 3.
- Automatically reset device after programming** – After programming, automatically does a reset to cause new program to execute.
- Automatically show disassembly if no source** – Automatically opens the disassembly window if no source code is found for current debug line.

- **Allow debugging even if build failed** – If the build failed, but the output from a previous build still exists, provide a dialog box to allow starting the debugger with the old build.

Note When debugging an old build, the code being debugged may not match the source code in the editor.

Fonts and Colors:

Under the Fonts and Colors category, you can adjust the font type and size, as well as various colors and properties on the different debugger windows available in PSoC Creator.

- Choose the window or control in the **Show settings for** pull-down menu. Click **Use Defaults** to reset the settings to the default.
- Choose the **Font** type. (Bold font items indicate they are fixed-width fonts.)
- Choose the font **Size**.
- Under **Display items**, choose an item to change and either select a color from the **Item foreground** or **Item background** pull-down menus, or click the associated **Custom...** button to create a custom color.
- Choose the **Font Style** to change the text format to bold, italic, strikeout, and/or underline, if applicable.

Device Recognition:

Device Recognition is used to configure PSoC Creator to recognize 3rd party devices. This is done so that the [Select Debug Target](#) dialog can list correct information about devices that are attached to a computer.

Note that while this configuration allows PSoC Creator to recognize 3rd party devices, these devices cannot be selected for debugging. One use of Device Configuration is to configure the size of the Instruction Register and Data Register for 3rd party devices attached in a JTAG chain.

Note that Device Configuration can be accessed in two ways: from the Options dialog and from the [Select Debug Target](#) dialog by right-clicking on a node.

Port Configuration:

This section is used to configure the appropriate port.

MiniProg3:

- **Active Protocol** – Selects the protocol used to communicate with the target device.
- **Clock Speed** – Selects the frequency at which the MiniProg3 attempts to communicate with the target device. The selected speed should be no more than 1/3 of the Bus Clock speed of the target device. Additionally, while slower speeds are possible for debugging, the speed should be at least 1 MHz to program the device. 6 MHz is the highest recommended clock speed for PSoC devices.
- **Power** – Specifies the amount of voltage that the MiniProg3 will provide to the target device or whether power is from an external source.
- **Acquire Mode** – Selects the mechanism used to reset the device so that debugging is possible.
- **Connector** – Selects which connector on the MiniProg3 to use for sending data.

FX2LP-SWD (First Touch Kit 3 / First Touch Kit 5 / DVK3 / DVK5):

- **Active Protocol** – Displays the protocol used to communicate with the target device.
- **Power** – Specifies the amount of voltage that the MiniProg3 will provide to the target device or whether power is from an external source.

- **Acquire Mode** – Selects the mechanism used to reset the device so that debugging is possible.

TrueTouch Bridge:

- **Active Protocol** – Displays the protocol used to communicate with the target device.
- **Power** – Specifies the whether the voltage is from an internal or external source.
- **Acquire Mode** – Selects the mechanism used to reset the device so that debugging is possible.

DVKProg1:

- **Power** – Specifies the whether the voltage is from an internal or external source.
- **Acquire Mode** – Selects the mechanism used to reset the device so that debugging is possible.

KitProg:

- **Power** – Shows the options available on the board for selecting power. Not configurable.
- **Acquire Mode** – Shows the acquire mode supported by the board. Not configurable.

KitProg2:

- **Power** – Specifies the amount of voltage is supplied to the target device or whether power is provided from another source.
- **Acquire Mode** – Shows the acquire mode supported by the board. Not configurable.

CMSIS-DAP:

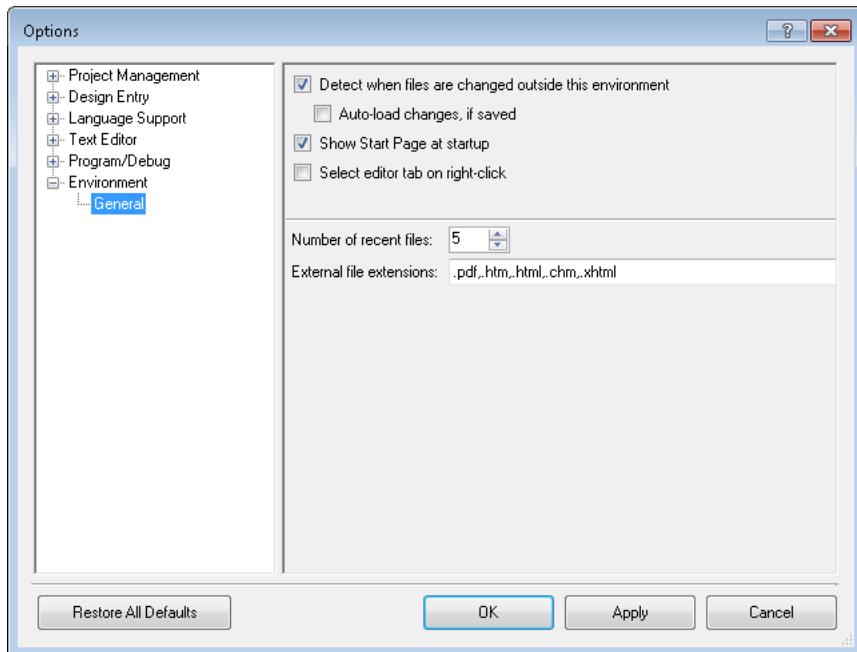
- **Power** – Shows the options available on the board for selecting power. Not configurable.
- **Acquire Mode** – Shows the acquire mode supported by the board. Not configurable.
- **Max Clock Speed** – Shows the speed at which communication happens. Not configurable.

See Also:

- [Options Dialog](#)
- [Using the Debugger](#)
- [Select Debug Target](#)
- [Device Configuration](#)
- [Debugger Windows](#)

Environment Options

The Environment section of the Options dialog contains several settings to specify how things display in PSoC Creator.



General Environment Options:

The following environment options are available under the General (default) section:

Detect when files are changed outside this environment

This check box specifies whether PSoC Creator will detect changes made outside of PSoC Creator. If checked, you can also specify whether to **Auto-load changes, if saved**.

Show Start Page at Startup

This check box allows you show or hide the Start page at startup. If you do not show the Start page at startup, you can open it from the [View menu](#).

Select Editor Tab on Right-Click

If this check box is selected, the mouse right-click action on a [document window tab](#) selects that document for display in addition to displaying a menu. If this check box is not selected, the right-click action only displays a menu.

Number of Recent Files

This text box allows you to specify the number of files shown under the Recent Files and Recent Projects items under the [File menu](#), as well as projects shown on the [Start page](#).

External File Extensions

Use this field to enter file extensions which will open with other applications instead of PSoC Creator. Separate the extensions using commas.

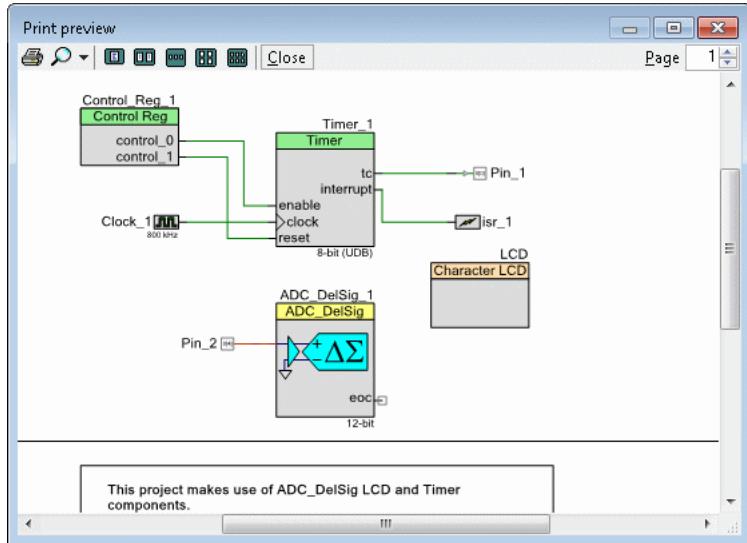
See Also:

- [Options Dialog](#)

- [File Menu](#)
- [View Menu](#)
- [Component Update](#)
- [Window Types](#)

Print Preview

The Print Preview dialog allows you to preview a document before printing it.



The types of PSoC Creator documents you can preview include: schematics, symbols, source code files, and some design-wide resources files.

To Open the Print Preview Dialog:

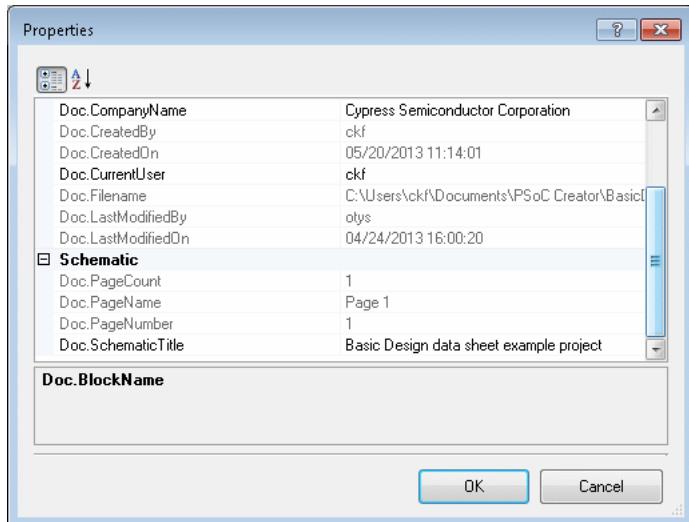
Click the **File** menu and select **Print Preview**.

To Use the Print Preview Dialog:

- Click **Print** to print the document.
- Use the Zoom pull-down menu to view the preview at different sizes.
- For files with multiple pages, use the various page view options. Use the **Page** box to jump to a specific one page view.
- Click **Close** to close the Print Preview dialog.

Properties

The Properties dialog provides information about selected items in other windows, such as the Workspace Explorer, Schematic Editor, Symbol Editor, and so on.



In many cases, you can change different values - or properties - of the selected items. In addition to project-level properties, each file in a project has properties. These properties allow you to set various options at a file level.

Depending on how you opened this dialog, it may contain different categories of properties, such as:

- **General** – file type, path, name
- **Document** – create date, current user
- **Schematic** – page count, title
- **Symbol** – catalog placement, summary
- **Build filters** – Configurations, Cores, Processors, Toolchains

To Open the Dialog:

Right-click on an item in PSoC Creator (for example, design canvas or file), and select **Properties**.

See Also:

- [Working with Text](#)
- [Defining Catalog Placement](#)
- [Assigning a Core in a Multi-Core Design](#)

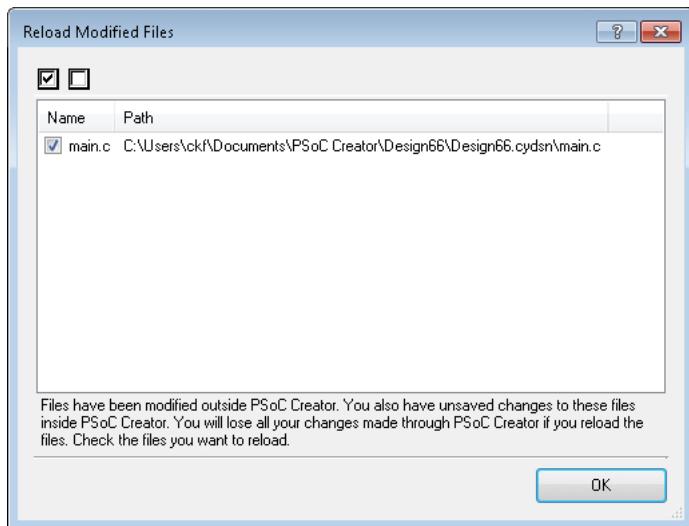
Reloading Files

If you work on PSoC Creator files outside the application, PSoC Creator -- when it regains focus -- will check if any files that are open have been changed on disk. If any files have changed, PSoC Creator will display two possible prompts, asking which of the files should be reloaded. If both types of documents exist, both prompts will be displayed.

To stop PSoC Creator from attempting to reload files that have been modified outside of the application you can uncheck the **Detect when files are changed outside this environment** option in [Environment Options](#).

Reloading Modified Files:

If the files requiring reloading have unsaved changes made from within PSoC Creator, the Reload Modified Files dialog will be displayed.

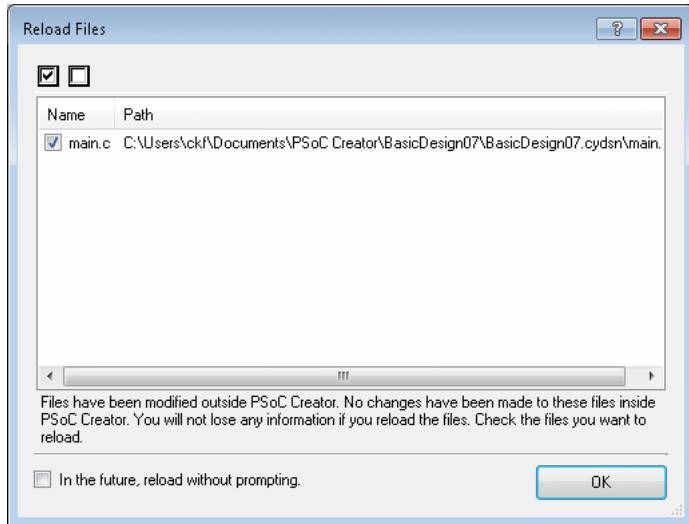


Any modified files that are reloaded will cause all unsaved changes to be lost.

- **Check All** – This option checks all the files specifying that they should all be reloaded.
- **Uncheck All** – This option unchecks all the files specifying that none of the files should be reloaded.

Reloading Unmodified Files:

If the files requiring reloading have no unsaved changes made from within PSoC Creator, the Reload Files dialog will be displayed.



From this dialog, the option to automatically reload unmodified files without prompting first can be set. If set to true this dialog will no longer be displayed.

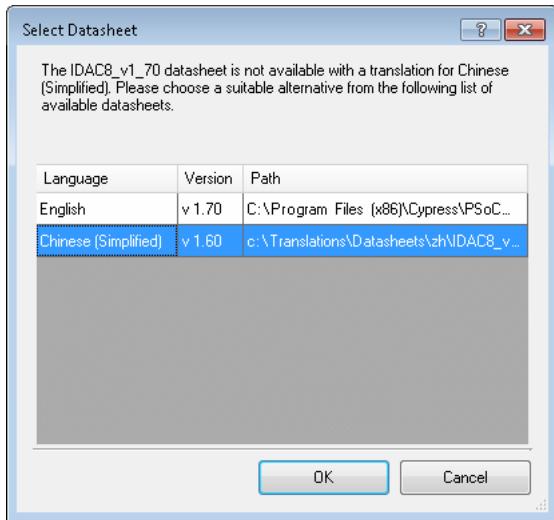
Note The option can also be changed from the Environment Option: **Auto load changes, if saved**.

See Also:

- [Modified Files](#)
- [Environment Options](#)

Select Datasheet

The Select Datasheet dialog is used to select an appropriate datasheet to open. This dialog displays when you try to open a datasheet, but the specific language and version of that datasheet is not available.



To Use the Dialog:

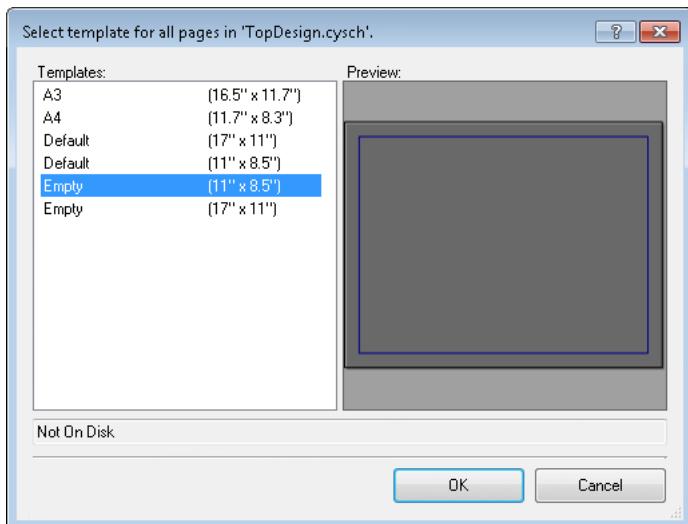
Select the appropriate datasheet to open and click **OK**.

See Also:

- [Language Support Options](#)

Select Sheet Template

The Select Sheet Template dialog is used to select the sheet template to use for your design.



To Open the Dialog:

For an existing project, right click on your design canvas and select **Change Template**.

Templates:

This area lists the available sheet templates to select for your design. Click on a template to select it.

Preview:

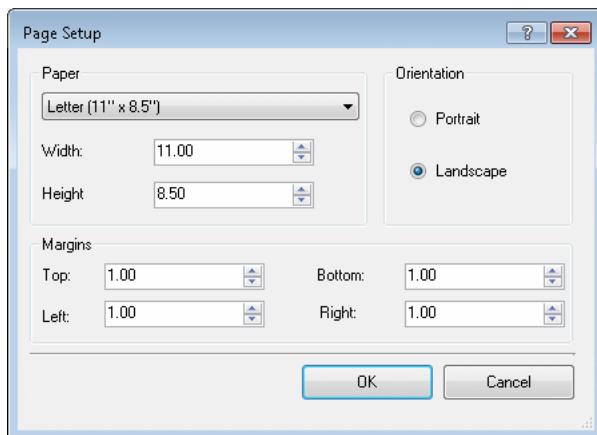
This area shows a preview of the selected template.

See Also:

- [Schematic Editor](#)
- [Sheet Template Editor](#)

Sheet Template Page Setup

The Sheet Template Page Setup dialog is used to specify various attributes for your sheet template. This is different from the Windows Page Setup dialog, used for printing.



This dialog only displays when you create a new sheet template. See [Sheet Template Editor](#) for more information.

Paper:

Choose a paper size from the pull down menu, or specify custom **Width** and **Height** measurements in inches.

Orientation:

Specify **Portrait** or **Landscape**.

Margins:

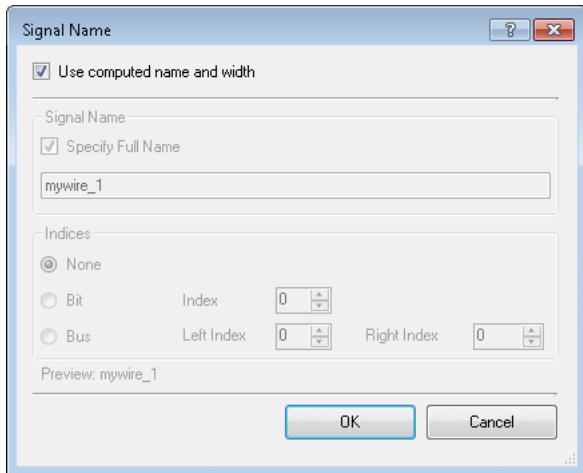
Specify the **Top**, **Bottom**, **Left**, and **Right** margins in inches.

See Also:

- [Sheet Template Editor](#)

Signal Name

The Signal Name dialog is used to name a signal.



A signal can have the following valid names:

- mywire_1 – base name only
- mywire_1[0] – base name with a bit index
- mywire_1[1:0] – base name with bus indices
- [0] – bit index only
- [1:0] – bus indices only

To Open the Dialog:

Open this dialog in two ways:

- For a wire with no label, double-click the wire, or right-click and select **Edit Name and Width**.
- For a wire with an existing label, double-click the wire label.

Use Computed Name and Width:

Select this check box to have PSoC Creator automatically specify the name and width; unselect to do it manually.

Specify Full Name:

1. To specify a name, you must first unselect the **Use Computed Name and Width** check box.
2. Then, select the **Specify Full Name** check box to enable the associated text box.
3. Use this text box to enter a base name for the signal.

Indices:

Use this section to select one of the following index options. This section becomes available when you unselect the **Use Computed Name and Width** check box.

- **None** – Select this option for no indices on the base name (e.g., my_wire). If you unselect the **Specify Full Name** check box, then the **None** option is disabled.

- **Bit** – Select this option to set a bit index on the signal (e.g., [0]).
- **Bus** – Select this option to set bus indices on the signal (e.g., [1:0]).

Note The maximum bus width allowed is 1024.

Preview:

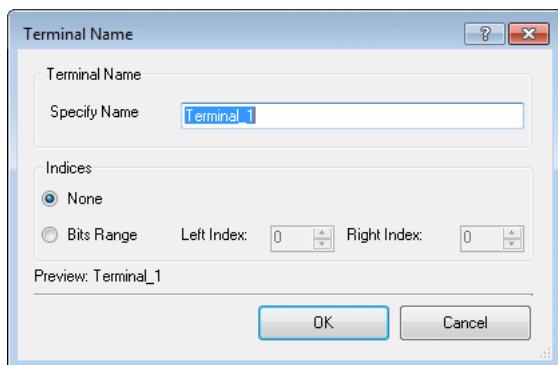
This field shows you how the signal name will appear on the schematic.

See Also:

- [Wire Labels and Names](#)
- [Working with Wires](#)
- [Drawing Buses](#)

Terminal Name

The Terminal Name dialog is used to name a terminal.



A terminal can have the following valid names:

- Base name only (Terminal_1) – A terminal must have a base name that begins with a letter.
- Base name with bus indices (Terminal_1[1:0]) – You may specify left and right indices, if desired.

To Open the Dialog:

Open this dialog in two ways:

- Drag a new terminal onto your design, or
- Double-click the label or terminal.

Specify Name:

The **Specify Name** field is used to enter a base name for the signal.

Indices:

Use this section to select one of the following index options:

- **None** – Select this option for no indices on the base name (e.g., Terminal_1).
- **Bits Range** – Select this option to set bus indices on the signal (e.g., [1:0]).

Note The maximum bus width allowed is 1024.

Preview:

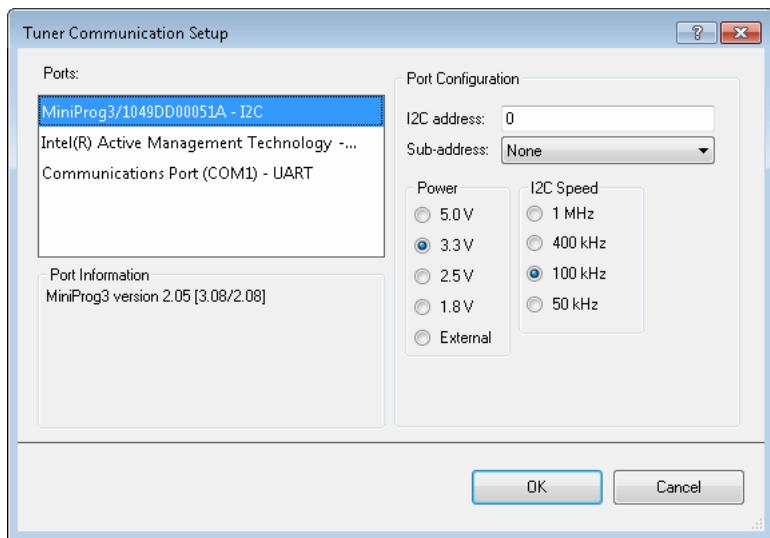
This field shows you how the signal name will appear on the schematic.

See Also:

- [Signal Name](#)
- [Working with Schematic Terminals](#)
- [Working with Component Terminals](#)

Tuner Communication Setup

The Tuner Communication Setup dialog allows you to configure the settings used for reading parameters back from the PSoC device. It allows for selecting the communication port and selecting properties.



Currently, tuners only support communication via I²C. This can be done using the EZ I²C Component or the standard I²C Component.

To Open the Dialog:

Open this dialog from the appropriate tuner application.

Currently, the only tuner application is provided with the CapSense® Components. Refer to the Component's datasheet, available from the [Component Catalog](#).

Interface Description:*Ports*

This is a list of all of the ports attached to the computer that can be used to communicate with the tunable Component. Based on which item is selected, different options will be available for the Port Configuration.

Port Configuration

This section allows for configuring the interface specific options for communicating with the Component. This is necessary to ensure both Component and tuner are configured the same.

For I²C communication, there are four pieces of required information:

- The voltage the port needs to supply to the device, if any.
- The speed at which the data can be clocked into the target device.
- The Address of the slave device that is being communicated with.
- The size of sub-addresses used for indicating what block of data to read.

Port Information

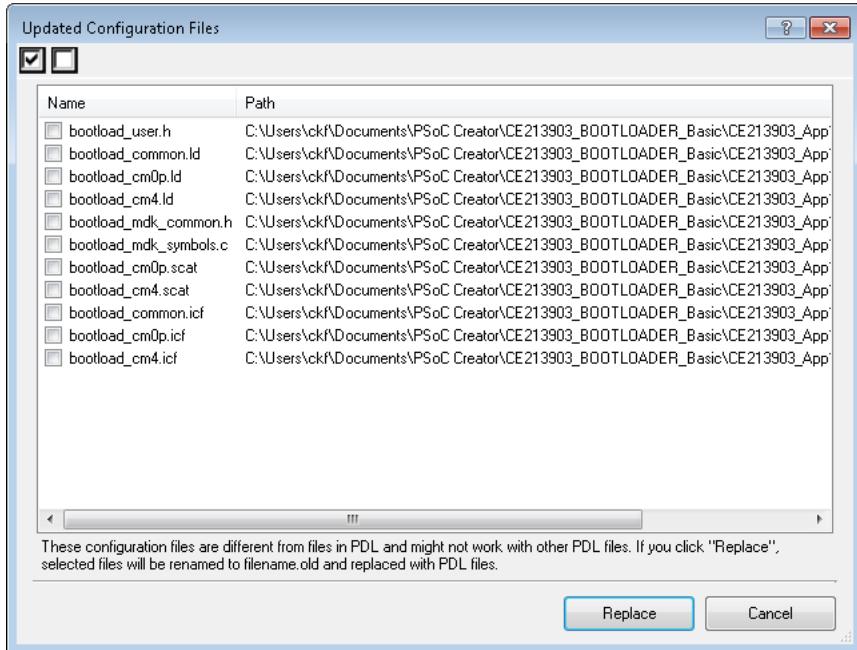
This section displays information about the currently selected port.

See Also:

- CapSense® Component datasheets (available from the [Component Catalog](#))

Updated Configuration Files

If you updated the version of the Peripheral Driver Library (PDL) installed on your computer, the Updated Configuration Files dialog might display during a build of an existing project. This dialog lists files in your project that might not be compatible with the newer PDL version. Use this dialog to replace the listed files, if needed.



If you are sure that the listed files are correct, then click **Cancel** to proceed with the build with these files.

If you want files from the newer PDL version, then click **Replace** to copy new files from the PDL. The existing files will be renamed to <filename>.old.

See Also:

- [Peripheral Driver Library](#)

Customizing the Framework

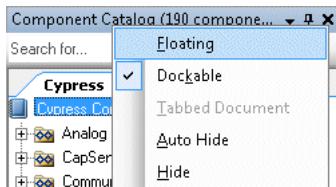
This section contains several "how to" topics that allow you to customize the PSoC Creator framework:

- [To Float Tool Windows](#)
- [To Dock Tool Windows](#)
- [To Use Tabbed Documents](#)
- [To Move Tool Windows](#)
- [To Auto-Hide Tool Windows](#)

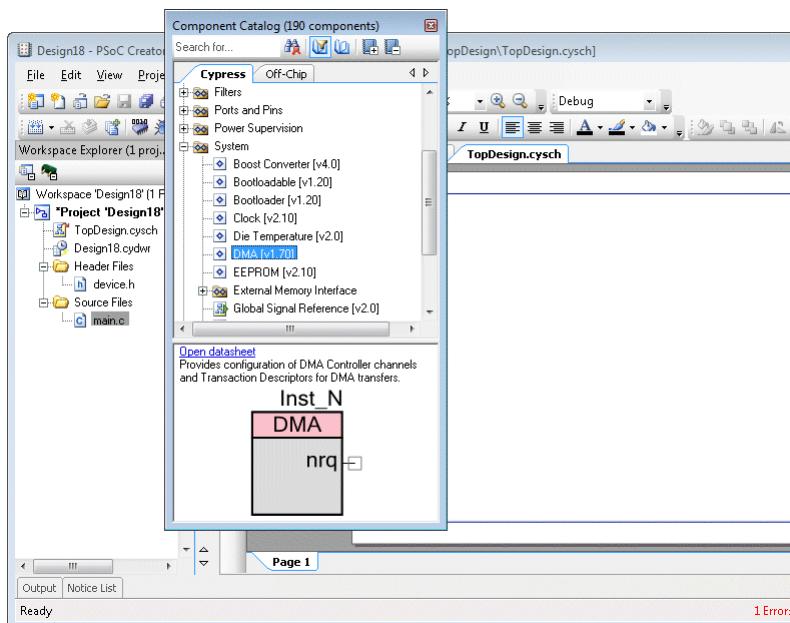
To Float Tool Windows

You can float various PSoC Creator windows outside the framework. Floating means a window is not attached to the framework.

1. Select the window you wish to float and select **Floating** from the Window mode menu.



Notice the window moves outside the PSoC Creator framework.



2. Move the window from its current location to anywhere on your screen.

See Also:

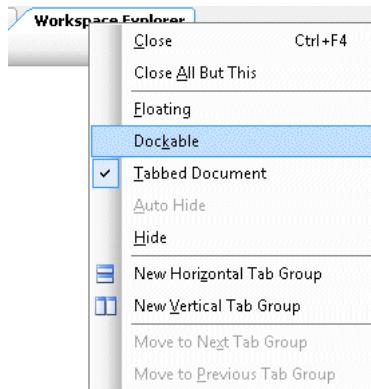
- [Tool Windows](#)

- [Framework Description](#)
- [To Move Tool Windows](#)
- [To Dock Tool Windows](#)

To Dock Tool Windows

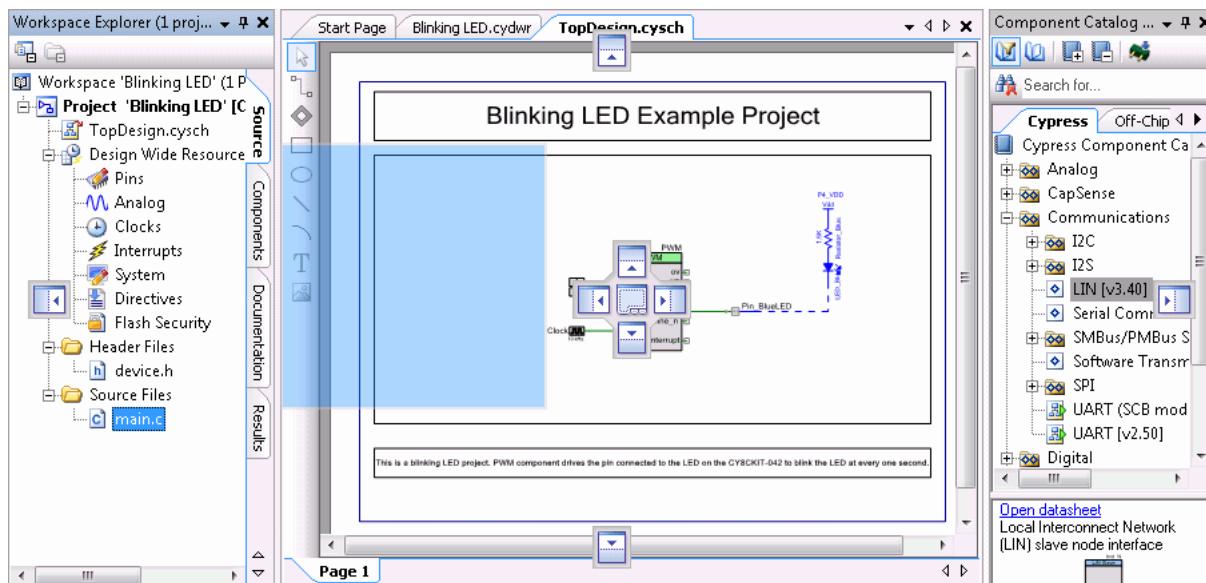
You can dock various windows in different locations within the PSoC Creator framework. Docking means attaching a window to the framework.

1. Select the window you wish to move and select **Dockable** from the Window mode menu.



2. Drag the window from its current location toward another location within the framework.

Notice as you drag the window that docking guides appear at different locations.

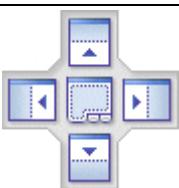


3. When the window you are dragging reaches the position where you want to dock it, place the mouse over the corresponding portion of the guide.

An outline of the window appears in the designated area.

4. To dock the window in the position indicated, release the mouse.

Tool windows can be docked to the framework edge or other existing edges, as well as within other windows as tabs. The following table shows the different docking guides and their meanings:

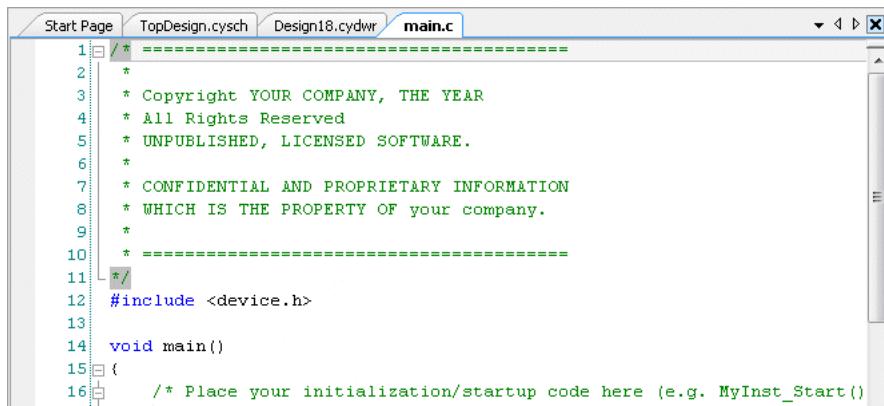
	Dock to the left edge.
	Dock to the right edge.
	Dock to the top edge.
	Dock to the bottom edge.
	Dock to the left, right, top, or bottom edge or dock within the window as a tab.

See Also:

- [Tool Windows](#)
- [Framework Description](#)
- [To Move Tool Windows](#)
- [To Float Tool Windows](#)

To Use Tabbed Documents

As described under [Window Types](#), certain windows can be docked to the framework or used as tabbed documents. Tabbed documents appear side-by-side in an area of the tool:



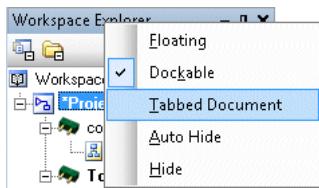
The screenshot shows the PSoC Creator interface with a tabbed document window. The tabs at the top are "Start Page", "TopDesign.cysch", "Design18.cydwr", and "main.c". The "main.c" tab is currently active, displaying C code:

```

1  /* ===== */
2  *
3  * Copyright YOUR COMPANY, THE YEAR
4  * All Rights Reserved
5  * UNPUBLISHED, LICENSED SOFTWARE.
6  *
7  * CONFIDENTIAL AND PROPRIETARY INFORMATION
8  * WHICH IS THE PROPERTY OF your company.
9  *
10 * =====
11 */
12 #include <device.h>
13
14 void main()
15 {
16     /* Place your initialization/startup code here (e.g. MyInst_Start())

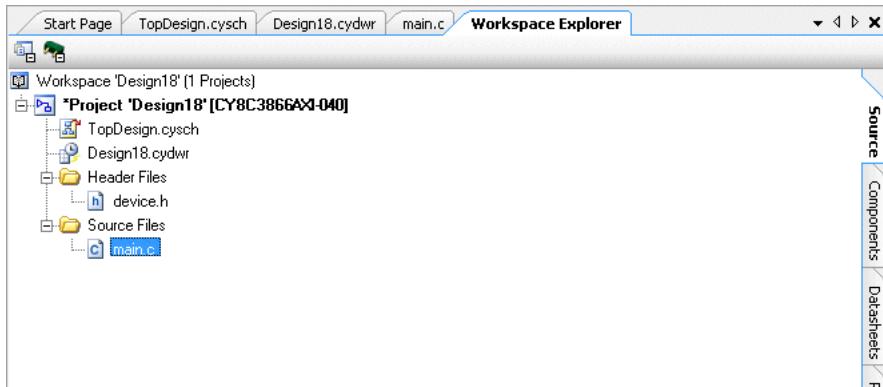
```

To change a window to a tabbed document (for example the [Workspace Explorer](#)), right-click on the window header and select **Tabbed Document**.



Note If a window cannot be changed to a tabbed document (such as the [Component Catalog](#)), the menu item will be disabled (grayed out).

After the menu item is selected, the window will be shown as a tabbed document along with other open documents.



See Also:

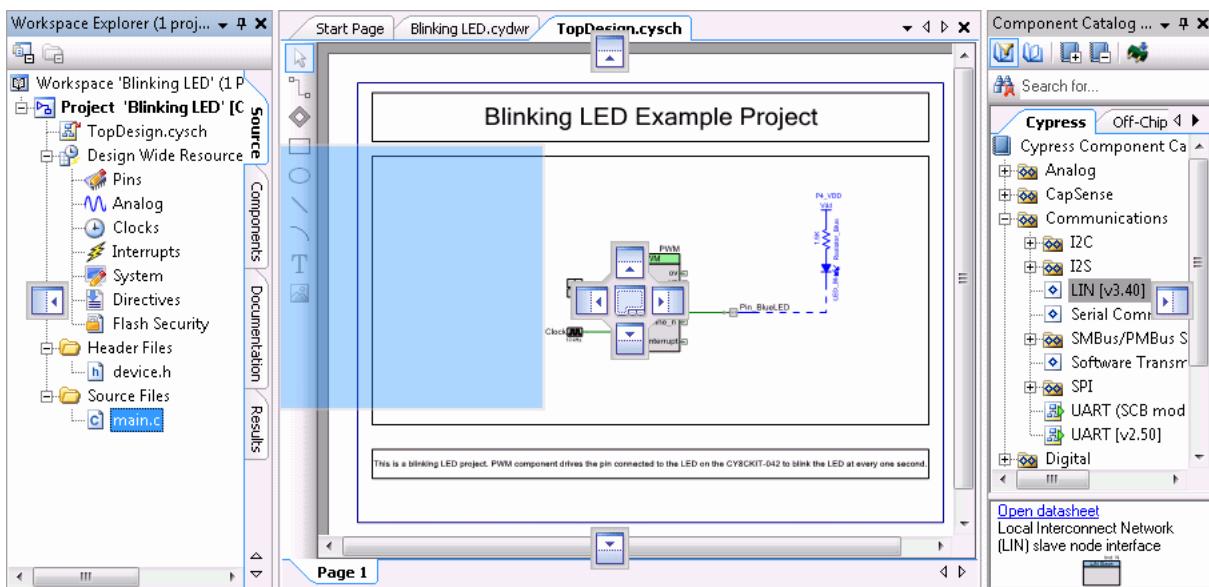
- [Window Types](#)
- [Tool Windows](#)
- [Document Windows](#)

To Move Tool Windows

You can move various windows to different locations within the PSoC Creator framework, or even outside the framework altogether.

1. Click the title bar of the window you wish to move.
2. Drag the window from its current location toward another location.

Notice as you drag the window that docking guides appear at different locations.



- Move the window to the desired location and release the mouse.

See Also:

- [To Dock Tool Windows](#)
- [To Float Tool Windows](#)
- [Framework Description](#)
- [Tool Windows](#)

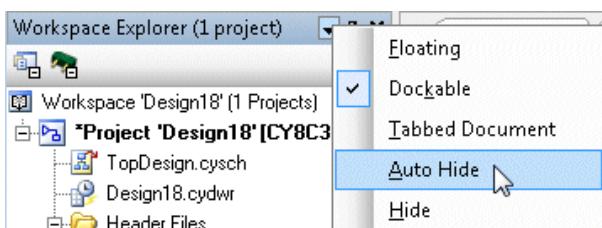
To Auto-Hide Tool Windows:

The auto-hide feature allows you to see more of the PSoC Creator framework by minimizing tool windows along the edges of the framework when not in use.

To Turn On Auto-Hide:

Turn on auto-hide using one of these methods:

- Select **Auto-Hide** from the Tool Window toolbar or right-click context menu.



- Click the Auto-Hide Pushpin icon.

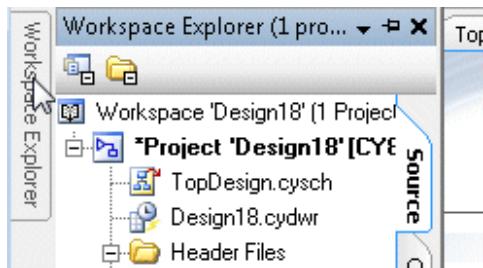


The window automatically slides to the edge of the framework. Its name is visible on a tab at the edge of the framework.



To Show the Hidden Window:

Place your cursor on the tab to slide the tool window back into place.



To Turn Off Auto Hide:

Turn off auto-hide using one of these methods:

- Select **Dockable** from Tool Window toolbar or right-click context menu, or
- Click to the **Auto-Hide Pushpin** again.

See Also:

- [Framework Description](#)
- [Tool Windows](#)
- [To Move Tool Windows](#)
- [To Dock Tool Windows](#)
- [To Float Tool Windows](#)

4 Using Design Entry Tools



The PSoC Creator design entry tools allow you to create a design using abstract symbols and focus on the system rather than the low-level device details.

This section is divided into the following main categories:

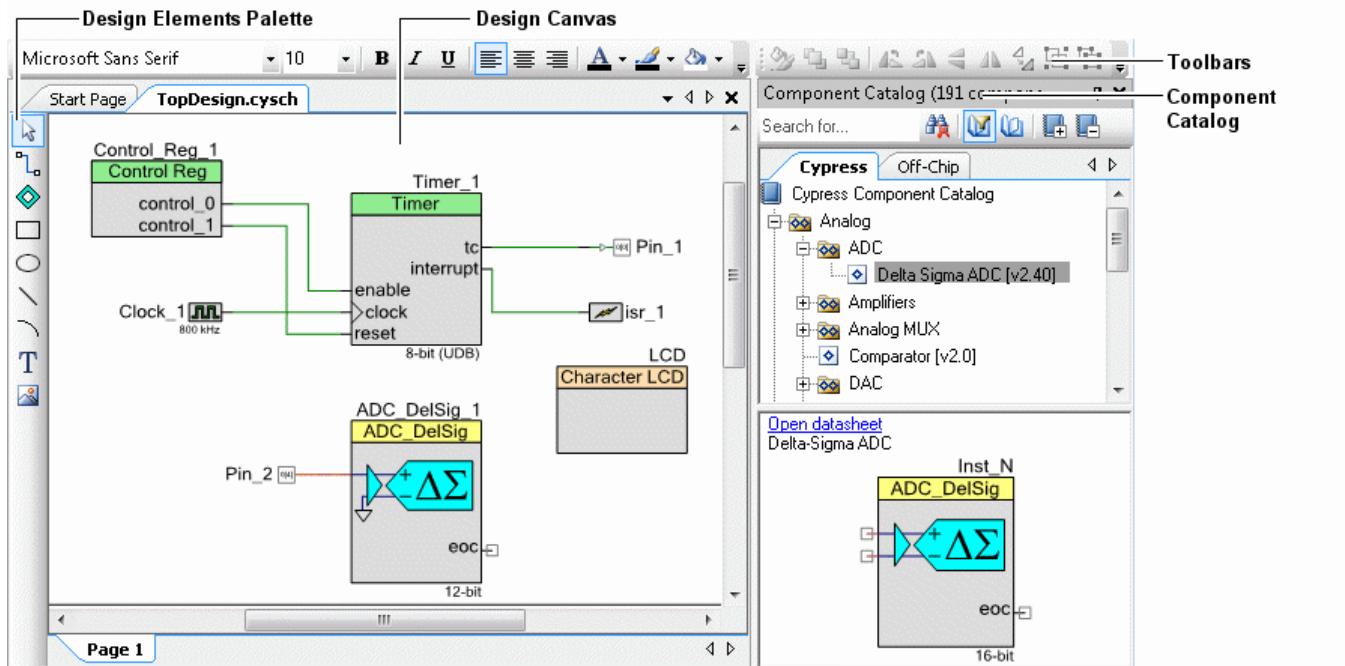
- [Schematic Editor](#) – primary tool to create designs
- [Code Editor](#) – tool to edit source code
- [Design-Wide Resources](#) – tools to configure settings for an entire design
- [Symbol Editor](#) – tool used to create Components
- [UDB Editor](#) – tool used to configure a UDB implementation
- Other Tools
 - [Schematic Macro Editor](#)
 - [Sheet Template Editor](#)
 - [Format Shape](#)

Additionally, there are several topics that pertain to both editors:

- [Common Toolbars](#)
- [Design Elements Palette](#)
- [Working with Text](#)
- [Working with Lines](#)
- [Working with Shapes](#)
- [Design Entry Reserved Words](#)

Schematic Editor

The Schematic Editor allows you to create and edit schematics for your designs and implementations for your symbols.



The main Components of the Schematic Editor include:

- Design Canvas – the canvas on which you draw designs
- [Design Elements Palette](#)
- [Common Design Entry toolbars](#) – commands common to the design entry tools
- [Context Menus](#) – commands available by right-clicking
- [Component Catalog](#) – library of Components to use in your schematic

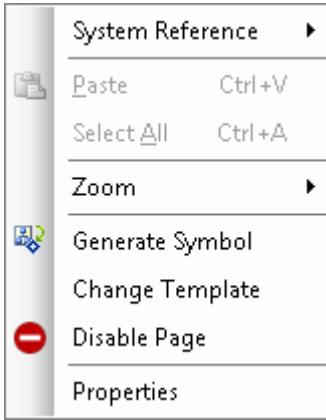
This section contains various topics related to working with the Schematic Editor:

- [Creating a New Schematic](#)
- [Configuring Component Parameters](#)
- [Working with Wires](#)
- [Rubber-Banding](#)
- [Drawing Buses](#)
- [Wire Labels and Names](#)
- [Using Multiple Pages and Connectors](#)
- [Disabling/Enabling Schematic Pages](#)
- [Working with Schematic Terminals](#)

Schematic Editor Context Menu Commands

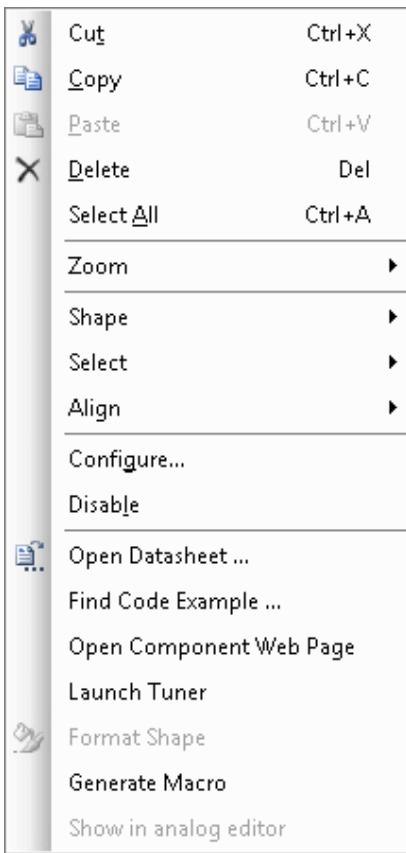
The Schematic Editor contains various commands available on right-click – or context – menus. The commands available will vary depending on whether you right-click on the canvas itself or on a Component/element. The following are the commands available:

On Canvas:



- **System Reference** – Provides access to the most current System Reference Guide on disk, as well as a link to a web page for other versions and translations of the document, if available.
- **Paste** – Same as command from the [Standard Toolbar](#).
- **Select All** – Selects everything on the canvas.
- **Zoom** – Same as zoom commands from the [View Menu](#).
- **Generate Symbol** – Creates a new symbol with an automatically generated default shape; it will automatically include terminals representing the schematic terminals of the source schematic.
- **Change Template** – Opens the [Select Sheet Template](#) dialog to select/change a sheet template to use for your design.
- **Disable Page** – Disables the selected page. See [Disabling/Enabling Schematic Pages](#).
- **Properties** – Opens the [Properties dialog](#).

On Selected Object(s):



- **Cut, Copy, Paste, Delete** – Same as commands from the [Standard Toolbar](#).
- **Select All** – Selects everything on the canvas.
- **Zoom** – Same as zoom commands from the [View Menu](#).
- **Shape** – Same as shape commands from the [Common Design Entry Toolbars](#).
- **Select** – Allows you to select a specific object when two or more objects are drawn on top of each other.
- **Align** – When two or more objects are selected, this command allows you to align selected shapes: left, right center, top, middle, and bottom.
- **Configure** – Opens the [Configure Component Parameters](#) dialog to edit parameters for the Component instance.
- **Enable/Disable** – Enables or disables the selected Component. Disabling a Component means that PSoC Creator will ignore it; this sets the CY_REMOVE flag to true on the Configure dialog under the [Built-In tab](#).
- **Open Datasheet** – Opens the datasheet for the selected Component.
- **Find Code Example** – Opens the [Find Code Example](#) dialog for the selected Component.
- **Open Component Web Page** – If available, opens a web page for the Component, where you can access datasheets in different languages.
- **Launch Tuner** – If the selected Component has a tuner application, this command launches the tuner.
- **Format Shape** – Opens the [Format Shape dialog](#) to change various characteristics for the selected shape(s).
- **Generate Macro** – Creates a [Schematic Macro](#) from the selected elements.
- **Show in analog editor** – Shows the selected Component in the Design-Wide Resources [Analog Device Editor](#).

See Also:

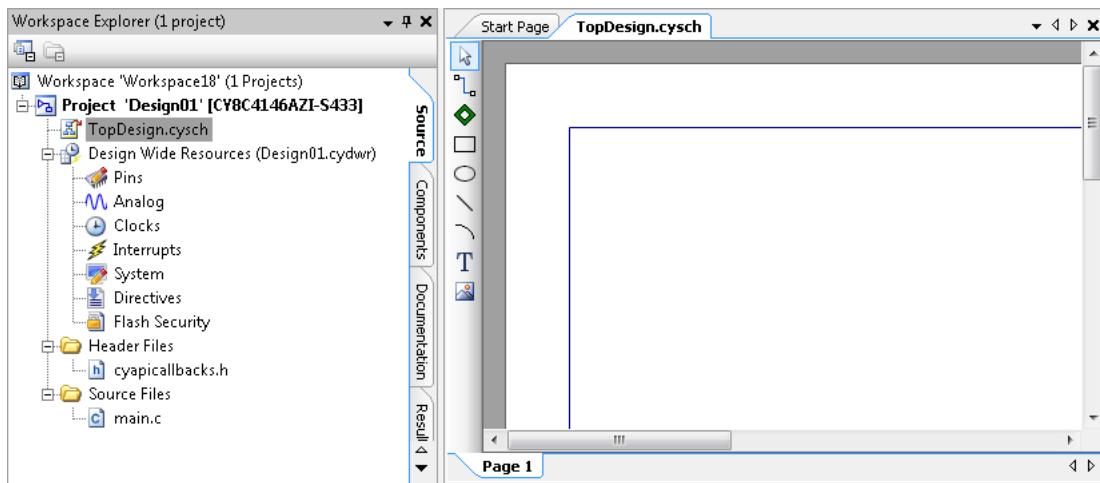
- [Design Elements Palette](#)
- [Standard Toolbar](#)
- [Common Design Entry Toolbars](#)

Creating a New Schematic

The process of creating a new schematic varies depending on what you are trying to accomplish with the schematic. Within a project, a schematic can be either the top-level design schematic or a Component implementation. The process varies for each context.

Top-Level Design Schematic:

If you create a new design project, a top-level design schematic will be created for you along with the rest of the initial files in your design. This schematic is the top-level design document in which you express your design graphically using Components and connections. For more information about creating a design project, see [My First Design "Hello World"](#) and [Creating a New Project](#).

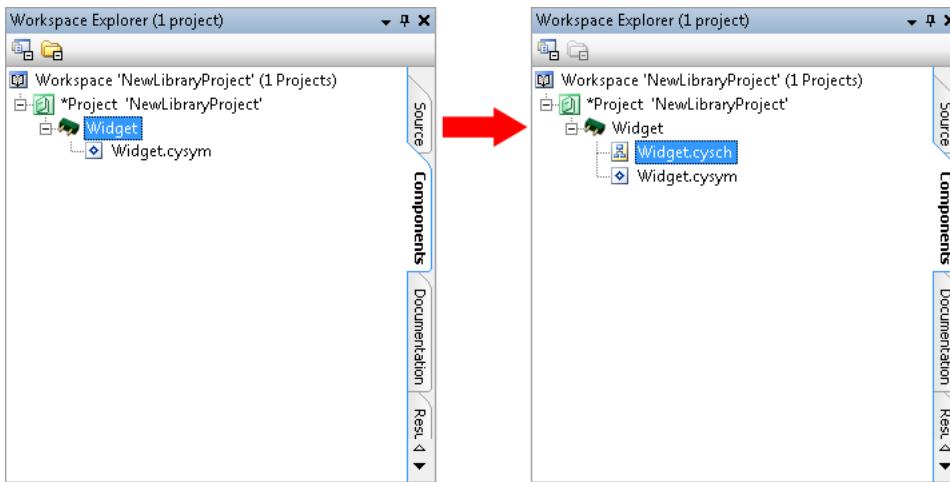


Component Implementation Schematic:

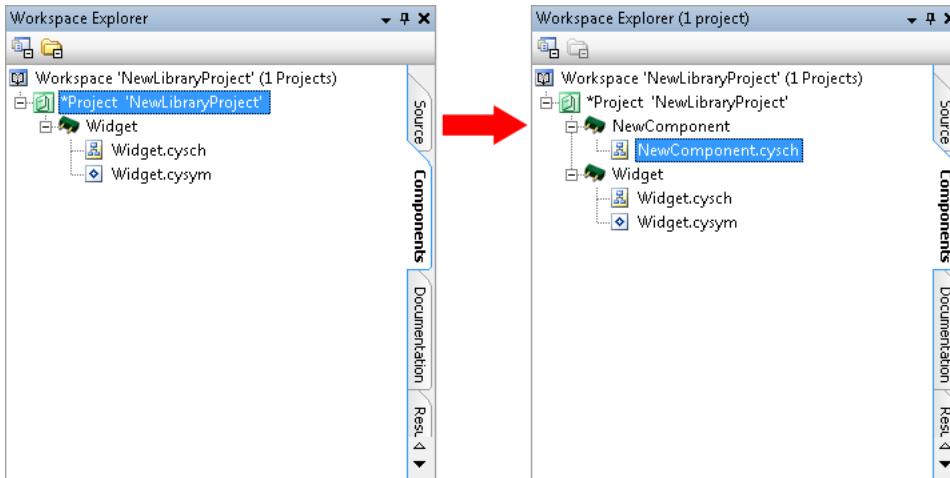
If you are working with a library project and Components, a schematic can be the implementation of a Component. For more information about creating Components, refer to the [Component Author Guide](#).

Within a Component, you must add a schematic implementation manually. You can add a schematic at the Component level or the project level. At either level, you are adding a Component item of the type schematic. For more information, see [Adding a Component Item](#).

- When you add a schematic at the **Component level**, you are adding a new schematic item to the existing Component:



- When you add a schematic at the **project level**, you create a new Component with a schematic as a Component item:



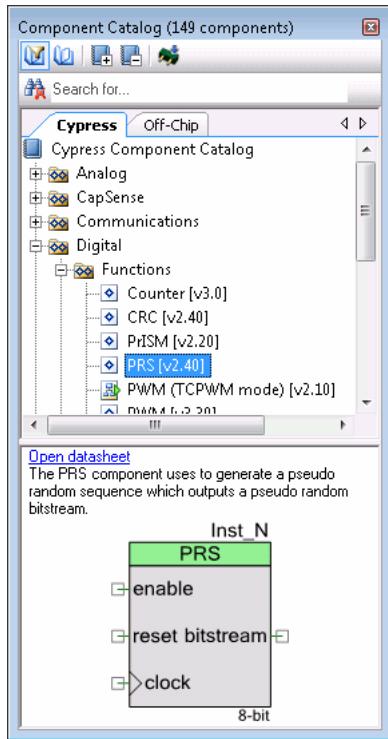
Note You may only have one top-level (generic device) schematic per Component. You may also only have one schematic for each architecture, family, and device level of your Component.

See Also:

- [Creating a New Project](#)
- [Component Author Guide](#)
- [Adding a Component Item](#)
- [Schematic Editor](#)
- [Workspace Explorer](#)

Component Catalog

The Component Catalog is a [Schematic Editor](#) tool window that contains one or more sets of Components to use in a design. The Components are organized into categories. Each category contains a tree with the Components. There can also be one or more tabs with different trees and categories.



Component Tabs

By default, there are two tabs of Components: **Cypress** and **Off-Chip**. Cypress Components are always installed. These are developed by Cypress for use in your designs. Off-Chip or "External" Components, if installed, are used to document connections the device may have on the development board. They are used for documentation purposes only. They are described in the External Library Component Datasheet.

Note The location and order of the Components in this catalog are determined by various properties for each Component. They do not necessarily reflect the actual library in which they are stored. Also, not every Component is available for every device.

Component Preview

Below the Components is an area with the following:

- **Open Datasheet** – Link to open the selected Component datasheet.
- **Description** – Short summary of the Component.
- **Component Preview** – Preview of the selected Component symbol.

Toolbars:



The Component Catalog contains a set of tools to work with the catalog, as follows:

- **Show Latest Versions** – Shows only the most recent versions of Components with numbers. If there is only one version, no numbers are shown.
- **Show All Versions** – Shows all versions of Components with Component numbers. If there is only one version, no numbers are shown.
- **Expand All** – Expands all nodes to show every visible Component.
- **Collapse All** – Collapses all nodes to show only category folders.
- **Find new Components** – Opens the [Component Installer](#) dialog to find and install new Components and Component versions.
- **Clear Results** – Clears the Search for filter.
- **Search for** – Filters the Components shown to those that match the text entered in this field.

Context Menu Commands:

If you right-click on a Component in the tree view, you can access the following commands:



- **Open Datasheet** – Select this option to open the Component's datasheet.
- **Find Code Example** – Select this option to open the [Find Code Example](#) dialog to open a code example specifically for the selected Component.
- **Open Component Web Page** – If available, this option opens a web page for the Component, where you can access datasheets in different languages.
- **Properties** – Select this option to see the basic properties of the Component. The properties include the library path of the Component.

To View the Component Datasheet:

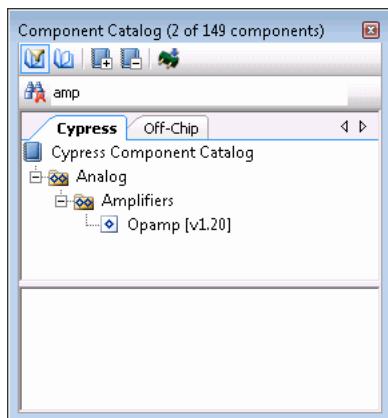
Click the **Open Datasheet** link, or select the option from the context menu. The datasheet for that Component will open.

To Add a Component to a Schematic:

Click on a Component in the catalog and drag it onto the [Schematic Editor](#) canvas.

To Search for a Component:

Type text in the search field. As you begin typing, the Components available will filter based on what you type.



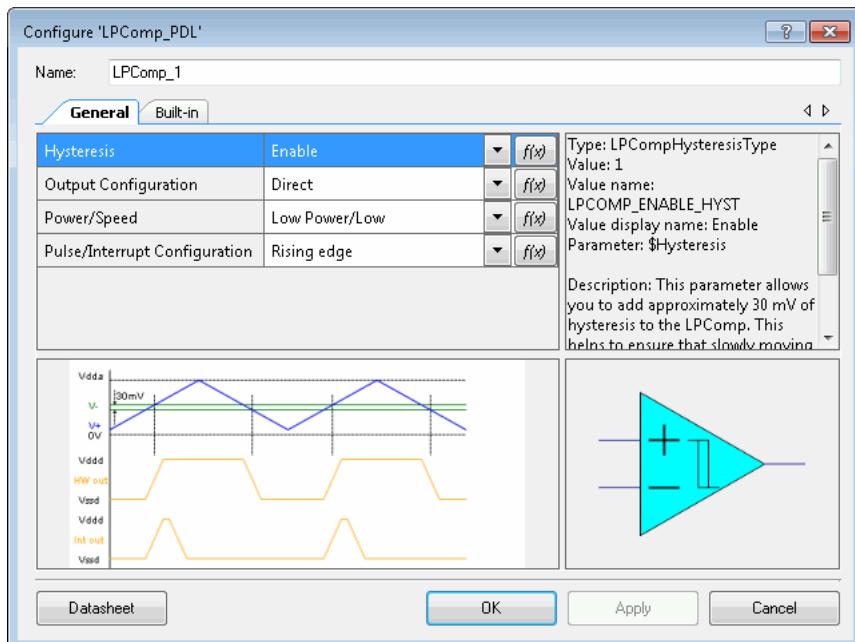
Click **Clear Results**  to clear the search results and restore the Component Catalog to view all Components.

See Also:

- [Schematic Editor](#)
- [Tool Windows](#)
- [Component/Instance](#)
- [Find Code Example](#)
- [Component Installer](#)

Configure Component Parameters

When you drag an instance of a Component onto your schematic, you can configure it with the Configure dialog.



Note This help topic is generic. Many Components have customized parameter configuration user interfaces. Refer to a Component's datasheet for specific information.

To Open this Dialog:

1. Double-click a Component instance or right-click on a Component instance and select **Configure**.
2. Edit the parameters as appropriate and click **OK**.

To Open the Component Datasheet:

Click the **Datasheet** button.

If a Component has a datasheet, it will open in a separate window.

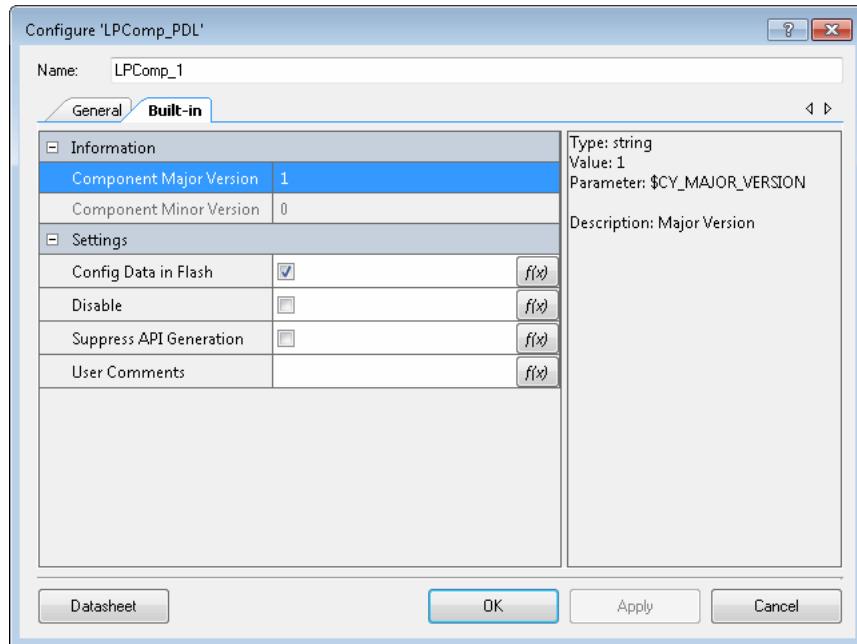
To Rename the Component Instance:

By default, the Component instance will receive the name "INSTANCE_1" where INSTANCE is the name of the Component, such as PGA, Counter, Timer, etc.

Type the instance name you prefer in the **Name** field.

Note Once you generate source code for a Component, case-only name changes will be ignored on subsequent code generation updates. For information about generating code, see [Building a PSoC Creator Project](#) and [Generated Files](#).

Built-In Parameters:



Every Component contains a tab with the following built-in parameters:

- Component Major Version - This parameter displays the major version number of the Component.
- Component Minor Version - This parameter displays the minor version number of the Component.

- Config Data in Flash - Controls whether the configuration structure is stored in flash (const, true) or SRAM (not const, false).
- Disable - This parameter is used to disable the Component and remove the instance from the generated netlist during a build. The default value is false.
- Suppress API generation - This parameter is used to prevent PSoC Creator from generating APIs for the specific instance. The default value is false.
- User Comments - User provided instance specific comments.

For more information about built-in parameters, refer to the [Component Author Guide](#).

See Also:

- [Component Catalog](#)
- [Schematic Editor](#)

Working with Wires

Using the [Schematic Editor](#), you can draw wires to connect Components and wires. This section covers a few of the different techniques to use when working with wires:

- [Drawing a Wire](#)
- [Connecting to a Terminal](#)
- [Drawing Multi-point Wires](#)
- [Connecting to another Wire](#)
- [Selecting a Wire/Net](#)

See also [Wire Labels and Names](#) and [Drawing Buses](#).

To Draw a Wire:

1. Click the **Draw Wire** tool. 
2. Click on the design canvas and drag the mouse to the desired location.
3. Click the left mouse once to continue the wire in a different direction.
4. Double-click to end the wire.

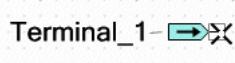
Note The Schematic Editor draws a digital (green) wire when it is connected to digital Components or not connected to any Components. To draw an analog (red) wire, you must connect it to analog Components.

To Connect to a Terminal:

1. To begin drawing a wire, first create an input and output Component on your schematic.
2. Click the **Draw Wire** tool. 

3. Move your pointer toward the contact point of one of the terminals.

Notice as you near the contact point that your pointer changes to a black X:



4. Click and release the mouse button, and move the pointer to begin drawing the wire.

Notice that the pointer changes to a + .

As you near the next terminal contact point, your pointer again changes to a black X:



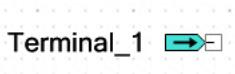
5. Click the left mouse button at the second terminal contact point to establish the connection.

The wire becomes selected with a dashed box surrounding it:



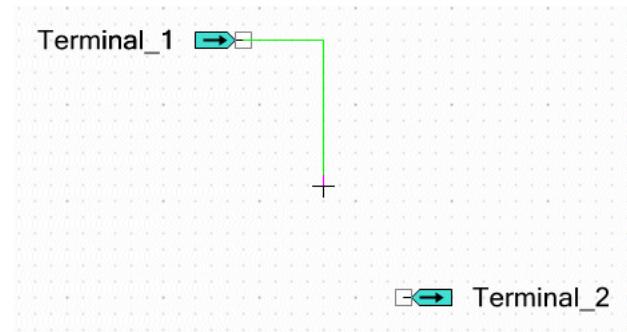
To Draw Multi-Point Wires:

1. Use the first example, and move the second terminal down on your schematic:



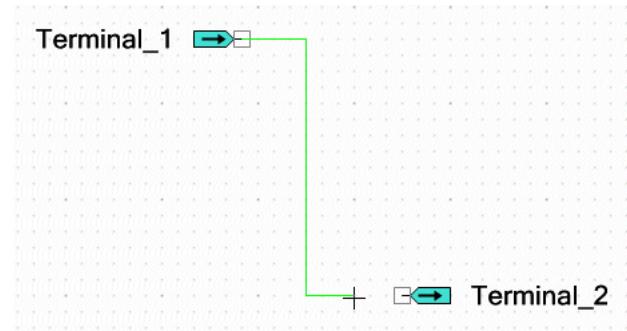

2. Click the **Draw Wire** tool. 
3. Click and release the mouse button, and move the pointer to begin drawing the wire.
4. Move the pointer down toward the second terminal.

Notice that wire changes shape to an inverted L.



- Move the pointer down to be even with the terminal, click and release the mouse button, and move the pointer right – toward the terminal.

Notice that wire obtains an additional point.



You can create as many points as you need by repeatedly clicking the mouse at different points on your schematic.

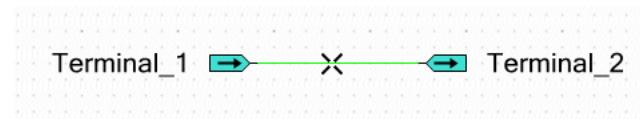
- Click the left mouse button at the second terminal contact point to establish the connection.

The wire becomes selected with a dashed box surrounding it.

To Connect to Another Wire:

- Use the first example again, and create an additional output terminal on your schematic.
- Click the **Draw Wire** tool. 
- Move your pointer toward an existing wire.

Notice as you near the contact point that your pointer changes to an X:



- Click and release the mouse button, and move the pointer to begin drawing the wire.

Notice that a dot appears on the wire to denote a connection point.



5. Click the left mouse button at the third terminal contact point to establish the connection.

The wire becomes selected with a dashed box surrounding it.

To Select a Wire/Net:

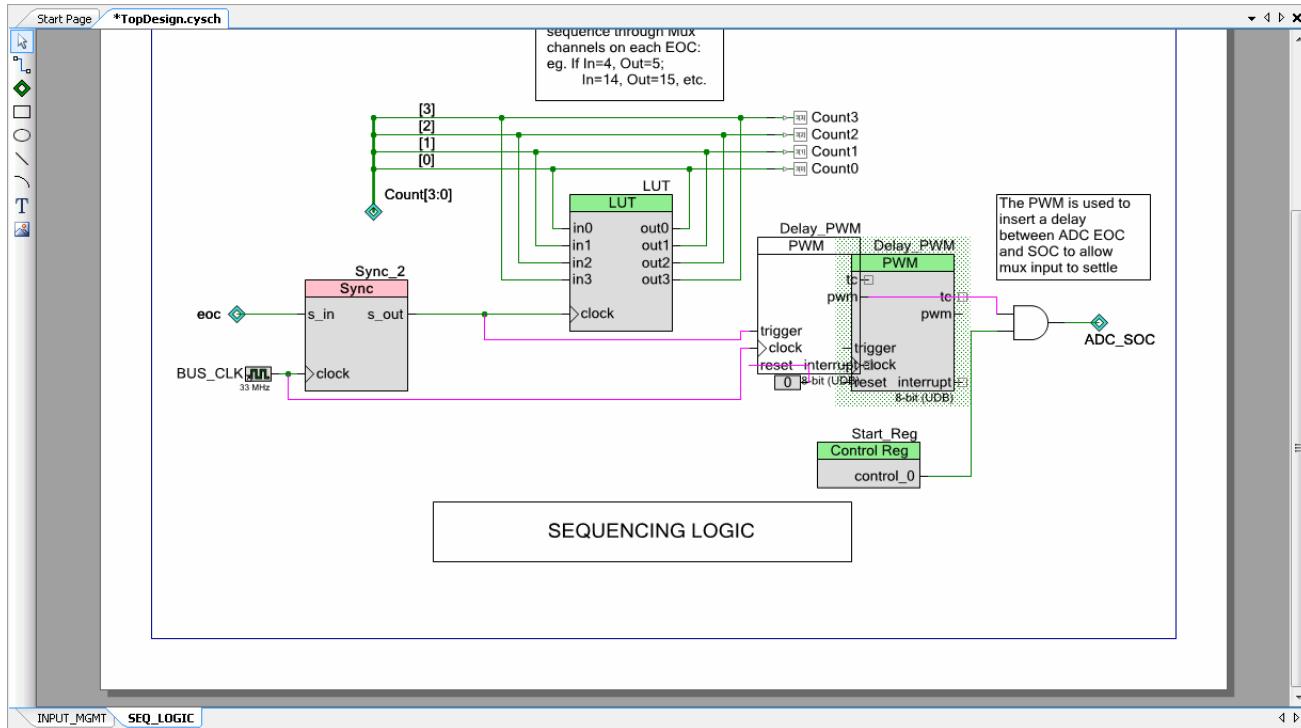
- To select a wire, click on it once with the left mouse button.
- To select a segment of a wire net from one connection point to another, right-click on a wire and select **Select Wire Segment**.
- To select an entire wire net, right-click on a wire and select **Find Wire Trace**.

See Also:

- [Using Design Entry Tools](#)
- [Schematic Editor](#)
- [Design Elements Palette](#)
- [Signal Name](#)

Rubber-Banding

The rubber-banding feature automatically redraws your wires as you move objects on your design schematic. The feature is enabled by default.



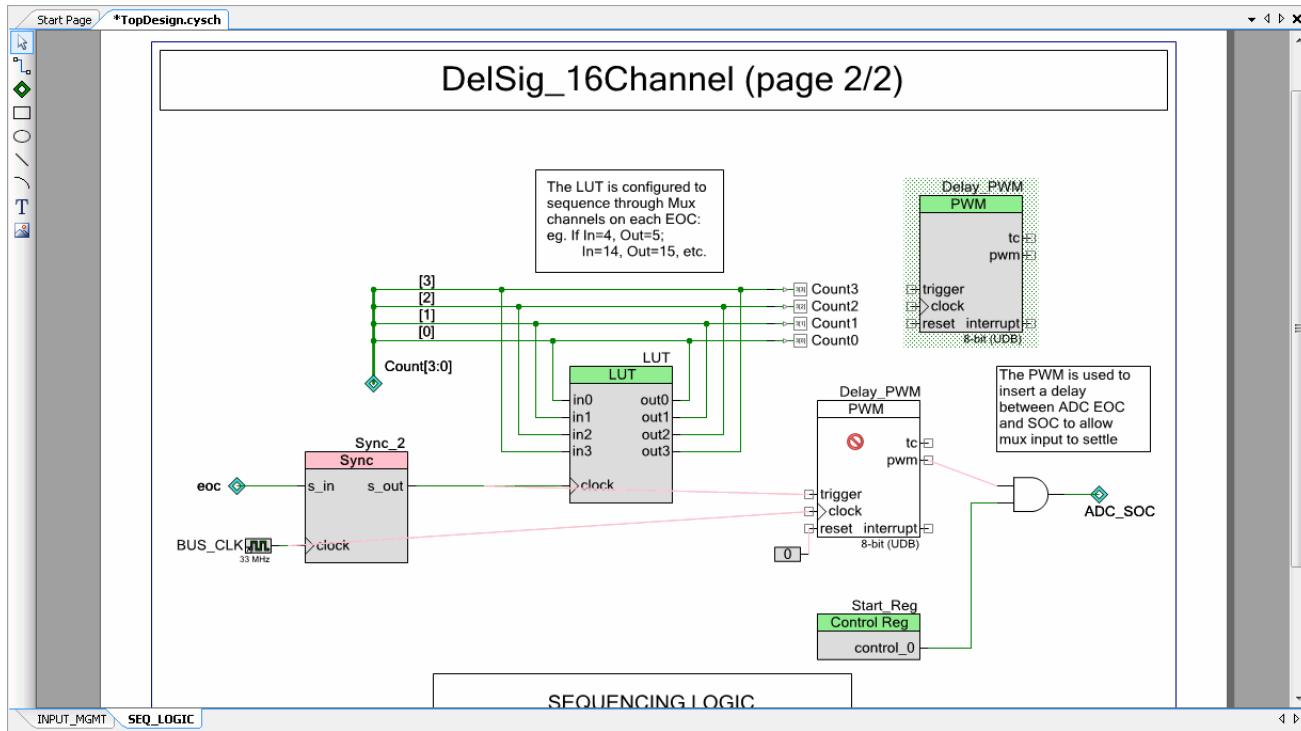
To Use Rubber-Banding:

Select one or more objects on your schematic and move them to another location on your screen. Notice as you move the objects, the wires stretch with the movement. You can move items by dragging the mouse or using [Arrow] keys.

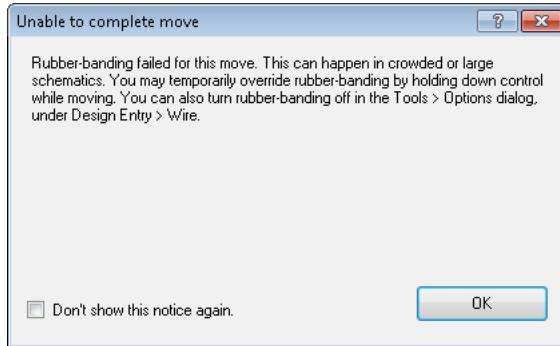
If you stop the movement with the mouse for a moment, PSoC Creator will show a preview of how the wires will be redrawn. Using [Arrow] will not show a preview.

To Correct a Rubber-Banding Problem:

Sometimes, PSoC Creator may not be able to redraw the wires. In this case, the move will not be allowed. The cursor will show a "No"  symbol, and the status bar will display a message indicating the move was not allowed.



If you release the mouse button, the Component will move back to the previous location. A dialog will also display as follows:



Try selecting fewer objects or temporarily disabling rubber-banding. You can also select a larger region of objects and try to move them as a unit.

To Temporarily Disable Rubber-Banding:

When rubber-banding is turned on, press and hold the **[Ctrl]** key while moving a Component. The wire preview will not be shown during a move. If you release the **[Ctrl]** key, the preview will reappear.

If you release the mouse button while holding the **[Ctrl]** key, the wires will not be redrawn for this move, and the connections will be broken.

To Turn Off Rubber-Banding:

1. Open the [Options dialog](#) from the **Tools** menu.
2. Under the Design Entry category, scroll down to the **Wires** section.
3. Find the rubber-banding option and select **Off**.
4. Click **OK** to close the dialog.

Now when you move a Component, the wires are not redrawn by default.

To Temporarily Enable Rubber-Banding:

When rubber-banding is turned off, press the **[Ctrl]** key while moving a Component.

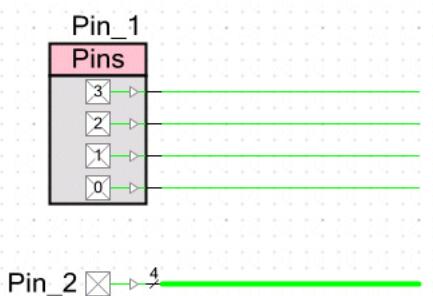
See Also:

- [Design Entry Options](#)
- [Keyboard Shortcuts](#)

Drawing Buses

To draw a bus, you simply connect a wire to a multiple bit connection. The wire inherits the bus width automatically. You may also specify the width of a wire name, using the [Signal Name dialog](#). This allows you to create a bus without connecting it to anything. It also allows you to rip signals from the bus to connect to smaller width signals.

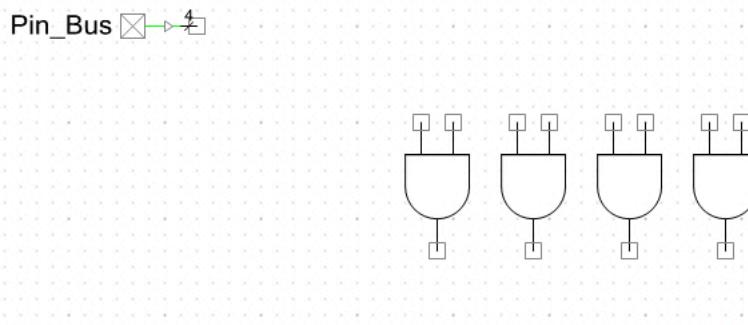
The following diagram shows an example of two Pins Components -- one configured with individual 1-bit terminals connected to wires and the other configured as a 4-bit bus:



To Connect Buses to Smaller Width Signals (Ripping Signals):

You cannot directly connect a bus to a smaller width signal (without receiving DRC errors in the [Notice List window](#)). Instead, you must create a multi-point wire and label the portion of the wire connecting to the smaller width terminal, as shown in the following example:

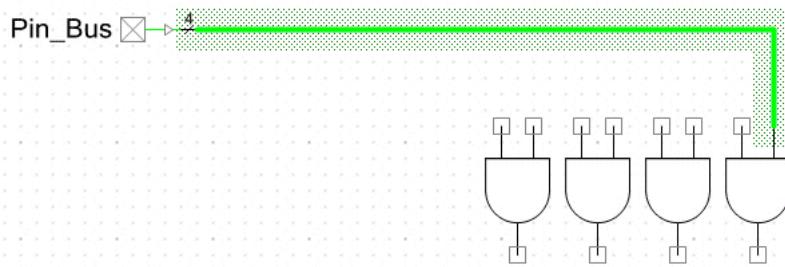
1. Create a 4-bit Pins Component and four AND Components each with a 1-bit terminal width.



Refer to [Configure Component Parameters](#), [Component Catalog](#), and [Working with Shapes](#) as necessary.

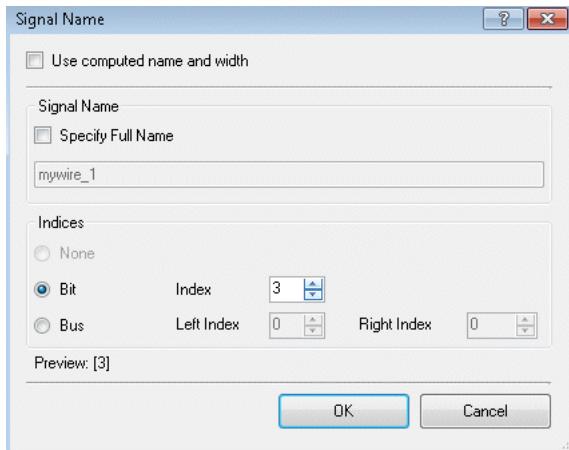
2. Use the **Draw Wire** tool  and make a multi-point connection from the 4-bit port to the farthest single-bit Component terminal.

As soon as the connection is made, an error will display about inconsistent widths (among other connection errors).



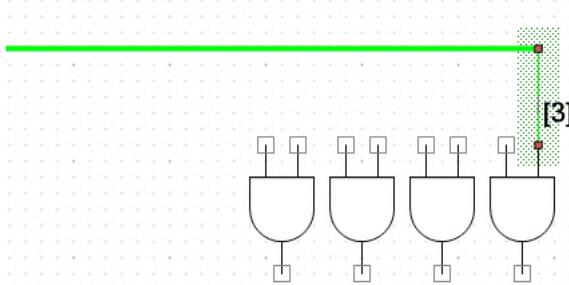
3. Click on the canvas to de-select the entire wire.
4. Double-click on the wire segment connecting to a single-bit terminal.

The Signal Name dialog opens to name the wire. See [Signal Name](#) for more information.

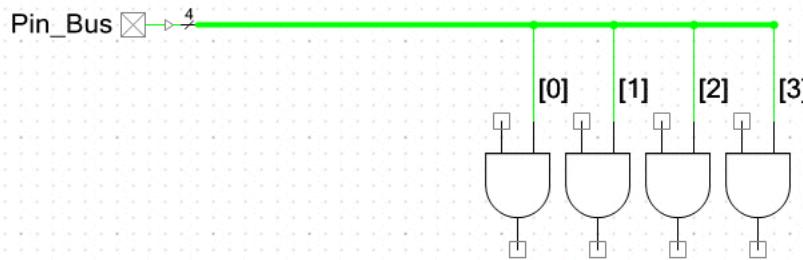


5. De-select Use computed name and width and de-select Specify Full Name.
6. Select the **Bit** option and select **Index [3]**.
7. Click **OK** to close the dialog.

Notice the label displays and the wire connecting to the terminal becomes thin.



8. Connect the bus to each of the other single-bit terminals. Label them [0], [1], and [2], respectively, using the same process as described above.



Tip You can copy and paste the completed wire, and then double-click the label to edit the signal name.

Note You can connect various width signals using the same process and not all bits from an input pin need to be used. However, every bit of an output bus must be driven.

See Also:

- [Signal Name Dialog](#)

- [Working with Wires](#)
- [Wire Labels and Names](#)
- [Notice List Window](#)

Wire Labels and Names

This section provides details about displaying and editing wire labels in the [Schematic Editor](#). A wire can have an explicit User Name, and every (valid) wire has an Effective Name. You can see these names on the schematic canvas, if displayed, or in the [Properties dialog](#).

- **Wire Label** – The wire label is the displayed name of a wire. Displaying the wire label is optional, and it is only displayed for wires with User Names.
- **User Name** – The User Name is the label you specify for a wire, using the [Signal Name dialog](#). Not every wire must have a User Name.
- **Effective Name** – This is the name computed by the connectivity subsystem. The name is derived from the sources in the schematic, for example, connected input schematic terminals. Every (valid) wire has an Effective Name and is never empty.

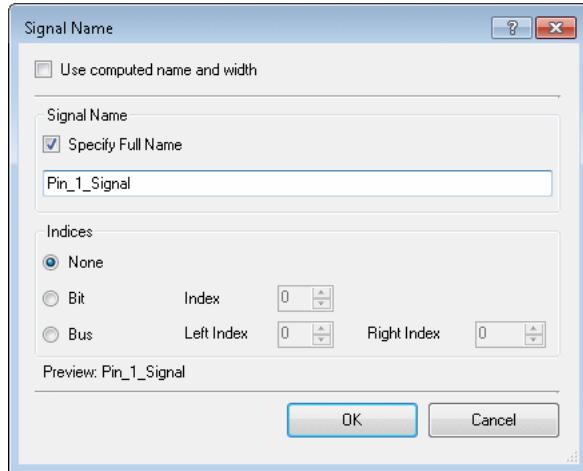
Wire Label Display Scheme in the Schematic Editor:

- If you provide a name, PSoC Creator will display the label with the User Name.
- If the wire does not have a User Name, PSoC Creator will not display the label.
- When you edit the name, the User Name will be set. The Effective Name cannot be set.
- The wire properties in the Properties dialog show the Effective Name and User Name.

Note If you copy and paste a wire with a User Name, PSoC Creator will **not** rename the wire, even though PSoC Creator will rename other pasted elements, such as Components and terminals. There may be some cases where copying and pasting a wire with a User Name connected to terminals and Components will cause DRC errors, and you will have to rename the wire appropriately, or remove the label.

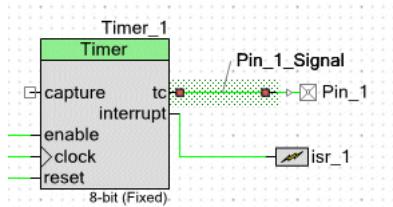
To Set a User Name for a Wire:

1. Double-click on a wire to open the Signal Name dialog.



2. On the dialog, de-select Use computed name and width and select Specify Full Name.
3. Type the desired User Name in the field and click **OK**.

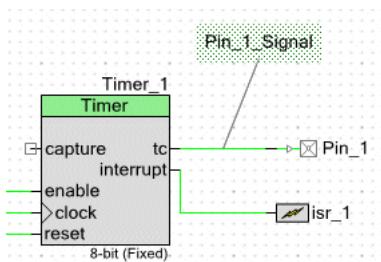
The specified User Name will be shown attached to the wire.



Note If a wire has an existing User Name, you can double-click the label to open the Signal dialog.

To Move a Wire Label:

Click the wire label to select it, and drag it another location. The wire anchor will always display attached to the middle of the wire segment.



See Also:

- [Schematic Editor](#)
- [Properties](#)
- [Signal Name](#)

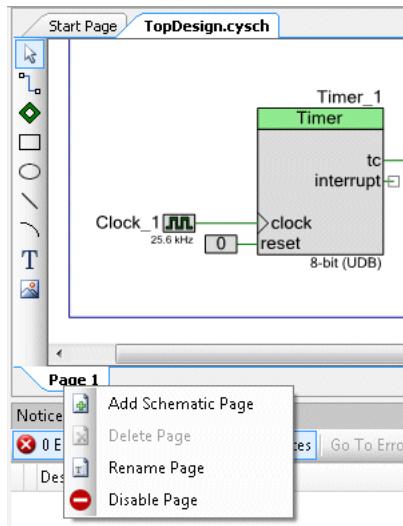
- [Working with Wires](#)

Using Multiple Pages and Connectors

You can add multiple pages to your design if it is too large for one sheet, or if you just want to separate different sections of your design for readability. You connect elements on different pages using sheet connectors. This topic shows you how to add a page to your schematic and how to connect elements on different pages.

To Add a Page:

Right-click on the schematic page tab at the bottom of the schematic, and select **Add Schematic Page**.

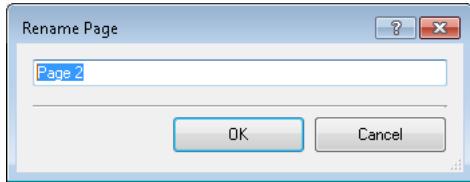


The new page is added as a tabbed document.

To Rename a Page:

1. Right-click on the schematic page tab at the bottom of the schematic, and select **Rename Page**.

The Rename Page dialog displays.



2. Type the name for the page and click **OK**.

The page tab displays the name you entered.

To Delete a Page:

Right-click on the schematic page tab at the bottom of the schematic, and select **Delete Page**.

The selected page tab is removed.

Note You cannot delete the page tab if there is only one.

To Use Sheet Connectors:

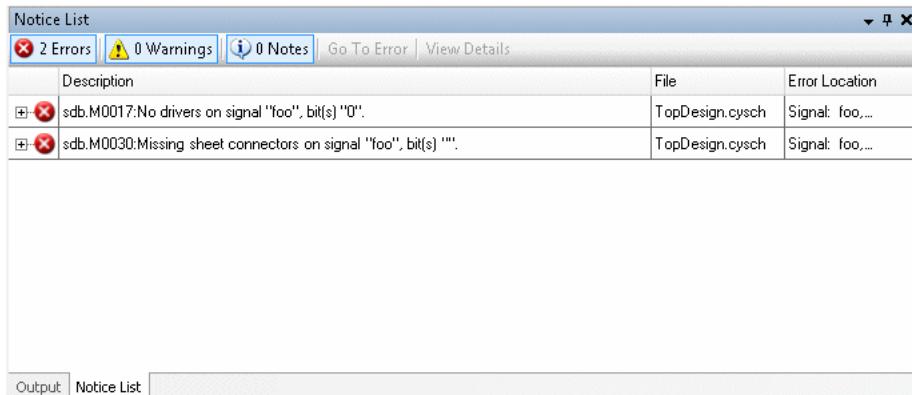
Sheet connectors connect to wires or buses on multiple sheets, with the connection point in the center of the shape. They are unnamed and have no defined flow direction.

Every simple wire name and every bit of a bus present on multiple sheets must be connected to a sheet connector on each sheet where used. When using sheet connectors with input terminals and buses, keep in mind that when a base name is used in an input terminal, it must define the entire signal. So, using bits of a signal that were not defined by the input terminal on a separate page will violate the existing rules. See [Wire Labels and Names](#) and [Drawing Buses](#).

Note You can connect wires by name on the same page of a schematic without sheet connectors.

1. Click the **Wire** icon  and draw a wire on two different pages of your schematic.
2. Double-click each wire and give each wire the same name (e.g., "foo").

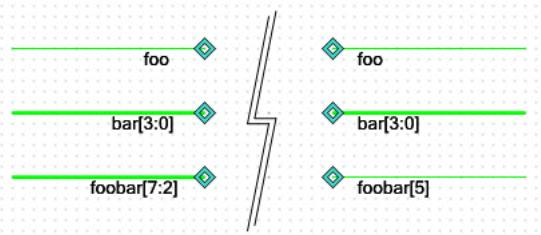
Notice that the Notice List window shows a sheet connector error.



3. Click the **Sheet Connector** icon  and place a connector on each wire on each page.

Once all wires are connected, the Notice List window will clear.

The following image shows some examples of valid sheet connections:



See Also:

- [Working with Wires](#)
- [Wire Labels and Names](#)
- [Wide Terminals and Wires](#)

Disabling/Enabling Schematic Pages

Disabling a schematic page allows you to specify a portion of a design to be excluded from the build. You can disable (and re-enable) one or more pages in your schematic design. This allows you remove sections for testing and debugging, as well as provide different configurations.

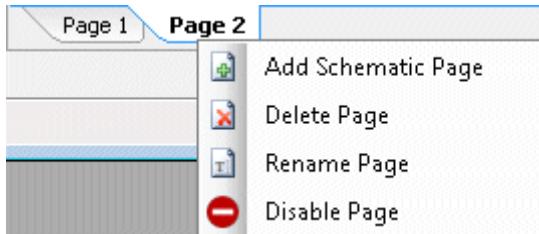
Possible Build Errors

Before disabling a schematic page, make sure the name of the schematic page tab starts with a letter, and includes only alphanumeric and space characters.

When a schematic page is disabled, the system creates a macro in the *cydisabledsheets.h* file, named after that schematic page's tab. If the tab name begins with a number, or is otherwise not a valid C identifier, the system creates an invalid macro name. This will cause build errors.

To Disable a Page:

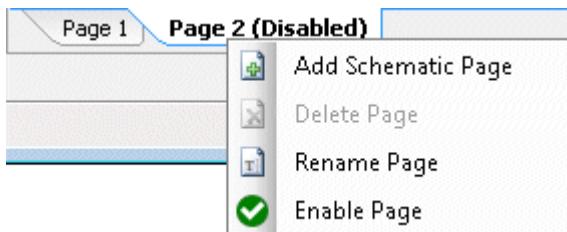
Right-click on the schematic page tab at the bottom of the schematic, and select **Disable Page**.



The schematic page will be disabled and the word "Disabled" will appear on the schematic and the tab.

To Enable a Disabled Page:

Right-click on the schematic page tab at the bottom of the schematic, and select **Enable Page**.



The schematic page will become active again.

Note If there are other active pages in the schematic that have Component instance names that are the same as those on the page you are trying to enable, the [Merge Dialog](#) will display to resolve the conflicts.

See Also:

- [Schematic Editor](#)
- [Merge Dialog](#)

Working with Schematic Terminals

The terminal tools in the [Design Elements Palette](#) for schematics allow you to draw digital input, output, and inout, as well as analog and external terminals.



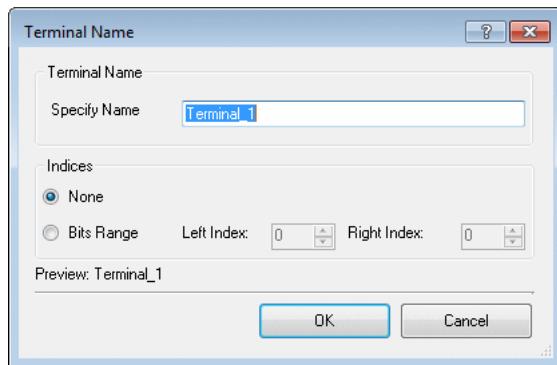
Use schematic terminals when you are implementing a Component with a schematic to represent the external connectivity of the Component within the chip. For more information about creating Components, refer to the [Component Author Guide](#).

Note Schematic terminals are hidden from the Design Elements Palette when you are editing a top-level schematic; they are only available for Component implementation schematics. For more information, see [Creating a New Schematic](#).

To Place a Single Terminal:

1. Click the appropriate **Terminal** tool on the Design Elements Palette and then click on the design canvas.

The Terminal Name dialog displays.



2. Specify the **Terminal Name** and **Index** information, as appropriate.
3. Click **OK**.

To Place Multiple Terminals:

1. Double-click on the appropriate **Terminal** tool in the Design Elements Palette and then click on the design canvas.
2. Click repeatedly on the canvas to place multiple terminals.

This is called "sticky mode."

3. Press [**Esc**] or click the Select Tool  to escape sticky mode.

To Rename a Terminal:

1. Double-click the label to open the Terminal Name dialog.
2. Type the appropriate **Terminal Name**.
3. Click **OK**.

Note If you copy and paste a terminal onto a schematic, PSoC Creator will rename the pasted Component by appending "_n" to the name, where n is next available number.

To Delete a Terminal:

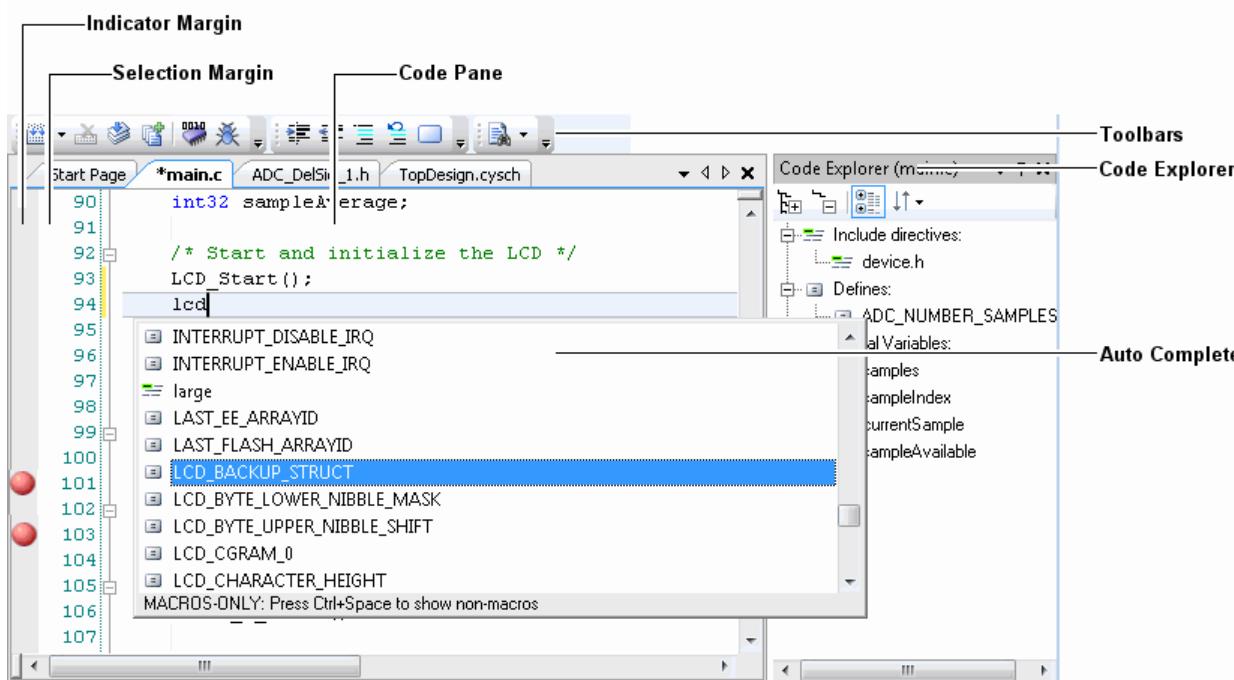
Select the terminal and press [**Delete**] or click .

See Also:

- [Schematic Editor](#)
- [Design Elements Palette](#)
- [Terminal Name](#)

Code Editor

The Code Editor, or text editor, allows you to view and edit source files in PSoC Creator. You can open multiple files in [tabbed document windows](#), and copy and paste among files.



The following describe the different sections of the Text Editor:

- **Code Pane** – The area where code or text is displayed for editing. It provides Autocomplete statement completion for the language in which you are developing. You can navigate back and forth to previous cursor locations, using **[Ctrl]+[-]** and **[Ctrl]+[Shift]+[-]**, respectively.
- **Indicator Margin** – A gray column on the left side of the Code Editor where indicators such as breakpoints, bookmarks, shortcuts, and [Inline Code Diagnostics](#) are displayed. Clicking this area sets a breakpoint on the corresponding line of code. For more information see the [Debugger](#) section.
- **Selection Margin** – A column between the Indicator Margin and the Code Pane where you can click to select lines of code. This area shows line number. Also, changes to code are tracked here when you select **Track Changes** in the Options dialog, under [Text Editor > General](#).
- [Toolbar](#) and [Commands](#)
- [Code Explorer Window](#)
- [Autocomplete](#)

To Open the Text Editor:

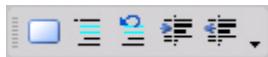
There are various ways to open a file in the Text Editor:

- [Workspace Explorer](#) – Double-click a file.
- [File menu](#) – Select **New** or **Open** to display a file.

See Also:

- [Document Windows](#)
- [Workspace Explorer](#)
- [Code Editor Toolbar](#)
- [Code Editor Context Menu Commands](#)
- [Autocomplete](#)
- [Code Outline Tool Window](#)
- [Find All References](#)
- [Inline Code Diagnostics](#)
- [Find Replace](#)
- [Options Dialog > Text Editor options](#)
- [Go To Line](#)
- [Using the Debugger](#)

Code Editor Toolbar



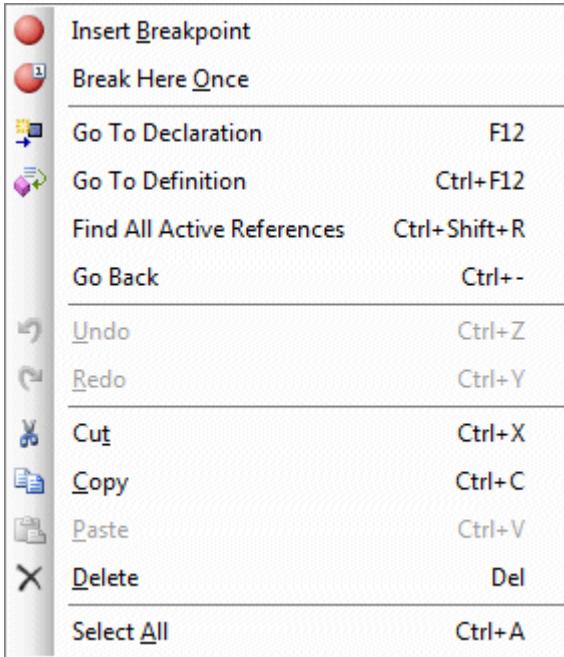
The Text Editor toolbar contains the following commands:

- **Toggle Bookmark** – Adds/removes a bookmark in the margin for the current line.
- **Comment Selection** – Changes selected text to a comment.
- **Uncomment Selection** – Removes comment for selected text
- **Increase Line Indent** – Indents selected text to next tab stop.
- **Decrease Line Indent** – Outdents selected text to the previous tab stop.

Code Editor Context Menu Commands

The Text Editor contains various commands available on right-click – or context – menus. The commands available will vary depending on whether you are editing source code files or running the debugger. The following are the commands available:

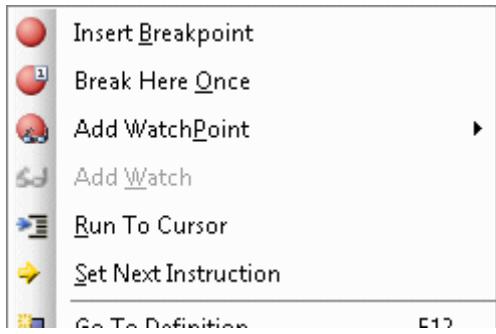
In Edit Mode:



- **Insert Breakpoint** – Adds a new file/line breakpoint to the line that the cursor is currently on. See [Breakpoints Window](#).
- **Break Here Once** – Adds a new temporary file/line breakpoint to the line that the cursor is currently on. After this breakpoint has been hit once, it will automatically be removed.
- **Go To Declaration** – Similar to the Go To Definition, this option jumps to the symbol implementation. However, unlike, the Go To Definition, this jumps to the actual declaration/implementation. Most of the time, this will jump directly to the source file that declared the function. The only cases where this will not occur are if there is no declaration found (e.g., declared in an included library).
- **Go To Definition** – If tool tip information is not sufficient for answering whatever question you might have, this option jumps to the definition of the selected symbol. If the symbol is declared in a different source file, this will jump to the included header file. If the symbol is declared in the current source file, it will jump to that location.
- **Find All Active References** – Allows you to search for all references to a symbol. See [Find All References](#).
- **Go Back** – Go back to previous cursor location.
- **Undo** – Undoes the last edit to the file.
- **Redo** – Undoes the last undo
- **Cut** – Cuts the selected text
- **Copy** – Copies the selected text
- **Paste** – Pastes the text currently in the clipboard into the file
- **Delete** – Deletes the selected portion of text
- **Select All** – Selects the text from the entire document

In Debug Mode:

The context menu contains all the same commands as in edit mode, plus the following commands:



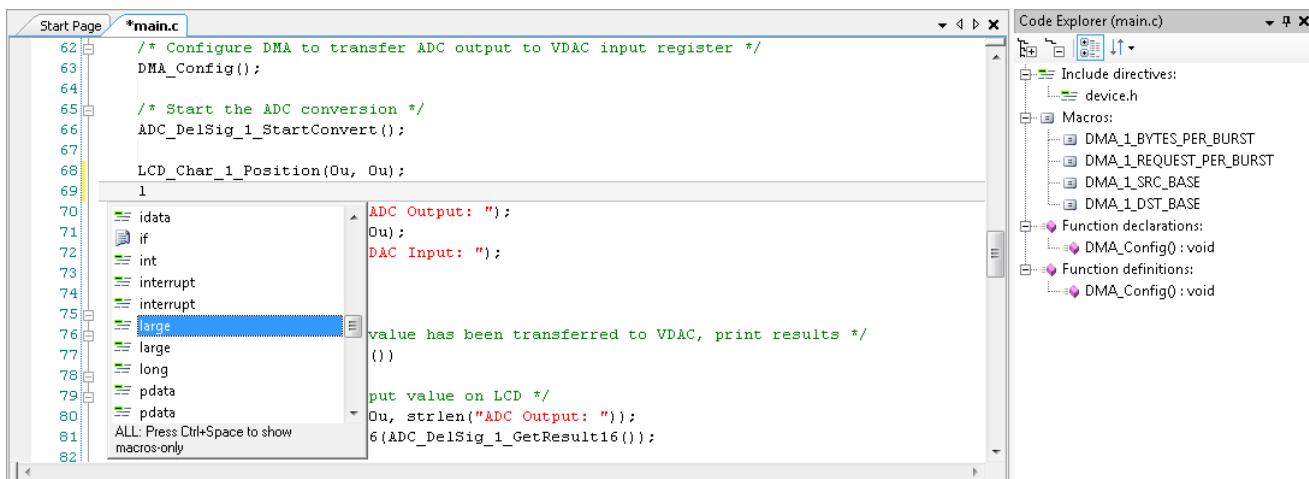
- **Add Watchpoint** – Adds a new watchpoint on the selected variable; see [Variable Watchpoints](#)
- **Add Watch** – Adds the variable, or selected text to the watch window; see [Watch Window](#)
- **Run to Cursor** – Resumes executing code till it reaches the line
- **Set Next Instruction** – Jumps the program to the current line of code

See Also:

- [Text Editor](#)
- [Text Editor Toolbar](#)
- [Using the Debugger](#)
- [Debugger Toolbar Commands](#)
- [Debugger Menu Commands](#)
- [Variable Watchpoints](#)
- [Watch Window](#)

Autocomplete

If enabled, the autocomplete feature provides a context-aware drop-down list of all potentially relevant keywords, types, variables, macros, and functions as you type.



Note This feature is enabled by default. You can disable it under the [Text Editor Options](#).

This feature helps you to write code faster. It also provides enormous value as a documentation source. You no longer need to constantly go back to the datasheet or source file to find the function you need. Simply start typing to see what is available.

Note Completions are limited until the project has been built.

In addition to providing the list of what is available, the Code Editor displays a tool tip of the selected item to provide the full signature. In the case of functions, this shows the return value, the name of the function, and the types and names of all argument variables.

To Use the Feature:

1. If not already enabled, enable the Autocomplete feature in the [Text Editor Options](#) dialog. The feature is enabled by default.
2. Create your design as usual, and click the **Generate Application** button  to allow PSoC Creator to generate or update the various API files.
3. Open the [Code Editor](#) and begin typing; notice the drop-down list opens when it finds items that match. You can also initiate the feature using **[Ctrl] + [Space]**.
4. Scroll through the list to find what you want, or just keep typing until the desired item is highlighted. You can press **[Ctrl] + [Space]** to toggle between show macros only, show non-macros only, or show all.
5. Once the desired item is selected in the list, press **[Tab]** or **[Enter]** to autocomplete the word. **[Esc]** will cancel the autocomplete process.

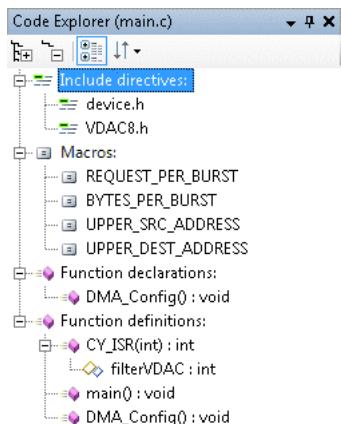
The selected item is inserted and case adjusted to match the actual signature that you completed.

See Also:

- [Code Editor](#)

Code Explorer Window

The code explorer feature allows you to see the entire structure of your source file with a quick glance. With it, you can see what is defined and where. You can also use it to jump to specific areas of your code.



To Display the Code Explorer Tool Window:

This tool window displays automatically when you open the [Code Editor](#). It is located on the right side of the tool. If you close the tool window, you can open it again by selecting **Code Explorer** under the **View** menu.

To Jump to Specific Location:

Double-click an item or press [**Enter**] when the item is highlighted.

The code editor will immediately scroll to that symbol.

Toolbar:



The Code Explorer toolbar contains the following formatting options to change how the outline is presented:

- **Expand All / Collapse All** – Separate buttons to expand or collapse all nodes in the entire tree.
- **Show in Groups** – Toggle button to group various symbols together or view them separately.
- **Sort Order** – Button and pull-down to change the ordering by name or position in the file.

Context Menu:

When you right-click on a node item in the tree, the following commands are available. See also [Code Editor Context Menu Commands](#).



- **Go To Declaration** – This option jumps to the declaration of the selected symbol.
- **Go To Definition** – This option jumps to the definition of the selected symbol.

Icons:

Each item show in the tree includes an icon. The following lists what the icons mean:

-  – Include Directive
-  – Macro
-  – Struct
-  – Typedef
-  – Union
-  – Enum

-  – Enum constant
-  – Function
-  – Argument Variable
-  – Variable

See Also:

- [Code Editor](#)
- [Code Editor Context Menu Commands](#)

Find All References

The Find All References feature allows you to find all items that are using a particular symbol (function, macro, variable, type).

To use This Feature:

Right-click on a symbol in the [Code Editor](#) and select **Find All Active References** from the context menu, or press **[Ctrl] + [Shift] +[R]**.

PSoC Creator scans through the entire project in order to find all uses of the symbol, and displays the results of this search in the [Find Results](#) window.

Double click on an entry in the Find Results to jump to that location in the source code.

See Also:

- [Code Editor](#)
- [Find Results](#)

Inline Code Diagnostics

When enabled, this feature displays compiler diagnostic information directly in the editor. This allows you to see what problems exist in the code without needing to do a full build of the project.



```

14
15 void main()
16 {
17     /* Place your initialization/startup code here (e.g. MyIn:
18     pwm_1_c|                                     use of undeclared identifier 'pwm_1_c'
19     /* Cytrobaire; */ /* Uncomment this line to enable
20
21     for(;;)
22     {
23         /* Place your application code here. */
24     }
25 }
26
27 /* [] END OF FILE */
28

```

Note This feature is enabled by default. You can disable it under the [Text Editor Options](#).

The diagnostics displayed in the editor may not correspond to the errors/warnings generated when performing a build. This is because the editor uses a generic build framework that does not have identical rule checkers as the active toolchain. Because of this, none of the diagnostics are displayed in the [Notice List](#). The errors/warnings displayed in the Notice List are strictly limited to what is generated for the current design configuration.

Note If an error about a missing include is shown, other errors in the document may not be displayed. This occurs because missing include files are considered fatal and limit what additional processing is performed. If a critical include cannot be found, it could very well cause almost every line in the file to be identified as an error, which would obscure the actual problem (missing include file).

To Use the Feature:

Save the current file. Any issues in the code will display with a waved line.

Hover the mouse over the margin indicator or the waved line to see a tooltip of the problem.

The margin on the left displays if the error is a warning or an error.

See Also:

- [Code Editor](#)
- [Reference Tooltips](#)

Reference Tooltips

If enabled, the [Code Editor](#) provides reference tooltips to make the code easier to read and understand, such as what a block of code is doing or what various arguments to a function do.

```
/* Place your initialization/startup code here (e.g. MyIn
PWM_1_SetCaptureMode(1); */

void PWM_1_SetCaptureMode( uint32 triggerMode )
/* CyGlobalIntEnable; */ /* uncomment this line to enable
for(;;)

19 #if(I2C_1_SCB_MODE_I2C_INC)
20 |     I2C_1_SCB_MODE_I2C_INC(0u!= (I2C_1_SCB_MODE_I2C & I2C_1_SCB_MODE))
21 #endif /* (I2C_1_SCB_MODE_I2C_INC) */
22
23 #if(I2C_1_SCB_MODE_SPI_INC || I2C_1_SCB_MODE_UART_INC)
24 |     #include "I2C_1_SPI_UART_PVT.h"
25 #endif /* (I2C_1_SCB_MODE_SPI_INC || I2C_1_SCB_MODE_UART_INC) */
```

Note This feature is enabled by default. You can disable it under the [Text Editor Options](#).

To Use the Feature:

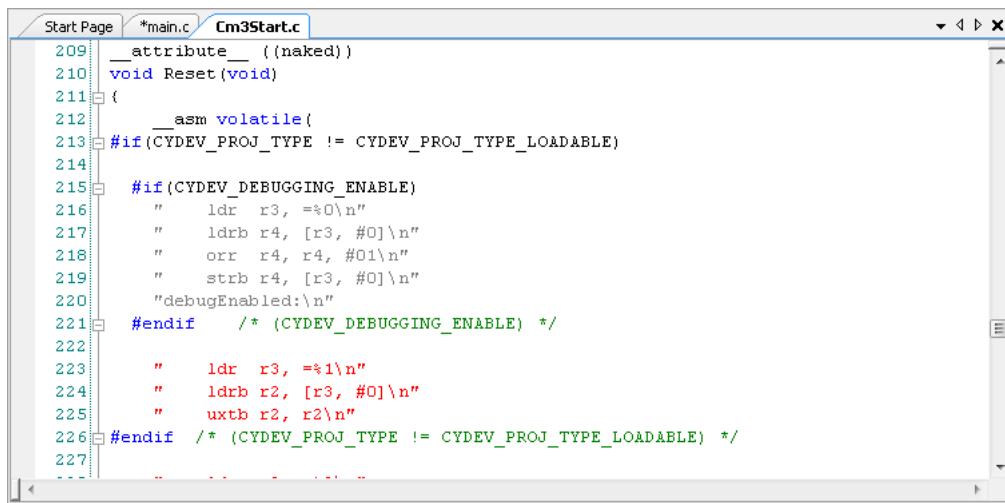
Hover the mouse over any reference type (variable, function, or macro) to see the declaring signature of that item.

See Also:

- [Code Editor](#)

Disabled Code

The [Code Editor](#) identifies disabled code in a grayed-out color. This helps alleviate the issue of code being difficult to read due to congestion.



The screenshot shows a code editor window with the tab 'Cm3Start.c' selected. The code is written in C and includes assembly-like comments. Several lines of code are highlighted in gray, indicating they are disabled. The code includes conditional compilation directives like '#if' and '#endif' and assembly instructions like 'ldr', 'ldrb', 'orr', 'strb', and 'uxtb'.

```
209 __attribute__ ((naked))
210 void Reset(void)
211 {
212     __asm volatile(
213 #if(CYDEV_PROJ_TYPE != CYDEV_PROJ_TYPE_LOADABLE)
214     #if(CYDEV_DEBUGGING_ENABLE)
215         "    ldr r3, =%0\n"
216         "    ldrb r4, [r3, #0]\n"
217         "    orr r4, r4, #01\n"
218         "    strb r4, [r3, #0]\n"
219         "debugEnabled:\n"
220     #endif /* (CYDEV_DEBUGGING_ENABLE) */
221
222         "    ldr r3, =%1\n"
223         "    ldrb r2, [r3, #0]\n"
224         "    uxtb r2, r2\n"
225 #endif /* (CYDEV_PROJ_TYPE != CYDEV_PROJ_TYPE_LOADABLE) */
226     ...
227 }
```

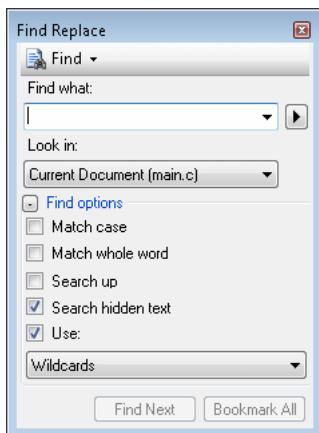
PSoC devices and Components used in designs are highly configurable, plus there are several different devices and toolchains supported by PSoC Creator. This requires the use of a significant number of #ifdef statements throughout the firmware code, which makes reading and debugging the code difficult.

The disabled code feature significantly improves the readability and understandability of the code. It also helps improve the debugging experience. You can clearly see that large blocks of code are disabled and understand immediately why the debugger stepped over them.

This functionality is provided automatically with no necessary user interaction.

Find Replace

The Find Replace dialog is used to locate text within a file and optionally replace it.

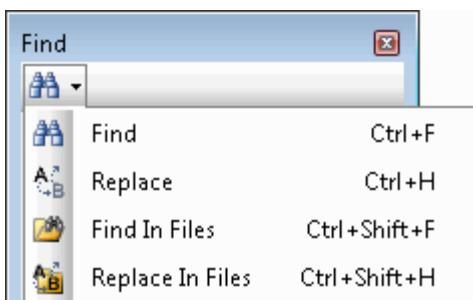


This dialog varies slightly depending on how you opened it. There are separate help topics for [Find in Files](#) and [Replace in Files](#).

To Open the Find Replace Dialog:

Use any of the following methods:

- Press **[Ctrl]+[F]** (for find) or **[Ctrl]+[H]** (for find and replace).
- On the **Edit** menu, select **Find and Replace**, and then select the appropriate find/replace command.
- Click the displayed find/replace button icon or select one of the find/replace options from the pull-down menu.



To Use the Find Replace Dialog:

Find what

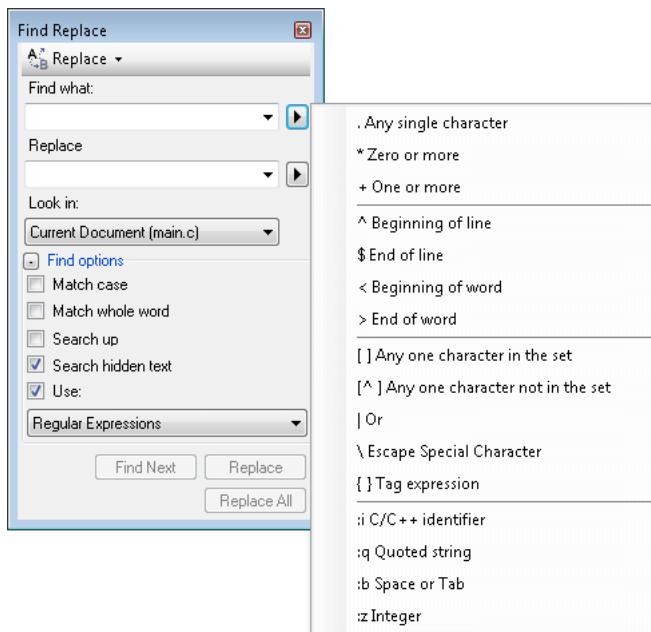
Use this field to specify the string or expression to find. You can reuse one of the last 20 search strings by selecting it from this drop-down list, or type a new text string or expression to find.

Replace with

Use this field to replace instances of the **Find what** string with another string. To delete instances of the **Find what** string, leave this field blank.

Expression Builder

Use the triangular button next to the **Find what** and **Replace with** fields when the **Use** check box is selected under **Find options**.



Click this button to display a list of [wildcards](#) or [regular expressions](#), depending upon the **Use** option selected. Choosing any item from this list adds it to the **Find what** or **Replace with** string.

Look in

Use this pull-down menu to select **Current Document** or **All Open Documents**.

Find options

You can expand or collapse the Find Options section. The following options can be selected or cleared:

- **Match case** – When selected, the Find Results windows will only display instances of the Find what string that are matched both by content and by case. For example, a search for "MyObject" with Match case selected will return "MyObject" but not "myobject" or "MYOBJECT."
- **Match whole word** – When selected, the Find Results windows will only display instances of the Find what string that are matched in complete words. For example, a search for "MyObject" will return "MyObject" but not "CMyObject" or "MyObjectC."

- **Search up** – When selected, files are searched from the insertion point to the top of the file.
- **Search hidden text** – When selected, the search will also include concealed and collapsed text, such as the metadata of a design-time control; a hidden region of an outlined document; or a collapsed class or method.
- **Use** – Indicates how to interpret special characters entered in the **Find what** or **Replace with** fields. The options include:
- **Wildcards** – Special characters such as asterisks (*) and question marks (?) represent one or more characters. See [Wildcards](#).
- **Regular Expressions** – Special notations define patterns of text to match. See [Regular Expressions](#).

Buttons

Click the appropriate button, as follows:

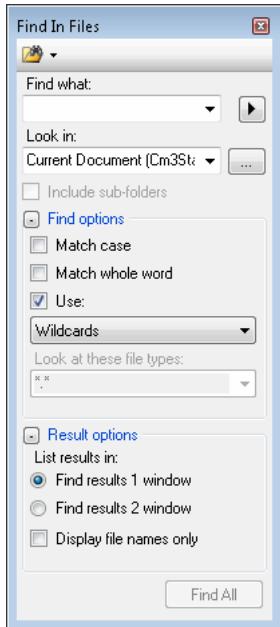
- **Find Next** – Click this button to find the next instance of the **Find what** string within the search scope chosen in **Look in**.
- **Bookmark All** – Click this button to display bookmarks at the left edge of the text editor to indicate each line where an instance of the **Find what** string occurs.
- **Replace** – Click this button to replace the current instance of the **Find what** string with the **Replace with** string, and find the next instance within the **Look in** scope.
- **Replace All** – Click this button to replace all instances of the **Find what** string with the **Replace with** string, in all files within the **Look in** scope.

See Also:

- [Text Editor](#)
- [Find in Files](#)
- [Replace in Files](#)
- [Regular Expressions](#)
- [Wildcards](#)

Find in Files

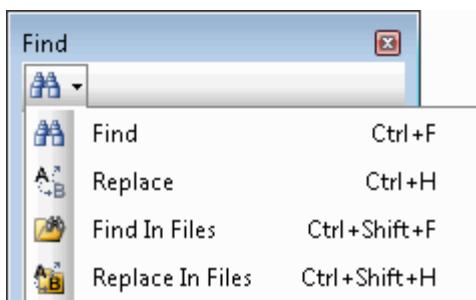
The Find in Files dialog allows you to search the code of a specified set of files for a string or expression. The matches found and actions taken are listed in the Find Results window selected under **Result options**.



To open the Find in Files dialog:

Use any of the following methods:

- Press **[Ctrl]+[Shift]+[F]**.
- On the **Edit** menu, select **Find and Replace**, and then select **Find in Files**.
- Select **Find in Files** from the pull-down menu.



To use the Find in Files dialog:

Find what

Use this field to specify the string or expression to find. You can reuse one of the last 20 search strings by selecting it from this drop-down list, or type a new text string or expression to find.

Expression Builder

Use the triangular button next to the **Find what** and field when the **Use** check box is selected under **Find options**.

Click this button to display a list of [wildcards](#) or [regular expressions](#), depending upon the **Use** option selected. Choosing any item from this list adds it into the **Find what** string.

Look in

Use this pull-down menu to select **Current Document** or **All Open Documents**.

Click the [...] button to select a directory in which to search. You can also check the **Include subfolders** check box to search sub folders of the specified search directory.

Find options

You can expand or collapse the Find Options section. The following options can be selected or cleared:

- **Match case** – When selected, the Find Results windows will only display instances of the Find what string that are matched both by content and by case. For example, a search for "MyObject" with Match case selected will return "MyObject" but not "myobject" or "MYOBJECT."
- **Match whole word** – When selected, the Find Results windows will only display instances of the Find what string that are matched in complete words. For example, a search for "MyObject" will return "MyObject" but not "CMyObject" or "MyObjectC."
- **Use** – Indicates how to interpret special characters entered in the **Find what** or **Replace with** fields. The options include:
 - **Wildcards** – Special characters such as asterisks (*) and question marks (?) represent one or more characters. See [Wildcards](#).
 - **Regular Expressions** – Special notations define patterns of text to match. See [Regular Expressions](#).
- **Look at these file types** – This list indicates the types of files to search through in the **Look in** directories. If this field is left blank, all of the files in the **Look in** directories will be searched.
 - Select any item in the list to enter a preconfigured search string that will find files of those particular types.
 - To find a type of file not available from the drop-down list, enter an asterisk (*) wildcard for the file name, followed by a period (.) and the desired file extension. To find more than one file type, enter multiple file extensions separated by a semicolon (;).

Result options

You can expand or collapse the Result options section. The following options can be selected or cleared:

- **Find Results 1 window** – Select this option to display the results of the current search in the Find Results 1 window. This window opens automatically to display your search results. To open this window manually, select **Find Results** from the **View** menu and choose **Find Results 1**.
- **Find Results 2 window** – Select this option to display the results of the current search in the Find Results 2 window. This window opens automatically to display your search results. To open this window manually, select **Find Results** from the **View** menu and choose **Find Results 2**.
- **Display file names** – Select this check box to display a list of files containing search matches rather than displaying the search matches themselves.

Buttons

Click the appropriate button, as follows:

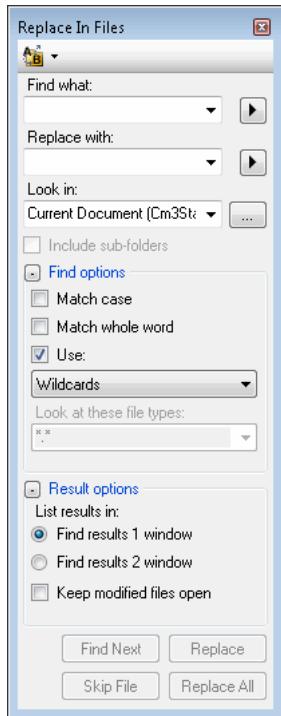
- **Find All** – Click this button to find all instances of the **Find what** string within the search scope chosen in **Look in**. The results are displayed in the Results window chosen under **Result options**.

See Also:

- [Replace in Files](#)
- [Find Replace](#)
- [Find Results](#)
- [Regular Expressions](#)
- [Wildcards](#)

Replace in Files

The Replace in Files dialog allows you to search the code of a specified set of files for a string or expression and change some or all of the matches found. The matches found and actions taken are listed in the Find Results window selected under **Result Options**.

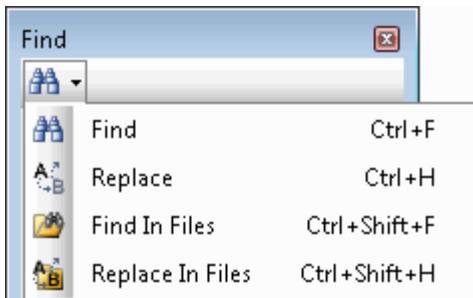


To Open the Replace in Files Dialog:

Use any of the following methods:

- Press **[Ctrl]+[Shift]+[H]**.

- On the **Edit** menu, point to **Find and Replace**, and then click **Replace in Files**.
- Select **Replace in Files** from the pull-down menu.



To Use the Replace in Files Dialog:

Find what

Use this field to specify the string or expression to find. You can reuse one of the last 20 search strings by selecting it from this drop-down list, or type a new text string or expression to find.

Expression Builder

Use the triangular button next to the **Find what** and field when the **Use** check box is selected under **Find options**.

Click this button to display a list of [wildcards](#) or [regular expressions](#), depending upon the **Use** option selected. Choosing any item from this list adds it into the **Find what** string.

Look in

Use this pull-down menu to select **Current Document** or **All Open Documents**.

Click the [...] button to select a directory in which to search. You can also check the **Include subfolders** check box to search sub folders of the specified search directory.

Find options

You can expand or collapse the Find Options section. The following options can be selected or cleared:

- **Match case** – When selected, the Find Results windows will only display instances of the Find what string that are matched both by content and by case. For example, a search for "MyObject" with Match case selected will return "MyObject" but not "myobject" or "MYOBJECT."
- **Match whole word** – When selected, the Find Results windows will only display instances of the Find what string that are matched in complete words. For example, a search for "MyObject" will return "MyObject" but not "CMyObject" or "MyObjectC."
- **Use** – Indicates how to interpret special characters entered in the **Find what** or **Replace with** fields. The options include:
 - **Wildcards** – Special characters such as asterisks (*) and question marks (?) represent one or more characters. See [Wildcards](#).
 - **Regular Expressions** – Special notations define patterns of text to match. See [Regular Expressions](#).
- **Look at these file types** – This list indicates the types of files to search through in the **Look in** directories. If this field is left blank, all of the files in the **Look in** directories will be searched.

- Select any item in the list to enter a preconfigured search string that will find files of those particular types.
- To find a type of file not available from the drop-down list, enter an asterisk (*) wildcard for the file name, followed by a period (.) and the desired file extension. To find more than one file type, enter multiple file extensions separated by a semicolon (;).

Result options

You can expand or collapse the Result options section. The following options can be selected or cleared:

- **Find Results 1 window** – Select this option to display the results of the current search in the Find Results 1 window. This window opens automatically to display your search results. To open this window manually, select **Find Results** from the **View** menu and choose **Find Results 1**.
- **Find Results 2 window** – Select this option to display the results of the current search in the Find Results 2 window. This window opens automatically to display your search results. To open this window manually, select **Find Results** from the **View** menu and choose **Find Results 2**.
- **Keep modified file open after** – Select this check box to leave open the files that were modified during the Replace in Files operation..

Buttons

Click the appropriate button, as follows:

- **Find Next** – Click this button to find the next instance of the **Find what** string within the search scope chosen in **Look in**.
- **Replace** – Click this button to replace the current instance of the Find what string with the Replace with string, and find the next instance within the Look in scope.
- **Replace All** – Click this button to replace all instances of the Find what string with the Replace with string, in all files within the Look in scope.
- **Skip File** – Becomes available when the Look in list includes multiple files. Click this button if you do not want to search or modify the current file. The search will continue in the next file on the Look in list.

See Also:

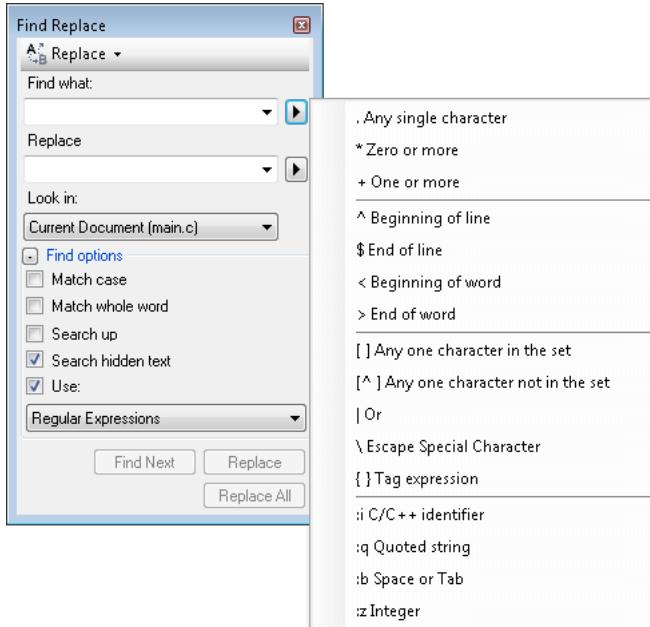
- [Find in Files](#)
- [Find Replace](#)
- [Find Results](#)
- [Regular Expressions](#)
- [Wildcards](#)

Regular Expressions

Regular expressions are a concise and flexible notation for finding and replacing patterns of text. A specific set of regular expressions can be used in the **Find what** field of the [Find Replace window](#).

To enable the use of regular expressions in the **Find what** field during find and replace operations, select **Use > Regular Expressions** under **Find Options**.

The triangular button next to the **Find what** field displays a list of the most commonly used regular expressions.



When you choose any item from the Expression Builder, it is inserted into the **Find what** string.

Note There are syntax differences between the regular expressions that can be used in **Find what** strings and those that are valid in .NET Framework programming. For example, in Find Replace, the braces notation {} is used for tagged expressions. So the expression zo{1} matches all occurrences of zo followed by the tag 1, as in Alonzo1 and Gonzo1. But within the .NET Framework, the notation {} is used for quantifiers. So the expression zo{1} matches all occurrences of z followed by exactly one o, as in "zone" but not in "zoo."

Regular Expressions for Find and Replace:

The following are the regular expressions available in the Reference List.

Expression	Syntax	Description
Any character	.	Matches any single character except a line break.
Zero or more	*	Matches zero or more occurrences of the preceding expression, making all possible matches.
One or more	+	Matches at least one occurrence of the preceding expression.
Beginning of line	^	Anchors the match string to the beginning of a line.
End of line	\$	Anchors the match string to the end of a line.
Beginning of word	<	Matches only when a word begins at this point in the text.

Expression	Syntax	Description
End of word	>	Matches only when a word ends at this point in the text.
Line break	\n	Matches a platform-independent line break. In a Replace expression, inserts a line break.
Any one character in the set	[]	Matches any one of the characters within the []. To specify a range of characters, list the starting and ending character separated by a dash (-), as in [a-z].
Any one character not in the set	[^...]	Matches any character not in the set of characters following the ^.
Or		Matches either the expression before or the one after the OR symbol (). Mostly used within a group. For example, (sponge mud) bath matches "sponge bath" and "mud bath."
Escape	\	Matches the character that follows the backslash (\) as a literal. This allows you to find the characters used in regular expression notation, such as { and ^. For example, \^ Searches for the ^ character.
C/C++ Identifier	:i	Matches the expression ([a-zA-Z_][\$][a-zA-Z0-9_*\$]*).
Quoted string	:q	Matches the expression ((["^"]*) (['^']*')).
Space or Tab	:b	Matches either space or tab characters.
Integer	:z	Matches the expression ([0-9]+).

Additional Regular Expressions

The list of all regular expressions that are valid in find and replace operations is longer than can be displayed in the Reference List. You can also insert any of the following regular expressions into a **Find what** string:

Expression	Syntax	Description
Minimal zero or more	@	Matches zero or more occurrences of the preceding expression, matching as few characters as possible.
Minimal one or more	#	Matches one or more occurrences of the preceding expression, matching as few characters as possible.
Repeat n times	^n	Matches n occurrences of the preceding expression. For example, [0-9]^4 matches any 4-digit sequence.
Grouping	()	Groups a subexpression.
nth tagged text	\n	In a Find or Replace expression, indicates the text matched by the nth tagged expression, where n is a number from 1 to 9. In a Replace expression, \0 inserts the entire matched text.
Right-justified field	\(w,n)	In a Replace expression, right-justifies the nth tagged expression in a field at least w characters wide.
Left-justified field	\(-w,n)	In a Replace expression, left-justifies the nth tagged expression in a field at least w characters wide.
Prevent match	~(X)	Prevents a match when X appears at this point in the expression. For example, real~(ity) matches the "real" in "realty" and "really," but not the "real" in "reality."

Expression	Syntax	Description
Alphanumeric character	:a	Matches the expression ([a-zA-Z0-9]).
Alphabetic character	:c	Matches the expression ([a-zA-Z]).
Decimal digit	:d	Matches the expression ([0-9]).
Hexadecimal digit	:h	Matches the expression ([0-9a-fA-F]+).
Rational number	:n	Matches the expression (([0-9]+.[0-9]* [0-9]*.[0-9]+) ([0-9]+)).
Alphabetic string	:w	Matches the expression ([a-zA-Z]+).
Escape	\e	Unicode U+001B.
Bell	\g	Unicode U+0007.
Backspace	\h	Unicode U+0008.
Tab	\t	Matches a tab character, Unicode U+0009.
Unicode character	\x##### or \u#####	Matches a character given by Unicode value where ##### is hexadecimal digits. You can specify a character outside the Basic Multilingual Plane (that is, a surrogate) with the ISO 10646 code point or with two Unicode code points giving the values of the surrogate pair.

Standard Unicode Character Properties

The following table lists the syntax for matching by standard Unicode character properties. The two-letter abbreviation is the same as listed in the Unicode character properties database. These may be specified as part of a character set. For example, the expression [:Nd:Nl:No] matches any kind of digit.

Expression	Syntax	Description
Uppercase letter	:Lu	Matches any one capital letter. For example, :Luhe matches "The" but not "the".
Lowercase letter	:Ll	Matches any one lower case letter. For example, :Llhe matches "the" but not "The".
Title case letter	:Lt	Matches characters that combine an uppercase letter with a lowercase letter, such as Nj and Dz.
Modifier letter	:Lm	Matches letters or punctuation, such as commas, cross accents, and double prime, used to indicate modifications to the preceding letter.
Other letter	:Lo	Matches other letters, such as gothic letter ahsa.
Decimal digit	:Nd	Matches decimal digits such as 0-9 and their full-width equivalents.
Letter digit	:Nl	Matches letter digits such as roman numerals and ideographic number zero.
Other digit	:No	Matches other digits such as old italic number one.
Open punctuation	:Ps	Matches opening punctuation such as open brackets and braces.
Close punctuation	:Pe	Matches closing punctuation such as closing brackets and braces.
Initial quote punctuation	:Pi	Matches initial double quotation marks.

Expression	Syntax	Description
Final quote punctuation	:Pf	Matches single quotation marks and ending double quotation marks.
Dash punctuation	:Pd	Matches the dash mark.
Connector punctuation	:Pc	Matches the underscore or underline mark.
Other punctuation	:Po	Matches (,), ?, !, @, #, %, &, *, \, (:), (;), ', and /.
Space separator	:Zs	Matches blanks.
Line separator	:Zl	Matches the Unicode character U+2028.
Paragraph separator	:Zp	Matches the Unicode character U+2029.
Non-spacing mark	:Mn	Matches non-spacing marks.
Combining mark	:Mc	Matches combining marks.
Enclosing mark	:Me	Matches enclosing marks.
Math symbol	:Sm	Matches +, =, ~, , <, and >.
Currency symbol	:Sc	Matches \$ and other currency symbols.
Modifier symbol	:Sk	Matches modifier symbols such as circumflex accent, grave accent, and macron.
Other symbol	:So	Matches other symbols, such as the copyright sign, pilcrow sign, and the degree sign.
Other control	:Cc	Matches Unicode control characters such as TAB and NEWLINE.
Other format	:Cf	Formatting control character such as the bi-directional control characters.
Surrogate	:Cs	Matches one half of a surrogate pair.
Other private-use	:Co	Matches any character from the private-use area.
Other not assigned	:Cn	Matches characters that do not map to a Unicode character.

Additional Properties

In addition to the standard Unicode character properties, the following additional properties may be specified as part of a character set.

Expression	Syntax	Description
Alpha	:Al	Matches any one character. For example, :Alhe matches words such as "The", "then", and "reached".
Numeric	:Nu	Matches any one number or digit.
Punctuation	:Pu	Matches any one punctuation mark, such as ?, @, ', and so on.
White space	:Wh	Matches all types of white space, including publishing and ideographic spaces.
Bidi	:Bi	Matches characters from right-to-left scripts such as Arabic and Hebrew.
Hangul	:Ha	Matches Korean Hangul and combining Jamos.

Expression	Syntax	Description
Hiragana	:Hi	Matches hiragana characters.
Katakana	:Ka	Matches katakana characters.
Ideographic/Han/Kanji	:Id	Matches ideographic characters, such as Han and Kanji.

See Also:

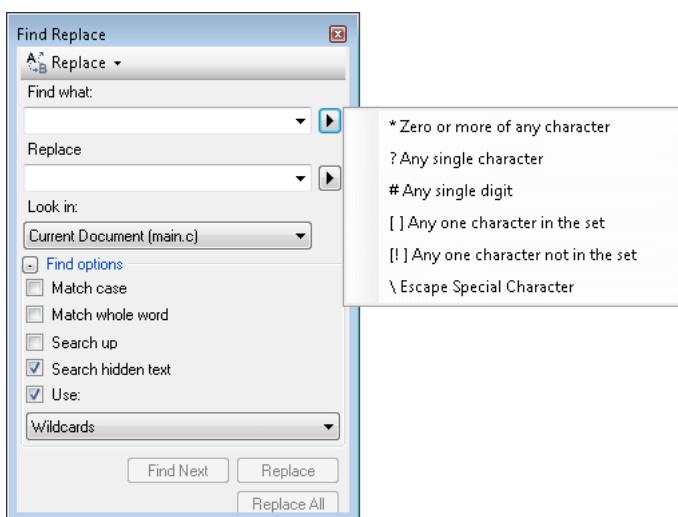
- [Wildcards](#)
- [Find Replace](#)
- [Find in Files](#)
- [Replace in Files](#)

Wildcards

The following expressions can replace characters or digits in the Find what field of the [Find and Replace window](#).

To enable the use of regular expressions in the **Find what** field during find and replace operations, select **Use > Wildcards** under **Find Options**.

The triangular button next to the **Find what** field displays a list of the available wildcards. When you choose any item from the Reference List, it is inserted into the **Find what** string.



Wildcards for Find and Replace:

The following are the wildcards available in the Reference List.

Expression	Syntax	Description
Any single character	?	Matches any single character.

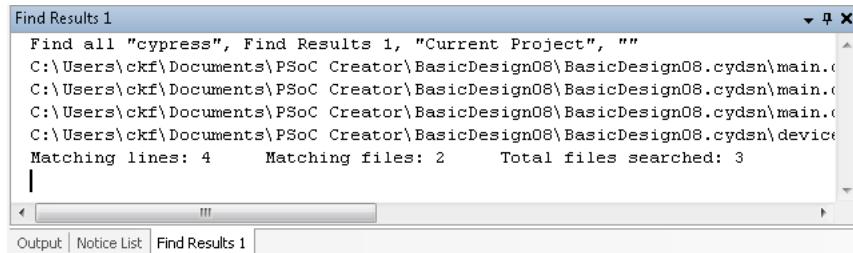
Expression	Syntax	Description
Any single digit	#	Matches any single digit. For example, 7# matches numbers that include 7 followed by another number, such as 71, but not 17.
Characters not in set	[!]	Matches any one character that is not specified in the set.
Escape	\	Matches the character that follows the backslash (\) as a literal. This allows you to find the characters used in wildcard notation, such as * and #.
One or more characters	*	Matches any one or more characters. For example, new* matches any text that includes "new", such as newfile.txt.
Set of characters	[]	Matches any one of the characters specified in the set.

See Also:

- [Regular Expressions](#)
- [Find Replace](#)
- [Find in Files](#)
- [Replace in Files](#)

Find Results

The Find Results window displays matches found when using the [Find in Files](#) and [Replace in Files](#) dialogs.



There are two Find Results windows. The **Result options** allow you to choose the Find Results window where any matches found will be listed. The selected Find Results window opens automatically whenever matches are found.

To Display Find Results Window Manually:

Select **Find Results** from the **View** menu and choose **Find Results 1** or **Find Results 2**.

To Select to a Match:

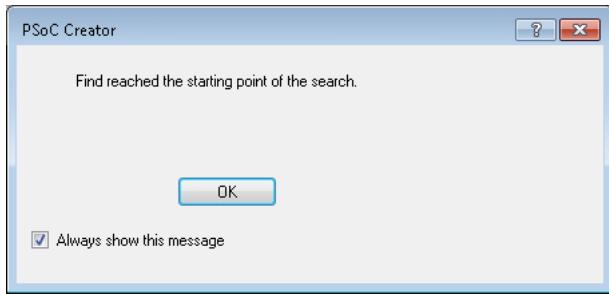
Double-click any line in the results list. The source file is displayed in the [Text Editor](#) with the insertion point placed where the matched text begins. A symbol appears in the indicator margin of the Editor to mark the line that includes the match, and the status bar displays its full text.

See Also:

- [Text Editor](#)
- [Find in Files](#)
- [Replace in Files.](#)

Search Result

The Search Result dialog displays informational messages for search results as part of [Find](#).



This dialog will only display when you select the option under [Text Editor Options](#).

You can disable this dialog by de-selecting the **Always show this message** check box.

The messages you may see with this dialog include:

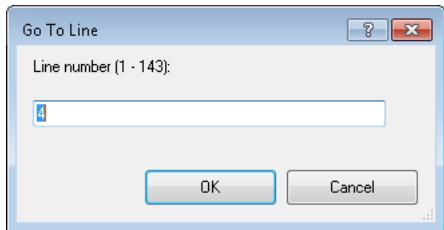
- The following specified text was not found: xxxx
- Find reached the starting point of the search.
- No more occurrences found in the specified documents.
- # occurrence(s) replaced.

See Also:

- [Text Editor](#)
- [Find Replace](#)
- [Text Editor Options](#)

Go To Line

The Go To Line dialog is used to go to a specific line of code.



To Open the Dialog:

Press **[Ctrl] + [G]** or select **Go To** from the **Edit** menu.

To Go to a Specific Line:

Type the line number and click **OK**.

The cursor goes to the specified line number.

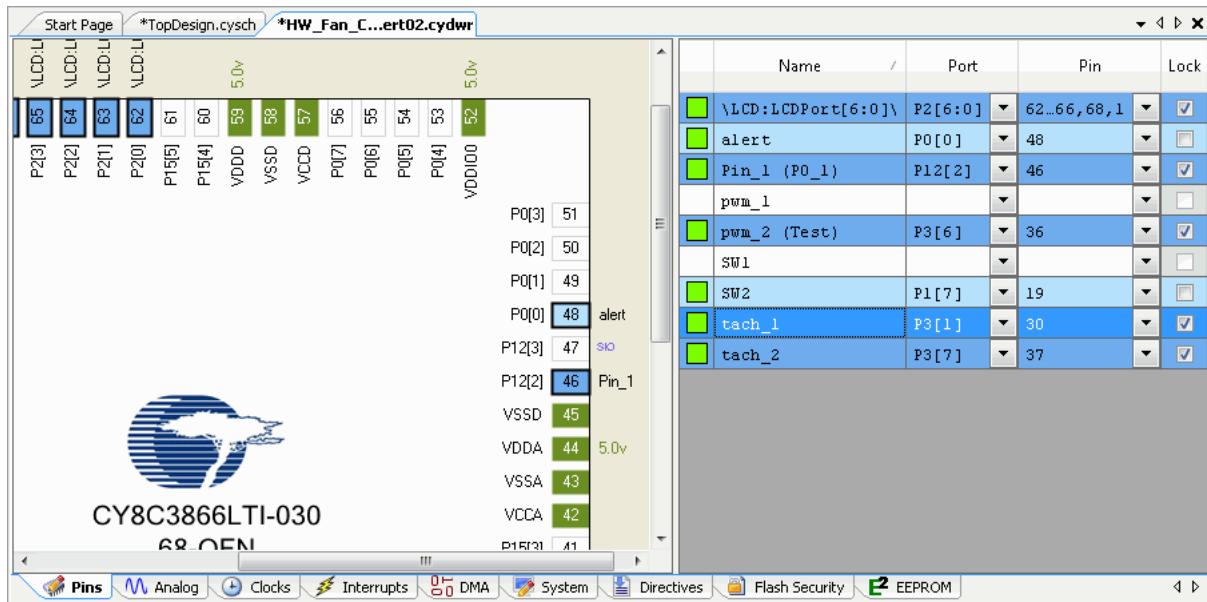
See Also:

- [Text Editor](#)

Design-Wide Resources

The PSoC Creator Design-Wide Resources (DWR) system provides a single location to manage all the resources in your design. Such resources include pins, clocks, interrupts, DMA, etc. Each new [design project](#) provides a default design-wide resources file (<project>.cydwr) file with the same name as the project.

Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.



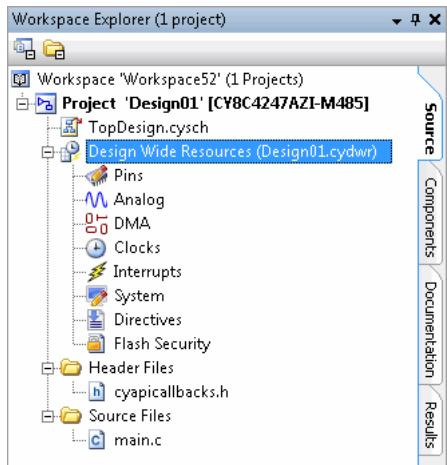
Only design projects can have a <project>.cydwr file, and there can be only one file per design project. All modifications to the DWR information are stored in this file. This design-level information is stored in a way that makes it portable between devices.

Note During the process of selecting a different device, if any errors will exist if the selection were to continue, you will be prompted before the device selection has actually changed. At this point, you can cancel the device change or continue and the appropriate errors will be generated.

To Open the <project>.cydwr File:

If not already displayed, open the **Source** tab of [Workspace Explorer](#) for a design project.

The <project>.cydwr file is located in the project tree, under the project's *TopDesign.cysch* file.



Double-click the appropriate resource to open its editor.

The file opens as a [tabbed document](#) in the work area, and it allows you to access the other design-wide resources in your project.

You can switch between the different resources by clicking the appropriate tab, but you can only edit one resource at a time.

To Add a <project>.cydwr File to a Design Project:

You cannot copy or cut the <project>.cydwr file from within PSoC Creator directly; however, you can add the file as an existing item to another design project. You can also add a new .cydwr file to a design project.

- **Add Existing Item** – Adds an existing .cydwr file to the design project. The file will be copied from the selected location into the design project's folder. It will also be renamed to match the project name.
- **Add New Item** – Adds a new .cydwr file to the design project. If one already exists, a message will display to ask if you want to overwrite the file.

To Delete/Exclude a <project>.cydwr File:

You can delete the <project>.cydwr file or exclude it from your project. If no <project>.cydwr file exists inside the project, only default values will be used and you cannot edit them.

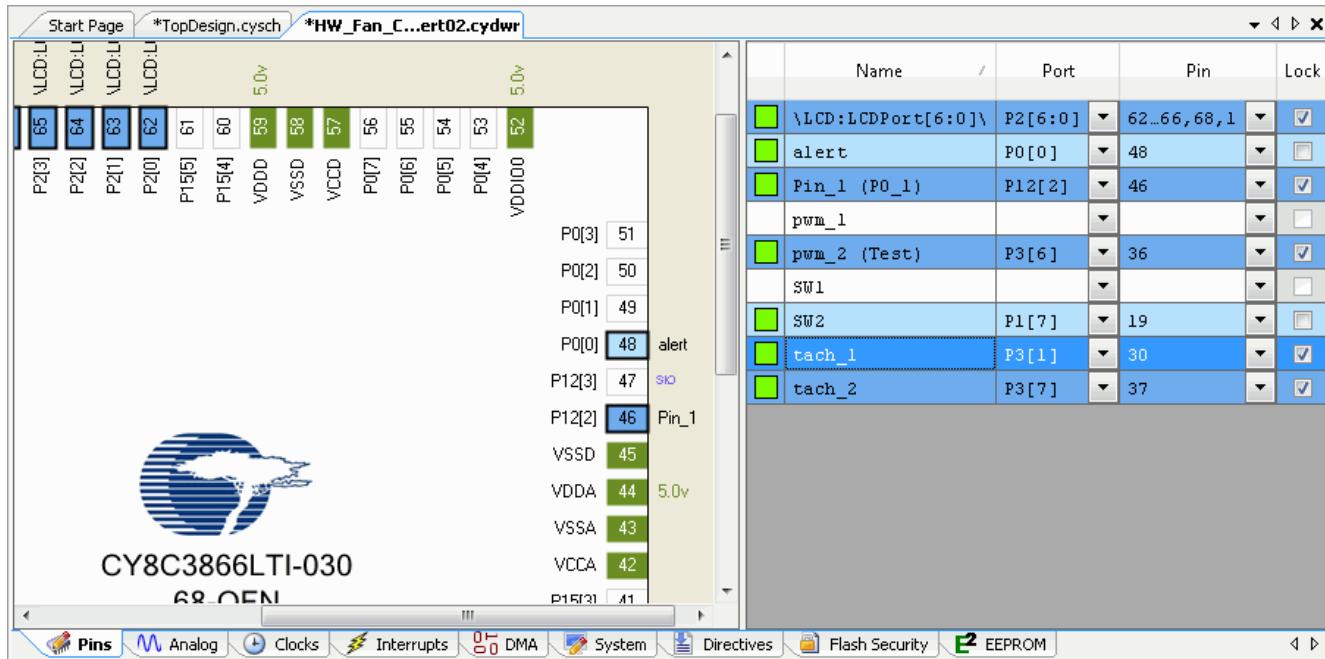
See Also:

- [Pin Editor](#)
- [Analog Device Editor](#)
- [Clock Editor](#)
- [MFT Editor](#)
- [Interrupt Editor](#)

- [DMA Editor](#)
- [System Editor](#)
- [Directives Editor](#)
- [Flash Security Editor](#)
- [EEPROM Editor](#)

Pin Editor

The Pin Editor consists of an interactive image of the selected device, as well as a table of available signals in your design. This editor allows you to manually assign and/or lock pins in your device before PSoC Creator executes the [place and route operation](#) of the build process. If you don't assign pins, or if you manually unassign them, PSoC Creator will automatically assign them during the next build. Assigned and locked pins will stay in the same location for each subsequent build. Unlocked pins could potentially be moved on subsequent builds, depending on resource usage.



Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.

To Open the Pin Editor:

Double-click the .cydwr file in the **Source** tab of [Workspace Explorer](#).

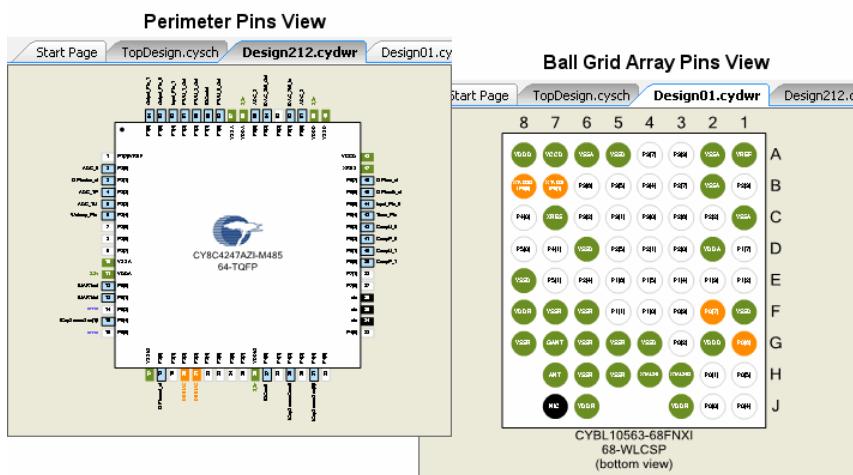
The file opens as a [tabbed document](#) in the work area, and it allows you to access the various design-wide resources in your project. The Pin Editor (**Pins** tab) displays on top by default. If another editor is displayed, click the **Pins** tab to bring it to the top.

Device Image:

The selected device image shows the various pins and ports. Each pin contains its corresponding pin number. Each pin's functionality (i.e., Vcc, n/c, etc.) or port name displays next to or inside the pin. Hovering the mouse over a particular pin will show all of the capabilities of that pin.

Perimeter vs. Ball Grid Array

For TQFP, QFN, and SSOP/SOIC devices, the Pin Editor shows the pins in a perimeter view. For BGA/CSP devices, it shows the pins using a ball grid array view. The following image shows an example of each view.



The coloring of the pins, as well as the port/pin numbers and labels are the same for both views. The process for assigning and locking pins is also the same for both views. However, for the perimeter view, any assigned signal name displays adjacent to the pin, but it doesn't for the ball grid array view.

Pin Coloring and Style

The style of the pin can help identify certain characteristics of the pin, as described in the following tables.

Static Image

Pin Style	Description
Text Color	Black with white text indicates this is a no-connect pin.
Text Color	Dark green with white text indicates this is a power pin.
Text Color	Orange with white text indicates this is a reserved pin. The reason the pin is reserved (that is, used for debugging, used for external crystal, etc.) will be displayed next to the pin in orange.
Text Color	White with black text and a light gray border indicates an unassigned port pin.
Text Color	Light blue with black text and a black border indicates an assigned, unlocked, port pin.
Text Color	Dark blue with black text and a black border indicates an assigned, locked, port pin.
Text Color (or Text Color)	Red with black or white text means this is an invalid pin assignment. Black text means that the current assignment is invalid, but it may be possible to assign other pins to this port; white text means the pin is either no-connect or power. This state could occur when switching the selected device after making some pin assignments. An error will be added to the Notice List for each invalid assignment.
	Pins that have signals assigned to them are drawn with a black border.

Dragging a Pin

Pin Style	Description
Text Color	White with black text indicates that pin guidance information has not yet been calculated. This pin may or may not be a valid assignment once it has been calculated.
Text Color	Green with black text indicates this is a valid (recommended) assignment.
Text Color	Yellow with black text indicates this is a valid assignment, but there is a consequence if you select it (for example, resource usage).
Text Color (or Text Color)	Gray with black or white text means this is not a legal pin assignment due to silicon or design constraint.
	Pins that have signals assigned to them are drawn with a black border.

Signal Table:

The signal table contains all the signals in a table format with the following columns:

Column	Description
(Status)	If assigned, this contains an indicator for the assignment's validity, as shown in the table under Dragging a Pin . If an illegal assignment has been made (and locked), there will be an error icon for that signal.
Name	The name of the signal as defined for the pin. If the pin has an alias, it is shown in parentheses.
Port	The device's pin shown in port form. This field can also be used to make a pin assignment by selecting the desired port[pin] from the drop-down list. An asterisk (*) indicates that a particular assignment is preferred. An empty cell indicates no assignment has been made.
Pin	The device's pin number for the device to which this signal is assigned. This field can also be used to make a pin assignment by selecting the desired pin number from the drop-down list. An asterisk (*) indicates that a particular assignment is preferred. An empty cell indicates no assignment has been made.
Lock	Specifies whether or not the signal's assignment is locked (i.e., cannot be moved by a build). This cell is only editable for assigned pins.

Signals can be displayed in one of the following states:

- Assigned and Locked – A signal that has been assigned and locked to a particular pin is displayed in the table as a dark blue row.
- Assigned and Unlocked – A signal that has been assigned but not locked is displayed in the table as a light blue row.
- Unassigned – Signals that have not been assigned are white. This will be auto-assigned by PSoC Creator on the next build.

Note Double-clicking a row in the table will display the design containing the associated Pins Component, and open the Configure dialog for it.

To Assign a Pin:

Assign a pin using either of the following methods:

- Click on a signal in the Signal Table or on a pin already assigned elsewhere on the device and drag it to the desired location on the device image.

Note While dragging a pin, a tooltip will indicate whether or not the current location is valid in addition to the coloring noted above.

- Select an assignment from the **Pin** column pull-down menu in the Signal Table. You can also type the pin assignment in the field. While typing, legal assignments that are still possible based on what has currently been typed will display. Values are entered in the form of:
 - P#[#] – Specifies a location where the first # is the port number and the second # is the offset within the port.
 - P#[#:#] – Specifies a range of locations where the first # is the port number, the second number is the offset within the port where the MSB of the signal should be placed, and the last # is the offset within the port where the LSB of the signal should be placed.
 - A range can also be specified as any combination of the above two formats separated by commas.

To Unassign a Pin:

Unassign a pin using either of the following methods:

- Right-click on an assigned pin on the device image and select **Auto-assign <signal> during build**.
- Select the **<Auto-assign during build>** row from the **Port or Pin** column pull-down menu in the Signal Table.

To Unassign All Pins:

Right-click anywhere in the device image section of the Pin Editor and select **Auto-assign all during build**.

To Lock a Pin:

Lock a pin using either of the following methods:

- Manually assign a pin; it will be locked by default.
- If a pin is assigned but not locked, right-click on the pin in the device image and select **Lock <signal>** or select the **Lock** check box in the table for the desired signal.

On the right-click menu, the **Lock All** option will lock all assigned pins.

To Unlock a Pin:

Right click on the pin in the device image and select **Unlock <signal>** or de-select the **Lock** check box in the table for the desired signal.

On the right-click menu, the **Unlock All** option will unlock all locked pins.

To Scroll, Pan, and Zoom:

If device image is too large, scroll bars will display to allow you to see other areas of the device. Use these techniques as appropriate:

- To auto-scroll, drag a signal to the edge of the device image.
- Use the mouse wheel to scroll up and down; press [**Shift**] + mouse wheel to scroll left and right.
- To pan, press [**Alt**] + left click and drag.
- To zoom, press [**Ctrl**] + mouse wheel.

See Also:

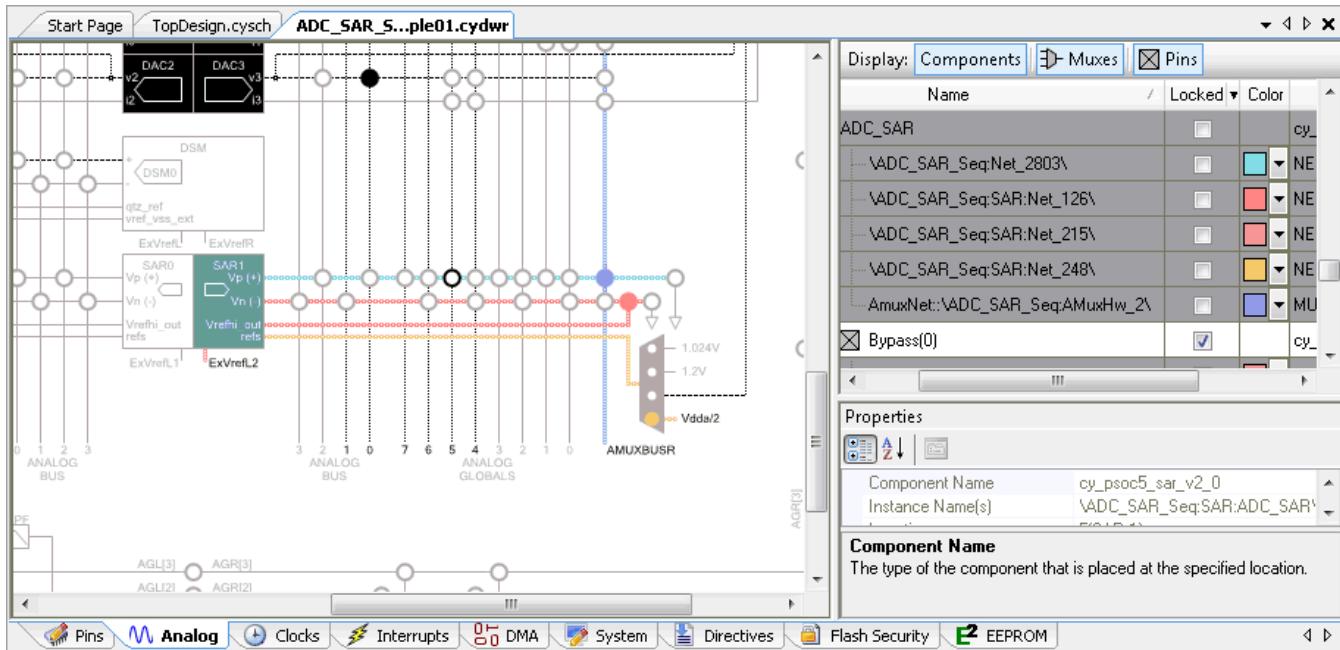
- [Mapper, Placer, Router](#)
- [Design-Wide Resources](#)

Analog Device Editor

The Analog Device Editor provides an interconnect view of the PSoC device along with place-and-route results for a particular design. The editor also allows for manual place-and-route with the ability to lock-down all or some of the results.

It operates in two separate modes:

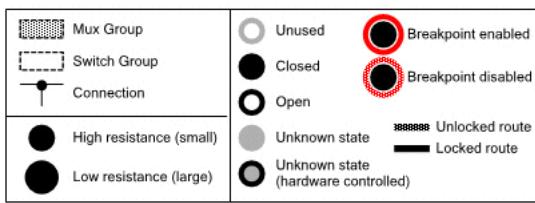
- **Design** – shows the results of a build in design mode (this is default mode of the tool)
- **Debug** – provides a view into the current state of the device while debugging (see [Analog Device Editor Debugging](#))



The Analog Device Editor contains three major sections: analog interconnect diagram, design information table, and properties.

Analog Interconnect Diagram:

The analog interconnect diagram shows how various analog Components, wires, switches, and pins are laid out on the device and how they may be connected. Unused resources are shown as gray. Used resources are shown as black if they are not selected. If a used Component or pin is selected, it is shown as blue. Selected routes and switches are shown as the color chosen in the table. The following image shows the Legend used for the interconnect diagram.



If you hover the cursor over used resources, tooltips display relevant information. This same information will be displayed in the **Properties** area when the resource is selected.

Wires

Wires can be locked (cannot move between builds). This includes resources locked by MARS Components and control files (even though the editor can over-ride those placements). Locked wires are solid lines; unlocked wires are dashed lines.

Wires locked in the Analog Device Editor can also be unlocked. Wires locked by other sources must be unlocked by those sources.

Switches

Switches are displayed as circles. A solid color means the switch is closed. White means the switch is open. Gray means the state is not known.

- In design mode, the check box selections in **Properties** for muxes control the display of run-time changeable switches.
- In debug mode, the state is based on the value of the actual register and can be edited from the GUI. If the mux that “owns” the switch is hardware-controlled it is displayed in gray when debugging.

Note: DMA access to a software-controlled AMux will potentially cause the debugger to show erroneous states.

Switch breakpoints are shown by a dotted/dashed line around the switch. If the breakpoint is enabled the line is red; if disabled, it is amber.

Switches are grouped within the device and this shall be shown by a dotted box around the circles. These groups allow either at-most-one active terminal or any number of active terminals. The latter shall be distinguished in the diagram with a gray fill in the surrounding box.

Pins and Components

Locked pins and Components include a small pad-lock icon.

Digital pins used in the design are shown with a black background, white text, and teal “tips”. However, no routing information is shown for digital pins.

Design Information Table:

The columns have the following headings.

- **Name** refers to the instance name.
- **Lock** is a check box coercing the item to use the associated resource. It is possible to lock or unlock all listed items in the **Lock** column using the pull-down menu, as follows:



- **Lock All** – Lock all routes in the design.
- **Unlock All** – Unlock all routes in the design.
- **Lock selection** – Locks/unlocks the selected route. Locked routes are shown as solid lines; unlocked routes are shown as dashed lines.
- **Unlock selection** – Marks the selected wire as unused. See [Route Editing](#).
- **Cleanup...** – Opens the [Locked Route Cleanup](#) dialog to remove locked net information that is no longer applicable to the current design.

- **Color** allows the user to choose a display color in the Interconnect panel, from a pull-down, for a routing resource. Note that it is common for resources to be displayed multiple times (for example, when a net connects a pin to an analog resource in the top schematic), and so changing the color in one place requires a (silent) change everywhere else.
- **Type** is the Component name (e.g. “PGA_v1_70”) or resource type (i.e. “MUX” or “NET”).

The entries in the table are listed alphabetical order by instance name and only that column may be used to change the order of entries.

Each entry (Components, pins and muxes) is expandable/collapsible to show lower levels of the Component hierarchy and the resources to which they are connected.

There are buttons at the top of the table to toggle on and off viewing selected items: **Components**, **Muxes**, and **Pins**.

Properties:

The **Properties** area displays information based on the selection in the table or diagram. If you select multiple items, the area will not show any information. The area displays differently if you select pins/Components versus muxes.

Component/Pins View

For Components and pins, this area displays various read-only properties depending on what is selected. The properties that can be displayed include the following:

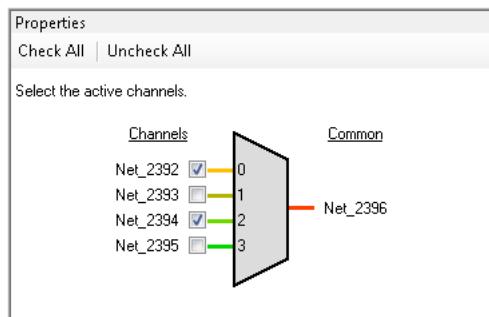
- Net name (for nets only)
 - For a named a wire in the schematic, the tool displays that name. When the schematic wire is un-named, it displays a machine-generated net name.
- Drive mode (for pins only)

- Component name
- Resource name
- Lock status

Use the buttons at the top of the panel to order the properties alphabetically or by category.

Mux View

For muxes, the **Properties** area displays an editable image of the selected mux to choose the active channels in the interconnect diagram and [Ohm Meter](#). If the target is a differential mux then making a channel active always applies to both connections.



The diagram includes check boxes to choose the active channel(s) of the mux. It also shows the net names associated with each channel and the common terminal. The coloring for each net will be the same as that chosen in the table. When you select and de-select a channel, the change is reflected in the interconnect diagram and the Ohm meter.

Use the **Check All** and **Uncheck All** buttons at the top of the panel to select all or de-select all channels, respectively.

Note If the **AtMostOneActive** parameter is set to "true," then you can only select one channel and the **Check All** button will be disabled.

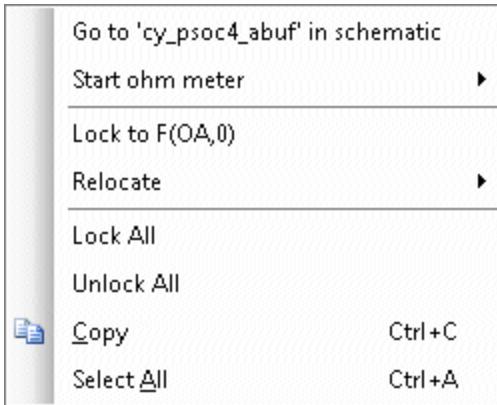
See Also:

- [Analog Device Editor Context Menus](#)
- [Ohm Meter](#)
- [Manual Placement](#)
- [Route Editing](#)
- [Analog Device Editor Debugging](#)

Analog Device Editor Context Menus

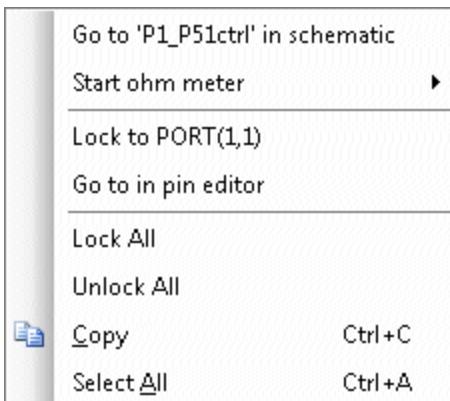
The [Analog Device Editor](#) contains various commands available on right-click – or context – menus. The commands available will vary depending on whether you right-click on a pin, Component, or signal. You can right-click on items in the interconnect diagram and in the table. The following are the commands available:

On Component:

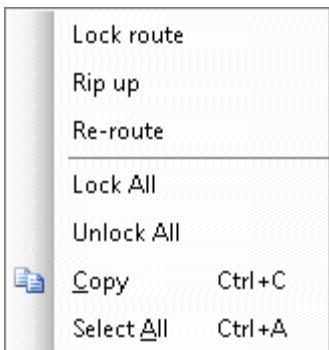


- **Go to <Component> in schematic** – Opens the selected Component in the [Schematic Editor](#).
- **Start Ohm meter** – Opens the [Ohm Meter](#) to the selected signal.
- **Lock to/Unlock From <Location>** – Toggles lock/unlock for this Component to/from the existing location.
- **Relocate** – Allows you to move the selected primitive to another location. This option is disable if no other valid locations are available. See also [Manual Placement](#).
- **Lock All** – Lock all routes in the design.
- **Unlock All** – Unlock all routes in the design.
- **Copy** – Copies the interconnect diagram to a bitmap file that you can paste in an appropriate editor.
- **Select All** – Selects everything on the interconnect diagram.

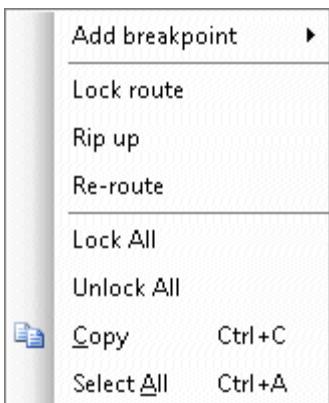
On Pin:



- **Go to <Pin> in schematic** – Opens the selected pin in the [Schematic Editor](#).
- **Start Ohm meter** – Opens the [Ohm Meter](#) to the selected pin.
- **Lock to/Unlock from <Location>** – Locks/unlocks this pin to/from the existing location.
- **Go to in pin editor** – Opens the [Pin Editor](#) and selects the same pin.
- **Lock All** – Lock all routes in the design.
- **Unlock All** – Unlock all routes in the design.
- **Copy** – Copies the interconnect diagram to a bitmap file that you can paste in an appropriate editor.
- **Select All** – Selects everything on the interconnect diagram.

On Wire:

- **Lock route** – Locks/unlocks the selected route. Locked routes are shown as solid lines; unlocked routes are shown as dashed lines.
- **Rip up** – Marks the selected wire as unused. See [Route Editing](#).
- **Re-route** – Switches the Analog Device Editor to Manual Routing mode. See [Route Editing](#).
- **Remove obsolete "Entire Net" route data** – Removes nets that were locked as part of an entire net when you change the schematic.
- **Lock All** – Lock all routes in the design.
- **Unlock All** – Unlock all routes in the design.
- **Copy** – Copies the interconnect diagram to a bitmap file that you can paste in an appropriate editor.
- **Select All** – Selects everything on the interconnect diagram.

On Switch:

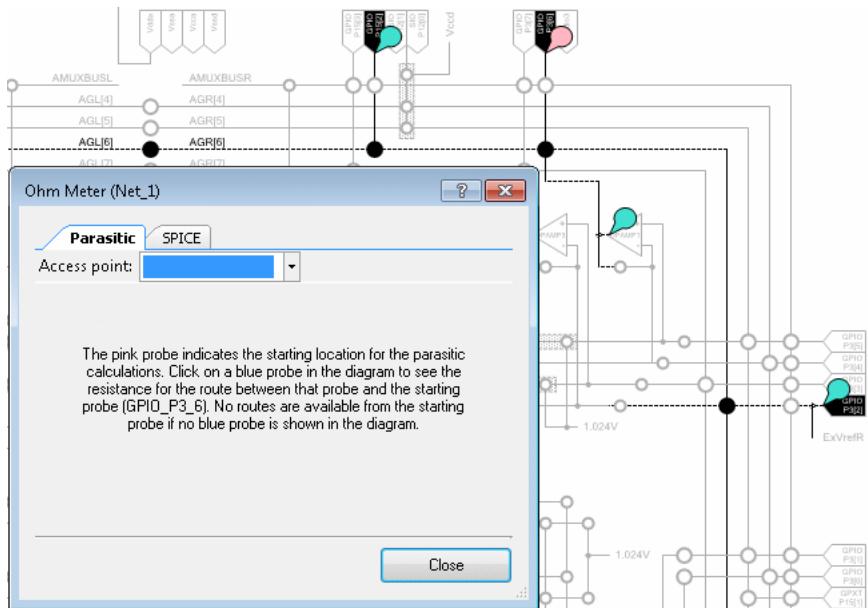
- **Add/Edit breakpoint** – Used to add or edit breakpoints. See [Analog Device Editor Debugging](#).
- **Lock route** – Locks/unlocks the selected route. Locked routes are shown as solid lines; unlocked routes are shown as dashed lines.
- **Rip up** – Marks the selected wire as unused. See [Route Editing](#).
- **Re-route** – Switches the Analog Device Editor to Manual Routing mode. See [Route Editing](#).
- **Lock All** – Lock all routes in the design.
- **Unlock All** – Unlock all routes in the design.
- **Copy** – Copies the interconnect diagram to a bitmap file that you can paste in an appropriate editor.
- **Select All** – Selects everything on the interconnect diagram.

See Also:

- [Analog Device Editor](#)
- [Schematic Editor](#)
- [Pin Editor](#)
- [Ohm Meter](#)
- [Manual Placement](#)
- [Route Editing](#)
- [Analog Device Editor Debugging](#)

Ohm Meter

The Ohm Meter displays the resistance between pairs of points (pins and/or Components but not wires or switches) in the [Analog Device Editor](#) interconnect view. The dialog allows you to change probe points in the diagram and see the parasitics. The dialog also contains SPICE route data.



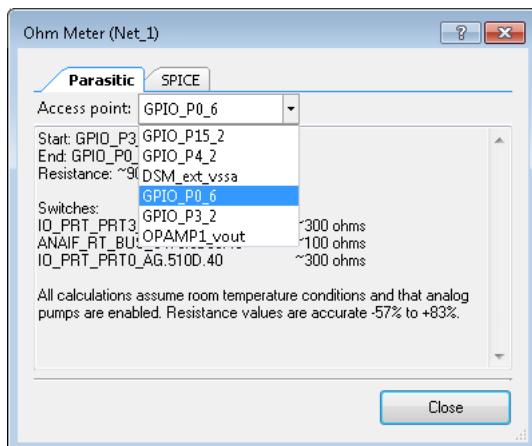
Probes are displayed in the interconnect diagram with an attached pin. The first-placed pin is pink and, when it is placed, all legal destination probe points are automatically marked with blue probes. Select a blue pin to see the resistance for that route. Re-selecting the start-point for probing clears previous probes, clears the dialog results, and refreshes the legal destination (blue) probes.

To Open the Ohm Meter:

Right-click on a pin or Component in the diagram or the table and select **Start Ohm Meter >**. Then point to the access point from which to start.

Parasitic Tab:

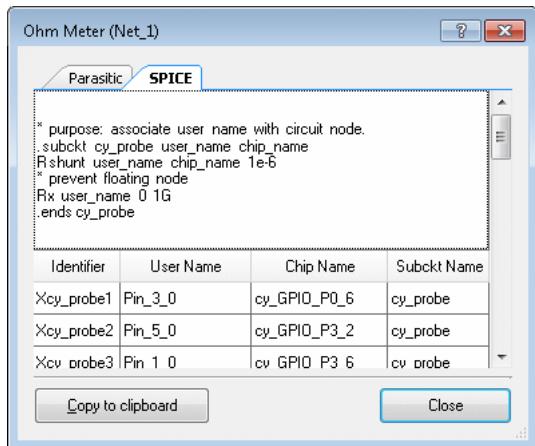
This tab displays the resistance between the two points when you click a blue pin in the diagram. This includes the total resistance and the resistance of every switch in the route.



The **Access Point** pull-down menu allows you to change the second selected pin as an alternative to clicking on different blue pins.

SPICE Tab:

This tab shows a SPICE netlist for the route. This is read-only text.



Use the **Copy to Clipboard** button to copy the data to a simulator of your choice.

See Also:

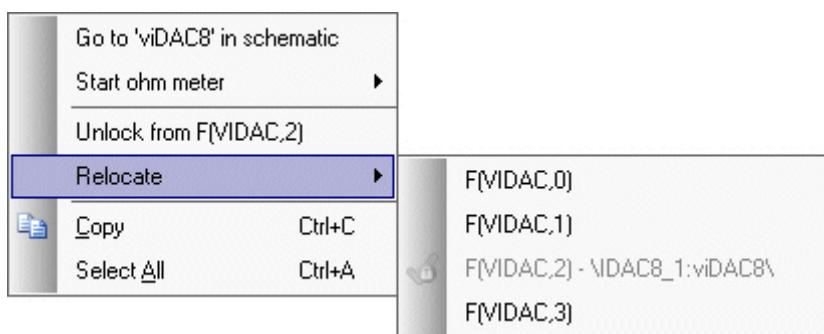
- [Analog Device Editor](#)

Manual Placement

The [Analog Device Editor](#) manual placement feature allows you to specify where analog Components should be placed when a build is performed. You cannot use this feature to place pins. Instead you must use the [Pin Editor](#).

To Manually Place a Component:

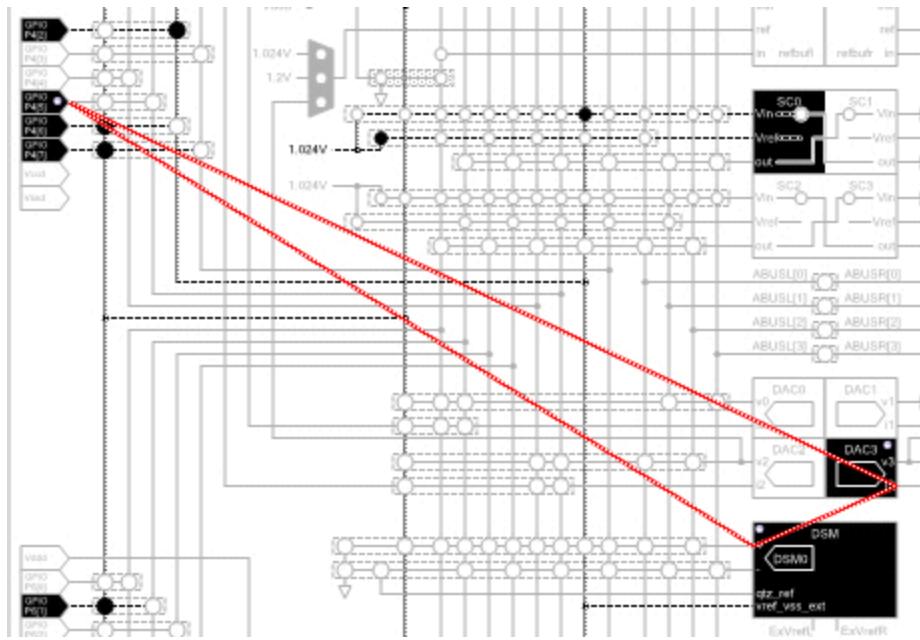
Right-click on a Component and select **Relocate >**. Then point to the desired location. The menu will only allow valid moves. If no move is available, the menu item will be disabled.



You may move a Component to a currently occupied location. The existing Component will be placed in a new location by the subsequent build.

Signal Routing

After selecting the new location, the routed signals will be redrawn in a temporary state, also known as a rat's nest. This is only shown if the route is selected.



During a build, the routing will updated to follow allocated signals.

Resource Locking

All Component moves result in the resource being locked. The Component will be shown with a lock symbol. 

To unlock the resource, right-click on the Component and select **Unlock from <location>**.

If you unlock the resource before doing a build, the unlock action will move the Component back to its original location.

If you unlock the resource after doing a build, the icon will remain until you do another build, because the editor interprets the lock being in place from an external source.

See Also:

- [Analog Device Editor](#)
- [Pin Editor](#)
- [Context Menus](#)
- [Route Editing](#)

Route Editing

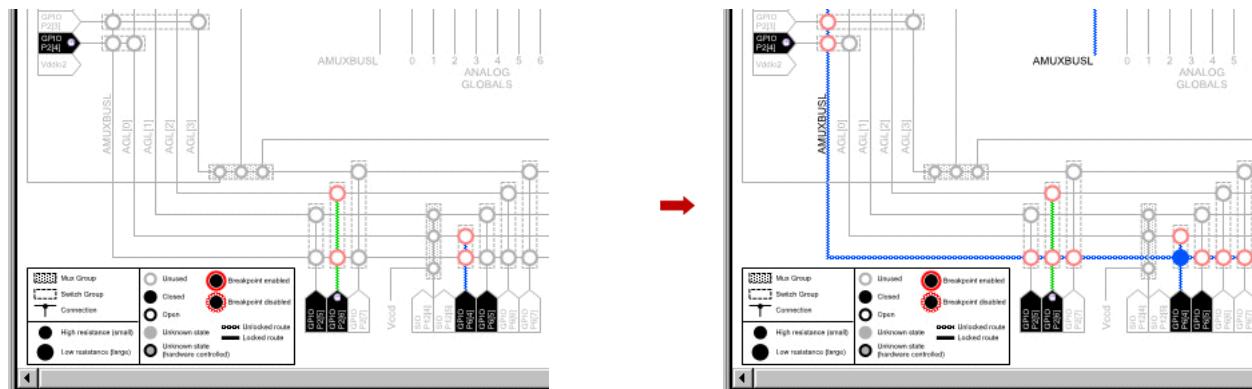
Route editing consists of ripping up routes and re-routing signals, as well as manually selecting switches to route signals. [Context Menus](#) are included on various elements to facilitate route editing.

Rip-Up

- **Nets** – When a net is ripped up, the wires and switches used to route the net are marked as unused
- **Net-Ties/Net-Joins** – When a Net-Tie or Net-Join is selected, you can rip up the entire net or any of the sub-nets collected to form the entire net. Ripping up any of the sub-nets is the same as ripping up a net. Ripping up the entire net causes all of the individual sub-nets to be ripped up, as well as the additional resources used to implement the net-tie and net-join Components.
- **Muxes** – Rip-up of a mux is applied to the entire mux. When a mux is ripped up, all of the wires and switches used to route the mux are marked as unused.

Re-Route

When you select the **Re-Route** command, the [Analog Device Editor](#) enters manual route editing mode. In this mode, you can manually route signals by clicking on switches to open or close them. Switches available for routing are shown highlighted in the default color; switches not available are grey. When you select a switch to route a signal, addition switches become available to select.



Open switches are shown as hollow circles; closed switches are shown as solid circles.

While editing, the Status Bar shows the operation in progress.

Two buttons are available: **Commit Edit** or **Cancel Edit**. "Commit" saves and locks the routing; "Cancel" reverts the Analog Device Editor the state prior to editing.

Mux Routes

Editing a mux arm is similar to editing a signal net with the following differences:

- The endpoints to be connected are nets rather than pins.
- When you start editing a mux, all of the resources used by the mux (for all arms) are temporarily marked as unused to allow the user maximum flexibility in reusing resources shared with other arms.

Net-Ties and Net-Joins

Editing net-ties and net-joins are similar to editing a mux. However instead of joining a single pair of nets, editing these may involve any number of sub-nets.

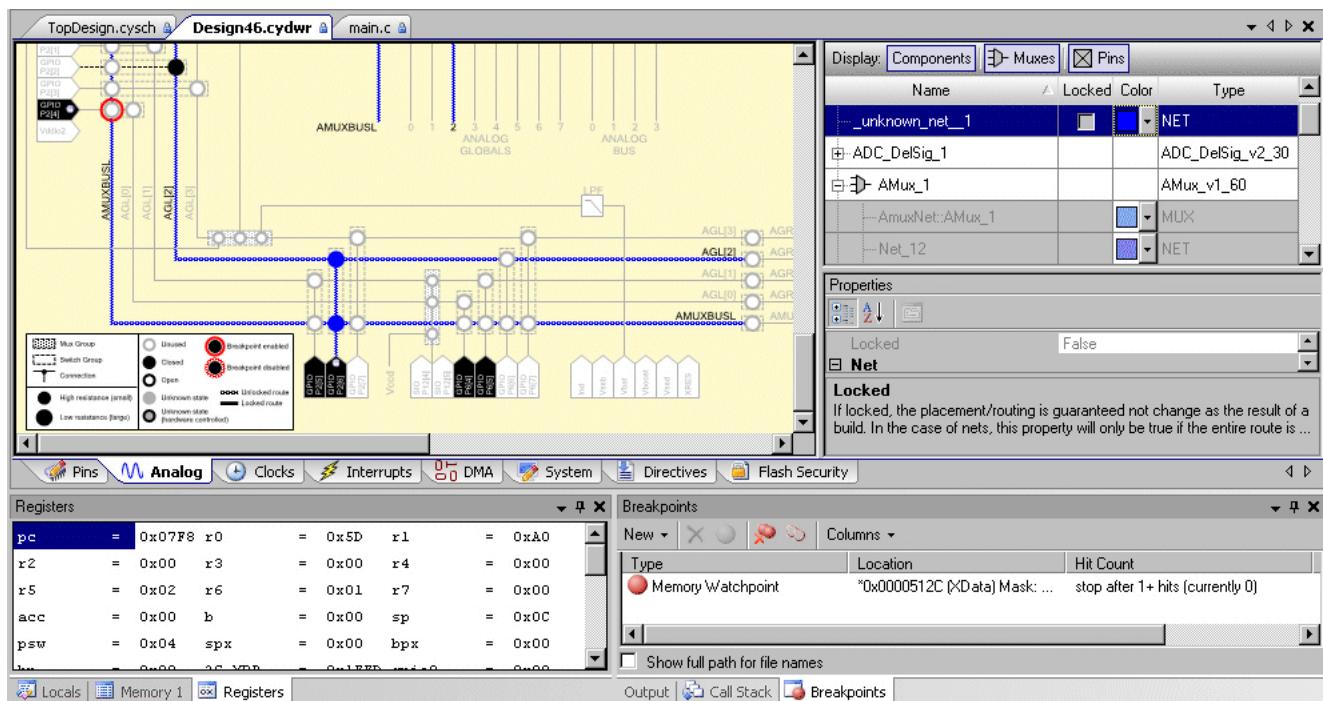
- The endpoints to be connected are nets rather than pins (like muxes).
- There are separate rip-up and edit operations since there is no resource sharing.

See Also:

- [Analog Device Editor](#)
- [Analog Device Editor Context Menus](#)

Analog Device Editor Debugging

The debug view in the [Analog Device Editor](#) allows users to view/modify the live state of the analog routing on the chip. The debug view will automatically become active when you start [debugging](#). While in debug mode, the interconnect diagram background is yellow, and the label at the bottom right indicates that it is in debug mode.



The Analog Device Editor will automatically go back to the design view when the debug session has stopped.

Open/Close Switches:

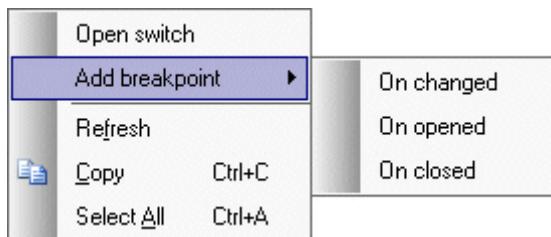
In debug view, you can use the [context menu](#) to open/close any switch that is not DSI controlled. Switches that are DSI controlled will be displayed with a grey center to indicate that they are “used;” however, PSoC Creator does not know the current state of the switch.

Note Because the state of the Analog Device Editor can be changed without the CPU running, it may be out of date. Right-click on the interconnect diagram and select **Refresh** to update all visible debug windows.

Switch Breakpoints:

Breakpoints are implemented with the on-chip address breakpoint(s). The breakpoint applies to the whole register, which controls a number of switches and unrelated elements. The debugger determines whether the break occurred as a result of the switch-of-interest and ignores (continues execution) other changes.

To set a breakpoint, right-click on a switch. You can set the following conditions: on changed, on opened, on closed.



Once selected, the breakpoint displays in the [Memory Watchpoint](#) window.

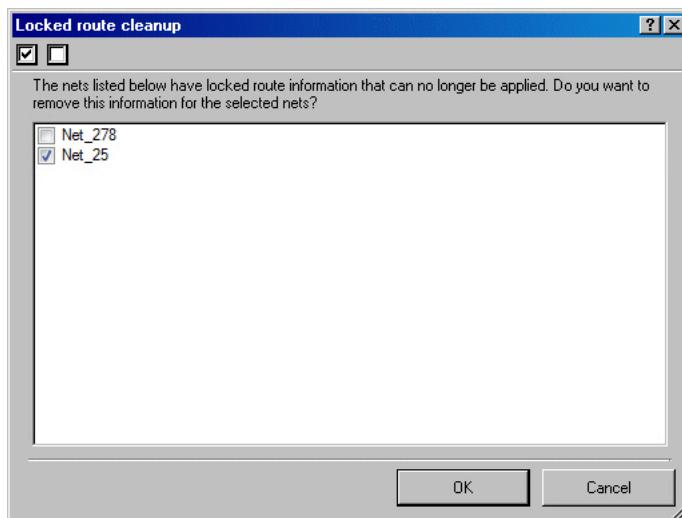
You can set multiple breakpoints in a signal register with a single breakpoint resource. If the maximum number of breakpoints has been reached (regardless of the use) and you try to set a new one, PSoC Creator will indicate that the break cannot be set until another breakpoint is cleared or disabled.

See Also:

- [Analog Device Editor](#)
- [Analog Device Editor Context Menus](#)
- [Memory Watchpoint](#)

Locked Route Cleanup

The Locked Route Cleanup dialog that lets you clean up locked route information for nets where either the net no longer exists in the schematic, or is no longer locked due to schematic changes. You can selectively delete any of this data without unlocking anything else.



To Open this Dialog:

Select **Cleanup...** from the **Locked** pull-down menu in the [Analog Device Editor](#) table.

To Use this Dialog:

Select one or more nets for which to remove locked route information, and click **OK**.

You can also use the **Select All** or **Deselect All** buttons.

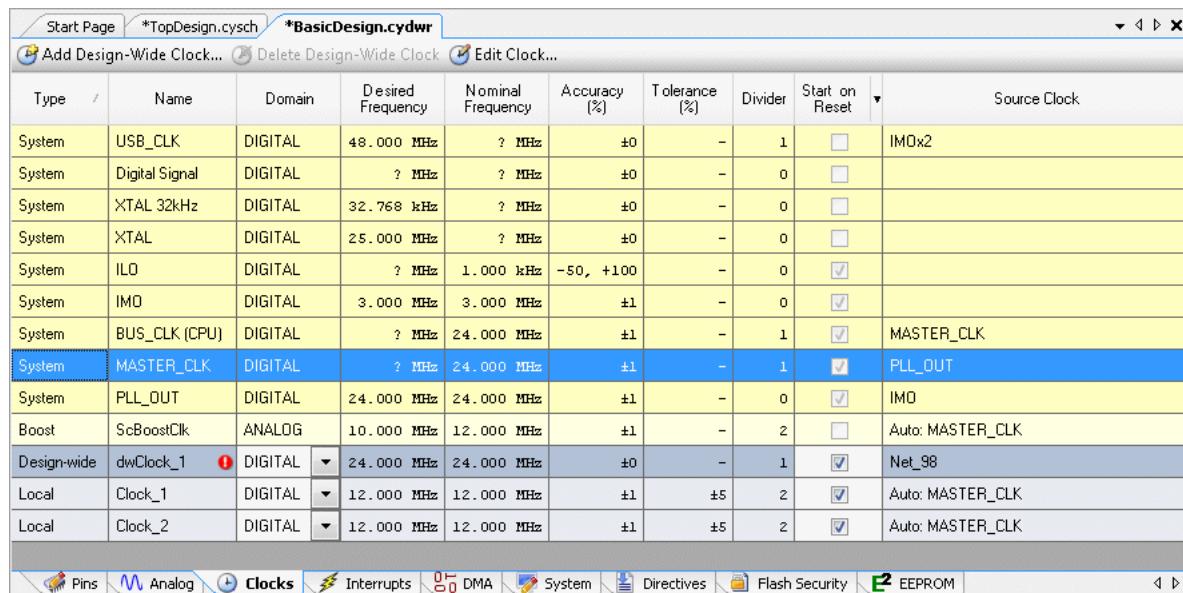
See Also:

- [Analog Device Editor](#)

Clock Editor

The Clock Editor is a design-wide resources tool to create and edit clocks. This tool allows you to view all clocks, add and delete design-wide clocks, as well as edit design-wide and system clocks.

Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.



Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	Source Clock
System	USB_CLK	DIGITAL	48.000 MHz	? MHz	±0	-	1	<input type="checkbox"/>	IMOx2
System	Digital Signal	DIGITAL	? MHz	? MHz	±0	-	0	<input type="checkbox"/>	
System	XTAL 32kHz	DIGITAL	32.768 kHz	? MHz	±0	-	0	<input type="checkbox"/>	
System	XTAL	DIGITAL	25.000 MHz	? MHz	±0	-	0	<input type="checkbox"/>	
System	ILO	DIGITAL	? MHz	1.000 kHz	-50, +100	-	0	<input checked="" type="checkbox"/>	
System	IMO	DIGITAL	3.000 MHz	3.000 MHz	±1	-	0	<input checked="" type="checkbox"/>	
System	BUS_CLK (CPU)	DIGITAL	? MHz	24.000 MHz	±1	-	1	<input checked="" type="checkbox"/>	MASTER_CLK
System	MASTER_CLK	DIGITAL	? MHz	24.000 MHz	±1	-	1	<input checked="" type="checkbox"/>	PLL_OUT
System	PLL_OUT	DIGITAL	24.000 MHz	24.000 MHz	±1	-	0	<input checked="" type="checkbox"/>	IMO
Boost	ScBoostClk	ANALOG	10.000 MHz	12.000 MHz	±1	-	2	<input type="checkbox"/>	Auto: MASTER_CLK
Design-wide	dwClock_1	DIGITAL	24.000 MHz	24.000 MHz	±0	-	1	<input checked="" type="checkbox"/>	Net_98
Local	Clock_1	DIGITAL	12.000 MHz	12.000 MHz	±1	±5	2	<input checked="" type="checkbox"/>	Auto: MASTER_CLK
Local	Clock_2	DIGITAL	12.000 MHz	12.000 MHz	±1	±5	2	<input checked="" type="checkbox"/>	Auto: MASTER_CLK

Toolbar icons: Pins, Analog, Clocks (highlighted), Interrupts, DMA, System, Directives, Flash Security, EEPROM.

To Open the Clock Editor:

Double click the Clocks icon in the Design-Wide Resources tree, located in the **Source** tab of [Workspace Explorer](#).

The `<project>.cydwr` file opens as a [tabbed document](#) in the work area, with the Clock Editor (**Clocks** tab) displayed on top.

Clock Editor Toolbar:

The toolbar contains the following commands:

- **Add Design-Wide Clock** – This command allows you to add a design-wide clock to your design. See [Add/Edit Design-Wide Clock](#).
- **Delete Design-Wide Clock** – This command allows you to delete any design-wide clocks you may have added.

- **Edit Clock** – This allows you to edit design-wide clocks in your design. See [Add/Edit Design-Wide Clock](#).

Clock Table:

The Clock Editor displays all the clocks using a table. By default, clocks are sorted by **Type**. You can sort by any column by clicking the column header and switch between ascending and descending order.

Note A secondary sort is always performed on the **Nominal Frequency** column.

Column	Description
Type	<p>The clock type:</p> <ul style="list-style-type: none"> • System – internal and external clocks sources that can be used in your design • Design-wide – clocks declared in the clock editor that are sharable across the entire design • Local – clocks added to schematics via Components • Boost – low voltage analog boost clock shown when the Variable Vdda option is selected in the System Editor.
Name	<p>The name of the clock.</p> <p>If a clock is in an error state, an icon will display by the clock's name. Hovering over the icon will display a tooltip with a message describing the error. This errors will also be displayed in the Notice List Window.</p>
Domain	Analog or Digital. This information is determined by PSoC Creator.
Desired Frequency	The desired frequency for the clock. If not used '? MHz' will be displayed.
Nominal Frequency	The nominal frequency for the clock. This is determined by the system solving the clocks. If not solvable '? MHz' will be displayed.
Accuracy	Displays the clock's accuracy as a percent.
Tolerance	Displays the tolerance range entered for the clock as a percent. If no tolerance has been specified '-' will be displayed. Note Tolerance can only be specified for Auto clocks.
Divider	Displays the divider used for the clock. This may have been specified elsewhere or calculated when the clocks were solved.
Start on Reset	<p>If checked, this option will cause the <code>_Start()</code> function to be called for the clock pre-main (checked by default). You can set this option for all 'New' local clocks and all design-wide clocks.</p> <p>There is a pull-down menu next to the column header, which allows you to check or uncheck all the applicable boxes at once.</p> <p>System clocks show this box as read-only. For them the value is determined by their enabled state in the Configure System Clocks dialog. 'Existing' clocks also have this field as read-only. It displays the 'Start on Reset' value for its source clock in this case.</p>
Source Clock	The clock, if any, used to create the clock. Clocks whose input clock was not explicitly specified (i.e., <Auto> was selected) will be displayed as "Auto: Clock the solver picked."

Digital and Analog Clocks:

PSoC Creator automatically figures out whether a user-created clock is digital or analog. It looks across the whole design at the fanout and if the clock drives analog primitives, it is an analog clock. If the clock does not connect to an analog primitive, it is assumed to be digital. If a clock connects to a mixture of digital and analog primitives, a DRC error is generated.

Tolerance Support:

PSoC Creator will support a robust system of clock accuracy and tolerance related DRCs.

Local and design-wide clocks have the ability to add required tolerances (+X%, -Y% or +X ppm, -Y ppm).

System clocks internal to the device display their accuracy information. System clocks that come from outside the device (e.g., XTAL, XTAL 32kHz, and Dig Sig) will allow you to enter accuracy statistics (+X%, -Y% or +X ppm, -Y ppm). The default accuracy values will be +0%, -0%.

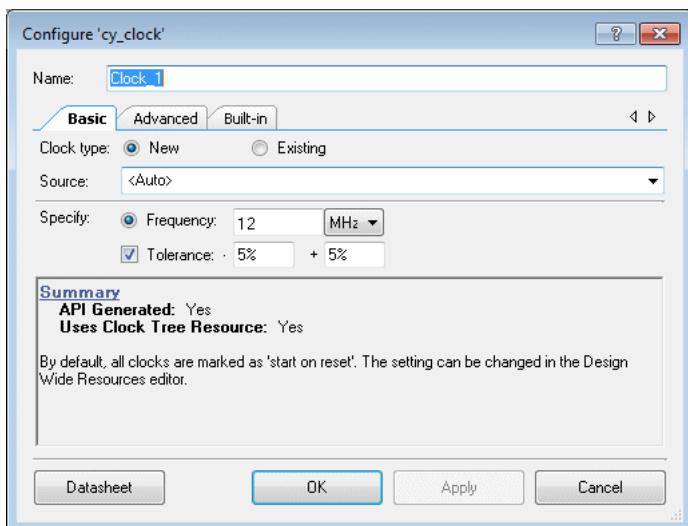
The PSoC Creator clock system will take the accuracy of the reference clock, the actual achieved frequency, and compare that against the tolerance and requested frequency. If the achieved frequency/accuracy falls outside the requested bounds, a DRC error will be generated.

See Also:

- [Configure Local Clock](#)
- [Configure System Clocks](#)
- [Add/Edit Design-Wide Clock](#)
- [Select Source Clock](#)
- Clock Component Datasheet (available from the [Component Catalog](#))
- [Design-Wide Resources](#)

Configure Local Clock

The Configure clock dialog allows you to configure various characteristics for a local clock that you placed onto a schematic. Refer also to the Clock Component datasheet available from the [Component Catalog](#).



This dialog has the following main sections:

- **Clock Name** – Allows you to type a name for the local clock.
- **Configuration** – Allows you to specify characteristics for the clock.

- **Summary** – Displays information about the clock being created and the source clock being used (if **Source** is not <Auto>).

To Create a Local Clock:

Create a local clock by dragging a clock Component from the [Component Catalog](#) onto a schematic.

To Open the Dialog:

Double-click the clock Component to open the Configure Clock dialog.

To Configure a Local Clock:

You can configure the local clock as follows:

1. In **Name**, type a name for the clock or accept the default name.
2. For **Clock Type**, select **New** or **Existing**. New clocks use device resources and have APIs generated for them. Existing clocks do not use hardware resources and do not have APIs generated for them; they are simply an alias to a clock already defined.
3. Specify the **Source** configuration, as follows:
 - **Clock Type: New / Source: <Auto>** – Enter the **Frequency** and optionally a **Tolerance** range. PSoC Creator figures how to implement it.
 - **Clock Type: New / Source: Specified** – Select a particular source clock from the pull down menu to divide down. Then enter a desired frequency or explicit divider.
 - If you specify a **Desired Frequency**, PSoC Creator calculates the divider automatically.
 - If you specify a **Divider**, PSoC Creator uses it as specified.
 - **Clock Type: Existing / Source: Specified** – Create an alias for the given source clock.

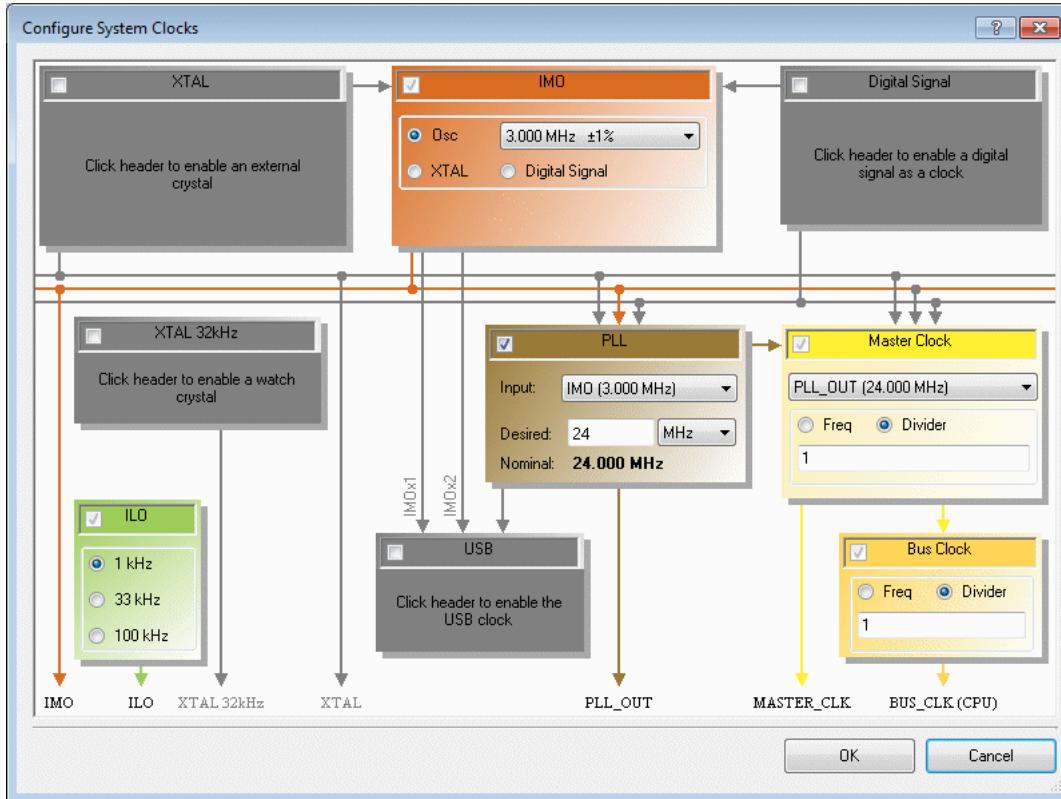
You can view the clock characteristics for various clocks using the design-wide resources [Clock Editor](#).

See Also:

- [Design-Wide Resources](#)
- [Clock Editor](#)
- [Component Catalog](#)
- Clock Component Datasheet (open from the [Component Catalog](#))

Configure System Clocks

The Configure System Clocks dialog provides a graphical diagram showing the various system clocks in the selected device for your design, as well as their relationship to each other.



This dialog allows you to specify different characteristics about the system clocks.

- A check mark indicates that the clock is enabled. When a clock is disabled it turns gray and its contents are replaced by text explaining how to enable it. Some clocks cannot be disabled.
- The lines show where clock signals can go. If a line is gray, then the current clock configuration does not use that path.
- The error icon  indicates that the clock is configured incorrectly. Hovering over the icon will display a message indicating exactly what is wrong.

Note This dialog will contain different clocks and options depending on the device selected for the design. For some devices, this dialog will be split into different tabs for high-frequency and low-frequency clocks.

For specific details about the configuration options for system clocks, refer to the appropriate *Technical Reference Manual*.

Digital Signal:

The Digital Signal can be configured to use any routed digital signal in your design. When using the output from a digital pin as the Digital Signal, you will need to ensure that the pin is configured with the input as unsynchronized. This is done via the **Input** tab in the Pins Component Configure dialog by deselecting the "Input Synchronized" option. For more information, refer to the Pins Component datasheet.

To Open this Dialog:

In the Design-Wide Resources [Clock Editor](#):

- Double-click a system clock, or
- Select a system clock and click **Edit Clock**.

To Enable/Disable a System Clock:

Click the check box for the appropriate clock to enable or disable.

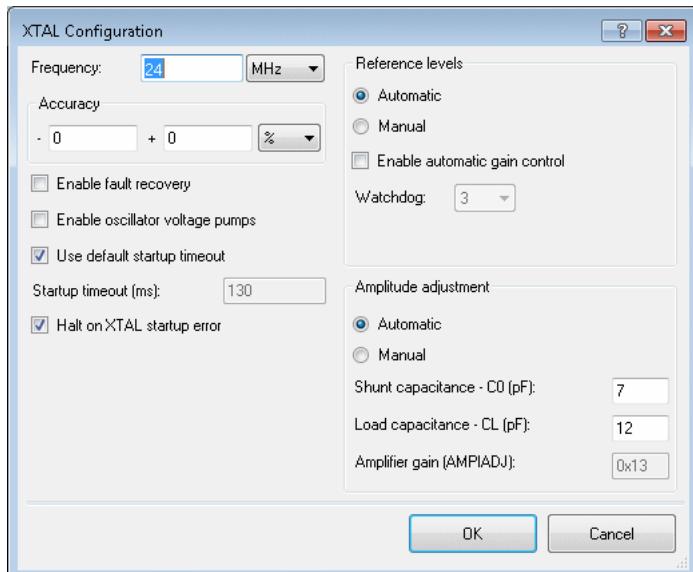
Note Some clocks cannot be disabled. Their check boxes are dimmed.

See Also:

- [Design-Wide Resources](#)
- [Clock Editor](#)
- Clock Component Datasheet (open from the [Component Catalog](#))
- Pins Component Datasheet (open from the [Component Catalog](#))

XTAL Configuration

The XTAL Configuration dialog is used to configure characteristics for the external crystal.



To Open the Dialog:

Enable the XTAL on the [System Clock Editor](#), and click the **Configure** button.

To Configure the XTAL:

Select/enter the appropriate fields and click **OK**.

Fields:

The following fields are used to configure the XTAL as needed.

Frequency/Accuracy:

Use these fields to specify the frequency and accuracy of the XTAL.

Enable fault recovery:

This check box enables fault recovery. Fault recovery allows the output of the XTAL to be switched from the XTAL clock to the IMO if the XTAL is detected to have stopped working properly. It uses the watchdog/error signal inside the XTAL.

Note This field produces an error: "The XTAL clock cannot use fault recovery when it is also used by the IMO." when it is set to true and the XTAL sources the IMO.

Enable oscillator voltage pumps:

This check box controls the oscillator voltage pumps, via bit 3 (xpump_dis) of the FASTCLK_XMHZ_CSR register. This can be used to save power and reduce jitter. The check box is not selected by default, meaning this feature is not enabled.

Use default timeout:

This check box is selected by default. It indicates whether the system will use the default timeout at startup, or a user-customized one.

- **Startup timeout (ms)** – When using the default timeout, this field is read only and displays the default timeout value. If you deselect the **Use default timeout** check box, this field becomes enabled to specify a timeout based on specific XTAL requirements.

Halt on XTAL startup error:

This check box enables the “while(1);” code in the ECO error handler. It is enabled by default, and the application halts before main(). If you deselect this check box, the code will continue to main (and you are expected to handle the clock error appropriately).

Note This field is ignored when the XTAL does not source the bus clock. No message will be shown if this occurs.

Reference levels:

- **Automatic** – This option is selected by default. **Reference levels** are calculated from the given XTAL frequency. The XTAL clock initialization uses these values to set the CFG1 register.
- **Manual** – This option is used by advanced users. This enables the **Feedback** and **Watchdog** fields to fine tune the reference level configuration. The XTAL clock initialization uses these values in the CFG1 register.
 - **Enable automatic gain control** – This check box enables automatic gain control (AGC). AGC measures oscillation amplitude and compares it to a reference value. If it is too high or low, an internal

adjustment is made in the XTAL to increase or decrease amplitude. This reduces drive level and helps to meet crystal requirements. It is not needed in all cases. This check box is not selected by default. When it is selected, the **Feedback** field becomes visible under **Watchdog**.

- **Watchdog** – The watchdog (XERR/error detection) is a circuit that measures oscillation amplitude and compares it to a reference value. If it is too low, an error signal that goes to a status register bit is asserted. This bit can be polled in software, and also controls a mux that implements "fault recovery." The watchdog is used at XTAL startup to determine when oscillations have reached an acceptable amplitude, and the XTAL can be used as a clock source throughout the part. The watchdog reference level is the voltage to which the oscillation amplitude is compared when determining if oscillation is acceptable.
- **Feedback** – The feedback reference level is the voltage of which the oscillation amplitude is compared.

Amplitude adjustment:

- **Automatic** – This option is selected by default. The **Amplifier Gain (AMPIADJ)** field is calculated from the given XTAL frequency. This option allows you to specify **Shunt capacitance** and **Load capacitance** of the crystal. These will be used with the XTAL frequency to calculate the value of AMPIADJ, which the XTAL clock initialization uses to set the CFG0 register.
- **Manual** – This option is used by advanced users. This enables the **Amplifier Gain (AMPIADJ)** field to manually enter a specific value for AMPIADJ. The XTAL clock initialization uses this value to set the CFG0 register.

See Also:

- [Clock Editor](#)
- [Configure System Clocks](#)

System Clock APIs

System Clock APIs are provided with the cy_boot generated API files. System clock APIs are listed and described in the System Reference Guide.

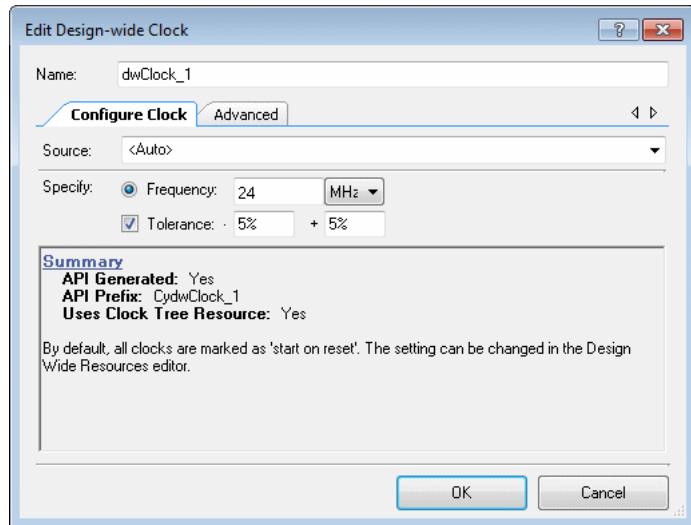
After a successful build, generated files are located in the [Workspace Explorer](#), in a folder named "Generated_Source/<Architecture Name>/cy_boot". For more information about the build process, refer to [Generated Files](#).

See Also:

- [System Reference Guide](#)
- [Workspace Explorer](#)
- [Generated Files](#)

Add/Edit Design-Wide Clock

The Add/Edit Design-Wide Clock dialog allows you to add and edit various design-wide clocks in your design.



Note Design-wide clocks use resources on the device and have APIs generated for them with a "Cy" prefix.

To Open the Dialog:

You open this dialog from the [Clock Editor](#).

- Click **Add Design-Wide Clock** to open the dialog to create a new design-wide clock.
- Double-click an existing design-wide clock to open the dialog for editing.

To Configure Design-Wide Clock:

You can configure the design-wide clock as follows:

1. In **Name**, type a name for the clock or accept the default name.
 2. In **Clock Type** (PSoC 4/PSoC 6 only), select "New" or "Existing."
 3. In **Source**, select the type of source clock from the pull down menu.
- Selecting "<Auto>" allows you to specify the **Desired Frequency** and optionally the **Tolerance**. The source and divider are then calculated by PSoC Creator.
 - Selecting "<Select Signal...>" opens the [Select Source Clock](#) dialog to select from a list of available signals.
 - Selecting "<Select Pin...>" (PSoC 4/PRoC BLE devices only) opens the [Select Source Clock \(from Pin\)](#) dialog to select from a list of available pins.
 - Selecting any other specific base clock allows you to specify the **Desired Frequency** or **Divider**, as follows:
 - If you specify a **Desired Frequency**, PSoC Creator calculates the divider automatically.
 - If you specify a **Divider** is selected, PSoC Creator uses it as specified.

You can view the clock characteristics for various clocks using the design-wide resources [Clock Editor](#).

Note For PSoC 4/PRoC BLE devices, there is a **Use fractional divider** check box. If you specify the **Frequency** option, selecting the **Use fractional divider** check box means that PSoC Creator will calculate the fraction. If you specify the **Divider** option, this check box provides a manual option for you to specify the fraction.

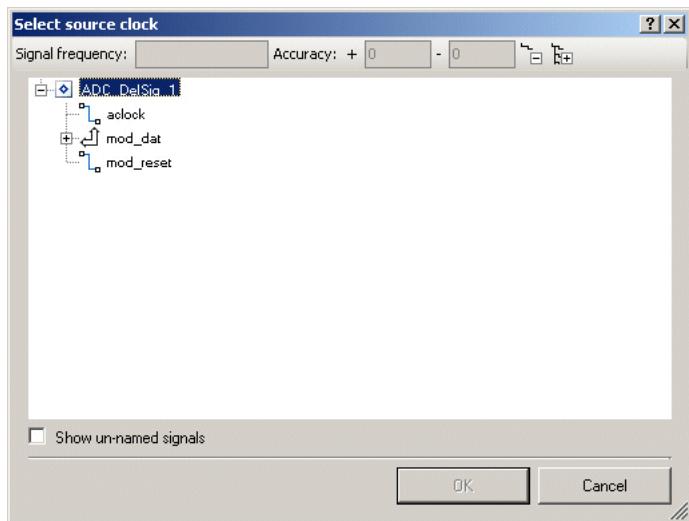
4. On the **Advanced** tab, specify whether or not to synchronize this clock with the **MASTER_CLK** (synchronize, by default). This tab is not applicable to PSoC 4/PSoC 6 devices.

See Also:

- [Design-Wide Resources](#)
- [Clock Editor](#)
- [Select Source Clock](#)
- [Select Source Clock \(from Pin\)](#)

Select Source Clock

The Select Source Clock dialog provides a tree view or a table view of the various signals (in alphabetical order) in your design from which you can select an input signal for a given clock.



To be used as a clock input, a signal must meet all of the following requirements:

- The signal must be a digital signal.
- If the signal is from a terminal, the terminal must be at the Top Schematic level.
- If not at the Top Schematic level, then the signal must not be connected to a schematic terminal.

To Open the Dialog:

You can open this dialog as follows:

- From the [Configure System Clocks](#) dialog, click the ellipsis button [...] in the Digital Signal clock section.

- From the [Add/Edit Design-Wide Clock](#) dialog, under **Source**, choose the "<Select Signal...>" option in the pull down menu. (For PSoC 4/PRoC BLE devices, you must select **Clock Type** "Existing.")

Signal Frequency:

This field is used to specify the frequency of the selected signal.

Accuracy:

These fields are used to specify the accuracy of the selected signal. The value is always displayed as a percent, but can be entered as a % or ppm.

Toolbar:

The toolbar provides expand and collapse commands to show or hide the entire signal tree.

Show Un-named Signals:

Select this check box to show all the signals in your design; de-select to show only named signals.

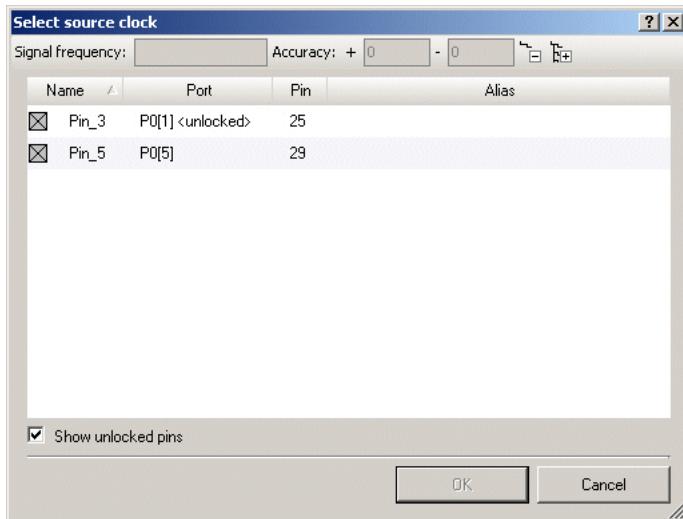
See Also:

- [Design-Wide Resources](#)
- [Clock Editor](#)
- [Add/Edit Design-Wide Clock](#)
- [Configure System Clocks](#)

Select Source Clock (from Pin)

The Select Source Clock dialog provides a table view of the various pin signals (in alphabetical order) in your design from which you can select an input signal for a given clock.

This dialog applies to PSoC 4/PRoC BLE devices only.



To be used as a clock input, a signal must meet be a digital input pin.

To Open the Dialog:

Open this dialog from the [Add/Edit Design-Wide Clock](#) dialog.

1. In **Clock Type**, select "Existing".
2. In **Source**, choose the "<Select Pin...>" option in the pull down menu.

Signal Frequency:

This field is used to specify the frequency of the selected signal.

Accuracy:

These fields are used to specify the accuracy of the selected signal. The value is always displayed as a percent, but can be entered as a % or ppm.

Toolbar:

The toolbar provides expand and collapse commands to show or hide the entire signal tree.

Table Fields:

The signal table contains all the signals in a table format with the following columns:

Column	Description
Name	The name of the signal as defined for the pin.
Port	The device's pin shown in port form. This also indicates if the pins is unlocked.
Pin	The device's pin number for the device to which this signal is assigned.
Alias	When applicable, this shows an alternate name defined for a pin.

Show Unlocked Pins:

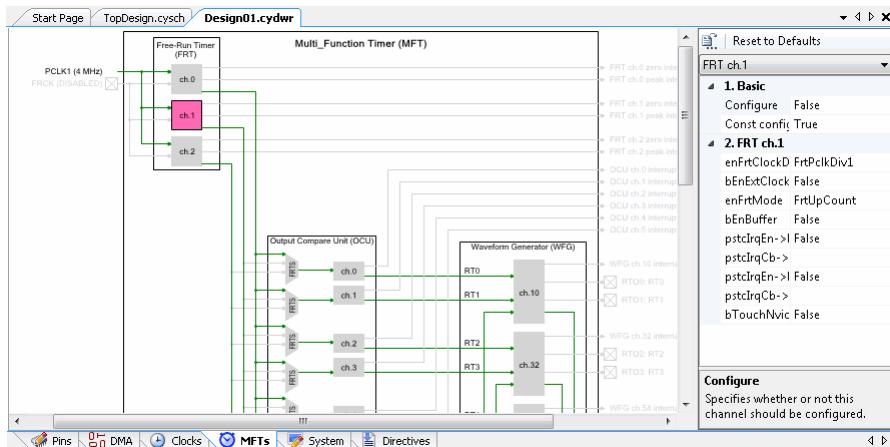
Select this check box to show any unlocked pins in your design.

See Also:

- [Design-Wide Resources](#)
- [Clock Editor](#)
- [Add/Edit Design-Wide Clock](#)

MFT Editor (Certain FM Devices Only)

For a certain set of FM devices, the multi-function timer (MFT) is a function block that enables three-phase motor control. In conjunction with a Programmable Pulse Generator (PPG) and an ADC, the MFT provides a variety of motor controls. The MFT Editor is a graphic representation that allows you to select and configure various blocks within the MFT, in order to generate the corresponding PDL code.



To Open the MFT Editor:

Double click the MFT icon in the Design-Wide Resources tree, located in the **Source** tab of [Workspace Explorer](#).

The `<project>.cydwr` file opens as a [tabbed document](#) in the work area, with the MFT Editor (**MFTs** tab) displayed on top.

To Configure One or More Blocks:

Select a block in the diagram or from the drop-down above the properties area.

Use the properties area to configure the selected block(s).

When you build the project, `cymft_config.h` and `cymft_config.c` files will be generated if any of the blocks in the MFT have a **Configure** property set to true.

MFT Blocks:

An MFT consist of the following blocks:

- **Free-Run Timer (FRT) Unit** – An FRT is a timer function block that outputs counter values for the operational criteria of the function blocks in the MFT. The MFT employs 3 channels.
- **Output Compare Unit (OCU) –** An OCU is a function block that generates and outputs PWM signals on the basis of the counter values of the FRT. The OCU employs 6 channels (2 channels × 3 units).
- **Waveform Generator (WFG) Unit** – A WFG is a function block that is located downstream from the OCU and generates signal waveforms for motor control from OCU output (RT0 to RT5) signals and PPG signals. The WFG employs 3 channels.
- **Noise Canceller (NZCL) Unit** – An NZCL is a function block that generates DTIF interrupts to the CPU from external input signal (DTTIX signal) for motor emergency shutdown. The NZCL employs 1 channel.

- **Input Capture Unit (ICU)** – An ICU is a function block that captures the FRT count value and generates an interrupt in the CPU when a valid edge is detected in an external input pin signal. The ICU employs 4 channels (2 channels × 2 units).
- **ADC Start Compare (ADCMP) Unit** – An ADCMP is a function block that generates AD conversion start signals on the basis of the FRT counter value. The ADCMP employs 6 channels.

Interrupt Editor

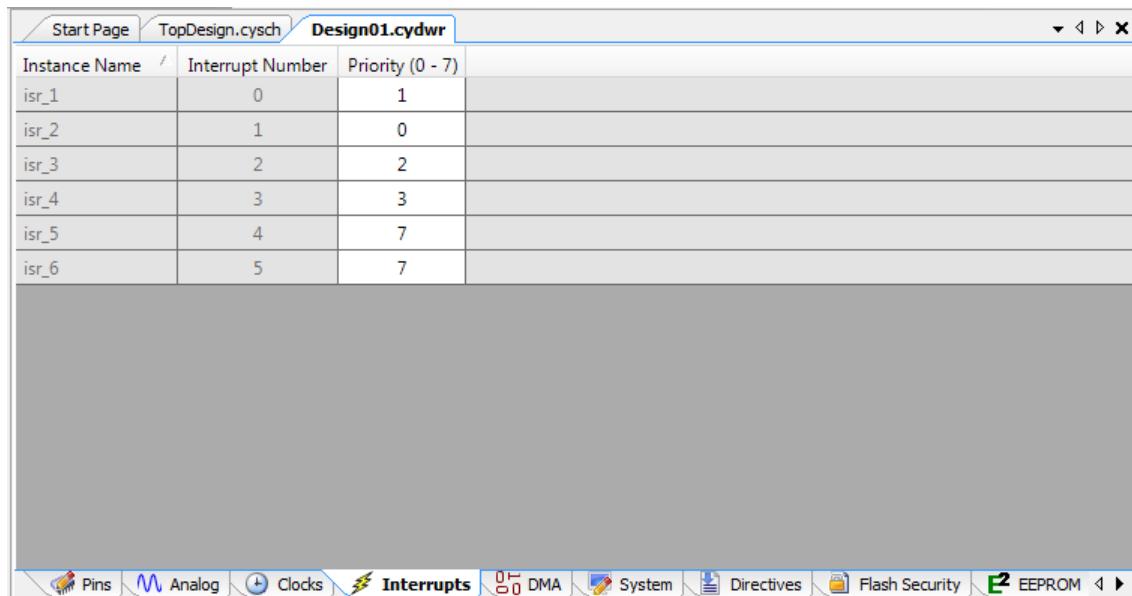
The Interrupt Editor allows you to change the priority of interrupt service routines (ISRs) in your design. This editor is part of the design wide resources file, which includes other resources, such as the Clock Editor.

There is difference between non-multi-core devices (PSoC 3/PSoC 4/PSoC 5LP) and multi-core devices (some PSoC 6).

Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.

Non-Multi-Core Devices

For most devices, the Interrupt Editor looks similar to the following.



The screenshot shows the PSoC Creator interface with the 'Interrupts' tab selected in the bottom navigation bar. The main window displays a table titled 'Design01.cydwr' with three columns: 'Instance Name', 'Interrupt Number', and 'Priority (0 - 7)'. The table contains six rows, each representing an interrupt instance (isr_1 to isr_6) with its corresponding interrupt number and priority level.

Instance Name	Interrupt Number	Priority (0 - 7)
isr_1	0	1
isr_2	1	0
isr_3	2	2
isr_4	3	3
isr_5	4	7
isr_6	5	7

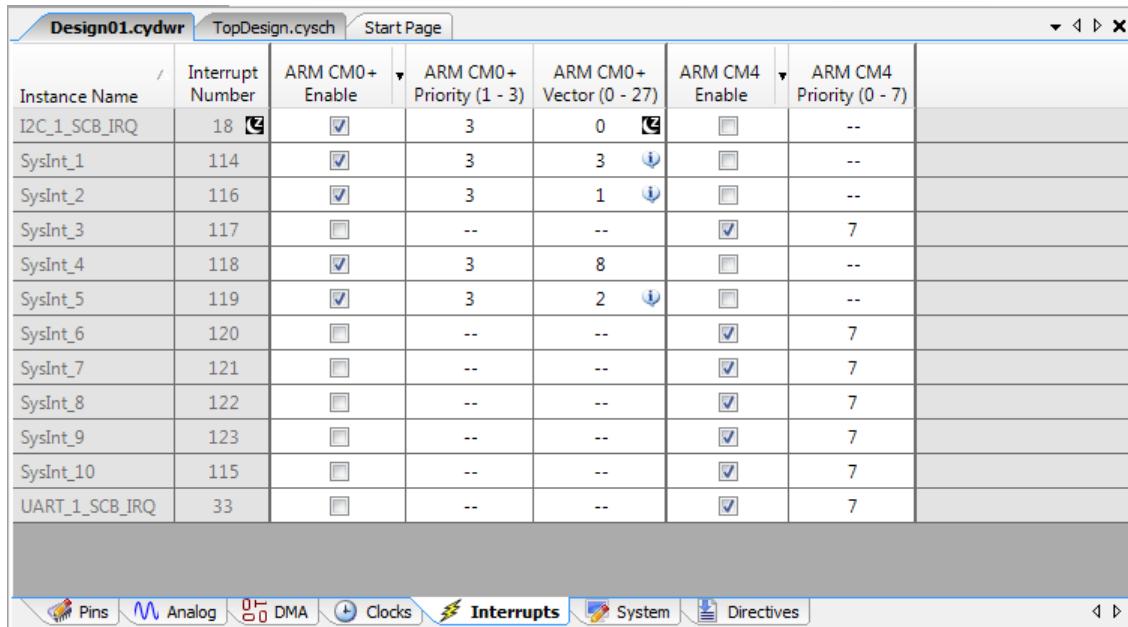
The Interrupt Editor contains a table with the following columns:

- **Instance Name** – The instance name is also used as the ISR name. This property is read-only in the table.
- **Interrupt Number** – This information (interrupt number) will only be available after place-and-route. This property is read-only in the table.
- **Priority** – This is used to enter the interrupt priority. The valid range is displayed in the header. The smaller the number, the higher priority.

If changing devices results in an illegal value, then an error icon will be displayed in the instance name cell. An error will also be placed in the [Notice List window](#).

Multi-Core Devices

For multi-core devices, such as some PSoC 6 devices, there are separate tabs and editors for each core. For non-multi-core PSoC 6 devices, the System Editor is the same as above.



The screenshot shows the 'Interrupts' tab of the Cypress PSoC Creator. The title bar says 'Design01.cydwr' and 'TopDesign.cysch'. The main area is a table with the following columns:

Instance Name	Interrupt Number	ARM CM0+ Enable	ARM CM0+ Priority (1 - 3)	ARM CM0+ Vector (0 - 27)	ARM CM4 Enable	ARM CM4 Priority (0 - 7)
I2C_1_SCB_IRQ	18	<input checked="" type="checkbox"/>	3	0	<input type="checkbox"/>	--
SysInt_1	114	<input checked="" type="checkbox"/>	3	3	<input type="checkbox"/>	--
SysInt_2	116	<input checked="" type="checkbox"/>	3	1	<input type="checkbox"/>	--
SysInt_3	117	<input type="checkbox"/>	--	--	<input checked="" type="checkbox"/>	7
SysInt_4	118	<input checked="" type="checkbox"/>	3	8	<input type="checkbox"/>	--
SysInt_5	119	<input checked="" type="checkbox"/>	3	2	<input type="checkbox"/>	--
SysInt_6	120	<input type="checkbox"/>	--	--	<input checked="" type="checkbox"/>	7
SysInt_7	121	<input type="checkbox"/>	--	--	<input checked="" type="checkbox"/>	7
SysInt_8	122	<input type="checkbox"/>	--	--	<input checked="" type="checkbox"/>	7
SysInt_9	123	<input type="checkbox"/>	--	--	<input checked="" type="checkbox"/>	7
SysInt_10	115	<input type="checkbox"/>	--	--	<input checked="" type="checkbox"/>	7
UART_1_SCB_IRQ	33	<input type="checkbox"/>	--	--	<input checked="" type="checkbox"/>	7

Below the table are tabs for 'Pins', 'Analog', 'DMA', 'Clocks', 'Interrupts' (which is selected), 'System', and 'Directives'.

The Interrupt Editor contains a table with the following columns:

- **Instance Name** – The instance name is also used as the ISR name. This property is read-only in the table.
- **Interrupt Number** – This information (interrupt number) will only be available after place-and-route. This property is read-only in the table. A deep sleep wakeup capable image is disabled when assigned to a deep sleep capable location.
- **Enable [Per Core]** – Provides a checkbox to enable an interrupt in a particular core. The priority and vector values can only be set for cores where the interrupt is enabled. The arrow on the header of this column can be used to check all/uncheck all.
- **Priority [Per Core]** – This is used to enter the interrupt priority. The valid range is displayed in the header. Note: smaller numbers equal higher priority.
- **Vector [Per Core if Applicable]** – Some cores do not have enough vectors to support all interrupts on the device. This column allows mapping an interrupt into a specific location. A deep sleep wake-up capable image is displayed when assigned to a deep sleep capable location and the interrupt number is also deep sleep capable.

If changing devices results in an illegal value, then an error icon will be displayed in the instance name cell. An error will also be placed in the Notice List window.

To Open the Interrupt Editor:

Double click the Interrupts icon in the Design-Wide Resources tree, located in the **Source** tab of [Workspace Explorer](#).

The `<project>.cydwr` file opens as a [tabbed document](#) in the work area, with the Interrupt Editor (**Interrupts** tab) displayed on top.

To Change Priority:

Enter a value for the appropriate interrupt in the text box.

See Also:

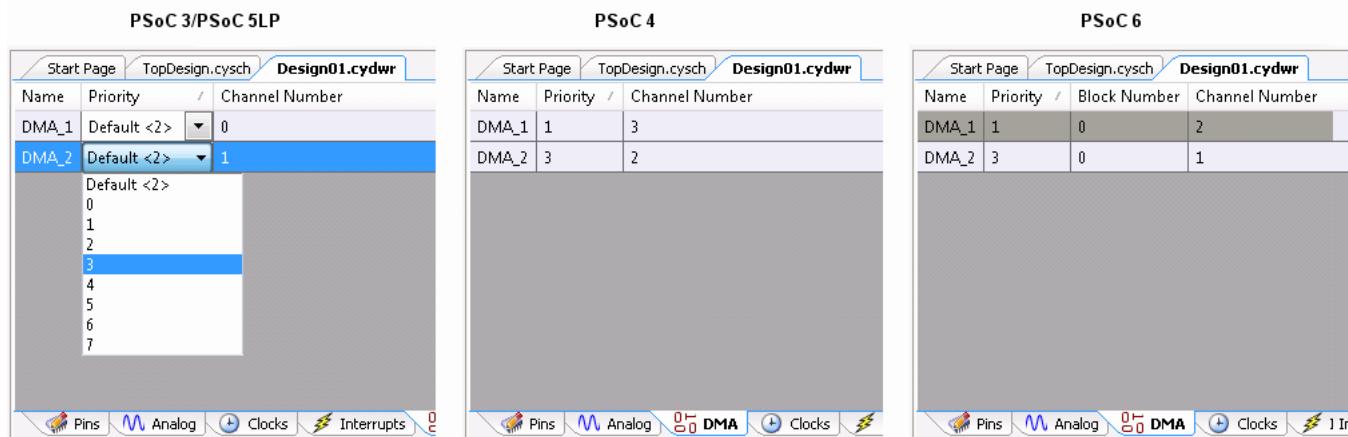
- [Design-Wide Resources](#)
- Interrupt Component Datasheet (open from the [Component Catalog](#))

DMA Editor

The DMA Editor displays all the direct memory access (DMA) Components that have been directly placed in the design, as well as all the DMA Components “inside” placed Components.

Note The DMA Editor does not display for devices that do not have DMA.

Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.



PSoC 3/PSoC 5LP

Name	Priority	Channel Number
DMA_1	Default <2>	0
DMA_2	Default <2>	1

PSoC 4

Name	Priority	Channel Number
DMA_1	1	3
DMA_2	3	2

PSoC 6

Name	Priority	Block Number	Channel Number
DMA_1	1	0	2
DMA_2	3	0	1

The DMA Editor consists of a table where each row represents a DMA instance. The table contains the following columns:

- **Name** – The name of the DMA instance. This property is specified in the Component Configure dialog and is read-only in the table.
- **Priority** – Specifies or indicates the priority of the DMA.
 - **PSoC 3/PSoC 5LP** – There is a default value assigned, denoted as “Default <#>”, so that it can be distinguished from explicitly selecting the default number. Unselected default values will be updated as the device changes; values explicitly chosen will not. If changing devices results in an illegal value, an error icon will be displayed in the cell. An error will also be placed in the [Notice List window](#).
 - **PSoC 4/PSoC 6** – The priority is specified in the Component Configure dialog. This property is read-only in the table.

- **Block Number – PSoC 6 only** has up to two DMA blocks and each DMA block consists of up to 16 channels. This information will only be available after place-and-route. This property is read-only in the table.
- **Channel Number –** This information will only be available after place-and-route. This property is read-only in the table.

Note If no DMA resources are used in a design, the DMA Editor will display a message to that effect.

To Open the DMA Editor:

Double click the DMA icon in the Design-Wide Resources tree, located in the **Source** tab of [Workspace Explorer](#).

The <project>.cydwr file opens as a [tabbed document](#) in the work area, with the DMA Editor (**DMA** tab) displayed on top.

To Change Priority:

For PSoC 3 and PSoC 5LP, click the down arrow on the menu, and select the appropriate Priority value from the pull-down menu.

For PSoC 4 and PSoC 6, the priority must be set in the DMA Component itself. The DWR simply shows the value specified in the Component.

To Sort DMA Editor Table:

By default the DMA Editor is sorted in ascending order by the **Priority** column, with a secondary sort on the channel number column.

To sort by a different column, click the appropriate column header. There will always be a secondary sort performed on the channel number column.

To Select/Edit a DMA:

Double-click an entry in the table; PSoC Creator opens the schematic file and highlights the associated DMA instance, plus it opens the Configure DMA dialog, where you can edit various parameters.

Refer to the appropriate device DMA Component datasheet for more information.

See Also:

- [Design-Wide Resources](#)
- Working with Interrupts
- [DMA Wizard](#)
- [Editing Component Parameters](#)
- DMA Component datasheet (open from the [Component Catalog](#))

DMA Wizard

The DMA Wizard aids in quick and accurate development of applications that use DMA. The wizard guides you through the process of defining transaction descriptors. It also generates the necessary C code that you can copy

and paste into your application. The DMA Wizard only works in the context of a design project that contains at least one DMA Component. If there is no DMA Component in your design, the wizard displays a message to that effect.

The DMA Wizard contains the following steps:

- [Getting Started](#) (this topic)
- [Global Settings](#)
- [Transaction Descriptors](#)
- [Generated Code](#)

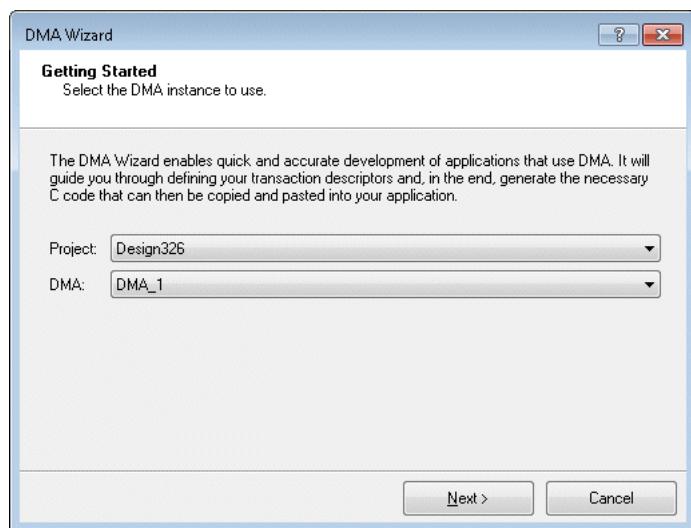
To Open the DMA Wizard:

Select **DMA Wizard** from the **Tools** Menu.

The system will analyze your project for DMA Components and open the wizard. If there are no DMA Components in your design, a message will display stating that you need to open a project with a DMA Component.

Getting Started:

The Getting Started step allows you to select the project and DMA instance for you to configure.



This step contains the following fields:

- **Project** – This lists all the design projects in the currently open workspace that contain DMA Components. By default the “Active” project is selected if it fits the previous requirements.
- **DMA** – Used to select the channel (DMA Component) to use.

After selecting the appropriate project and instance, click **Next >** to proceed to the next step.

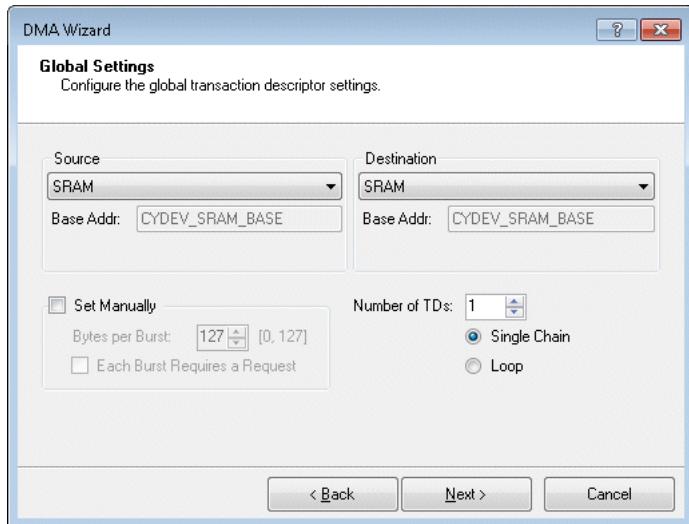
See Also:

- [DMA Wizard Global Settings](#)
- [DMA Wizard Transaction Descriptors](#)

- [DMA Wizard Generated Code](#)
- [DMA Editor](#)

DMA Wizard Global Settings

The Global Settings step is where you configure the global transaction descriptor (TD) settings



This step contains the following settings:

Source:

- **Source** – This can be SRAM, Flash, or EEPROM. It can also be a Component that was created with the ability to be used with the DMA Wizard. If the selected Component has more than one possible source, a second drop-down will appear allowing for a more specific selection.
- **Base Addr** – Depending on which PSoC device is used, the contents of the Base Address may or may not be automatically provided. When not filled in by default, a C expression needs to be provided. This is used to provide the upper 16 bits of the address for the DMA Channel configuration. If not added, an error icon will appear. This will not prevent you from continuing with the wizard. It will only generate code that will not compile.

Destination:

- **Destination** – This can be SRAM or a Component that was created with the ability to be used with the DMA Wizard. If the selected Component has more than one possible destination, a second drop-down will appear allowing for a more specific selection.
- **Base Addr** – Depending on which PSoC device is used, the contents of the Base Address may or may not be automatically provided. When not filled in by default, a C expression needs to be provided. This is used to provide the upper 16 bits of the address for the DMA Channel configuration. If not added an error icon will appear. This will not prevent you from continuing with the wizard. It will only generate code that will not compile.

Set Manually:

The following fields are defined automatically. To set these manually, select the **Set Manually** check box.

- **Bytes per Burst** – Allows you to set the number of bytes to transfer in a single burst. This value is calculated to be the maximum value that is supported by both the source and the destination. To the right of this field the

range of legal values for the current source and destination is displayed. If you enter an invalid value or if there is no legal value, an error icon will appear next to the field. This will not prevent you from continuing with the wizard. It will only generate code that will not compile.

- **Each Burst Requires a Request** – Allows you to set whether or not each burst requires a request before it is sent. The value is automatically calculated based on the selected source and destination.

Transaction Descriptors:

- **Number of TDs** – Specifies the number of transaction descriptors to create (between 1 and 128).
- **Single Chain or Loop** – This determines what the **Next TD** will be for the last TD entered. If single chain the Next TD will be END. If Loop it will loop back to the first TD.

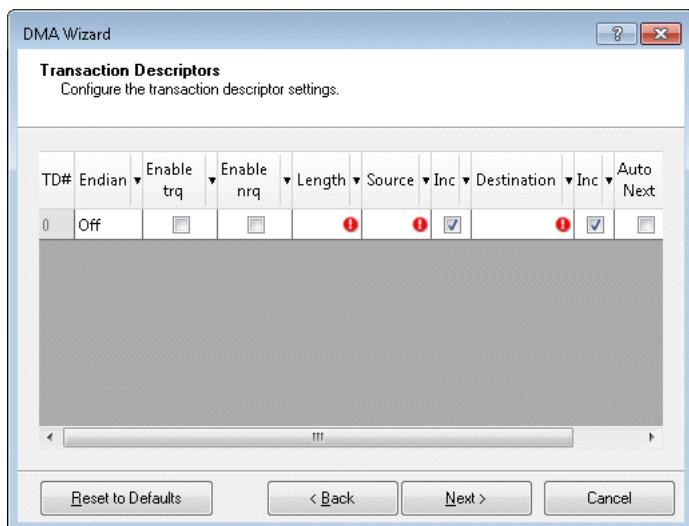
After configuring the appropriate settings, click **Next >** to proceed to the next step.

See Also:

- [DMA Wizard](#)
- [DMA Wizard Transaction Descriptors](#)
- [DMA Wizard Generated Code](#)
- [DMA Editor](#)

DMA Wizard Transaction Descriptors

The Transaction Descriptors step is where you configure settings for each individual transaction descriptor (TD).



This step contains the following fields:

Field	Description
TD#	Displays the logical Transaction Descriptor number. It is used in conjunction with Next TD.
Endian	Enables 2- or 4-byte endian byte swapping. When set to 2 or 4, the Bytes per Burst setting must be set as a multiple of the endian selection. An error will be added to the cell if this is not the case. This will not prevent you from continuing with the wizard. It will only generate code that will not compile.

Enable trq	Enables terminating this TD on a rising edge of the trq signal.
Enable nrq	Enables the generation of the nrq signal when the TD completes.
Length	The length in bytes for this TD (0 to 4095). This field is a C expression that is evaluated at run time. If endian swapping is enabled, the length must be a multiple of endian swap size. An error will be added to the cell if this is not the case. This will not prevent you from continuing with the wizard. It will only generate code that will not compile.
Source	The lower 16-bit source address for the DMA transfer. This field is a C expression that is evaluated at run time to configure the TD. It is combined with the upper 16-bit base address provided on the Global Settings page. If a Component is selected as the source, this field may be a drop-down list of addresses. In any case, the cell is editable. If empty, an error icon will be added to the cell. This will not prevent you from continuing with the wizard. It will only generate code that will not compile.
Inc (Source)	Enables incrementing of the Source address as the DMA progresses through the specified number of bytes.
Destination	The lower 16-bit destination address for the DMA transfer. This field is a C expression that is evaluated at run time to configure the TD. It is combined with the upper 16-bit base address provided on the Global Settings page. If a Component is selected as the source, this field may be a drop-down list of addresses. In any case, the cell is editable. If empty, an error icon will be added to the cell. This will not prevent you from continuing with the wizard. It will only generate code that will not compile.
Inc (Destination)	Enables incrementing of the Destination address as the DMA progresses through the specified number of bytes.
Auto Next	Specifies whether or not to automatically execute the next TD once this TD completes without requiring another request.
Next TD	Specifies the next logical TD in the chain of TDs. Set to END if this TD chain is complete with this TD.

The **Reset to Defaults** button will reset all the values in the table to be their default, calculated values.

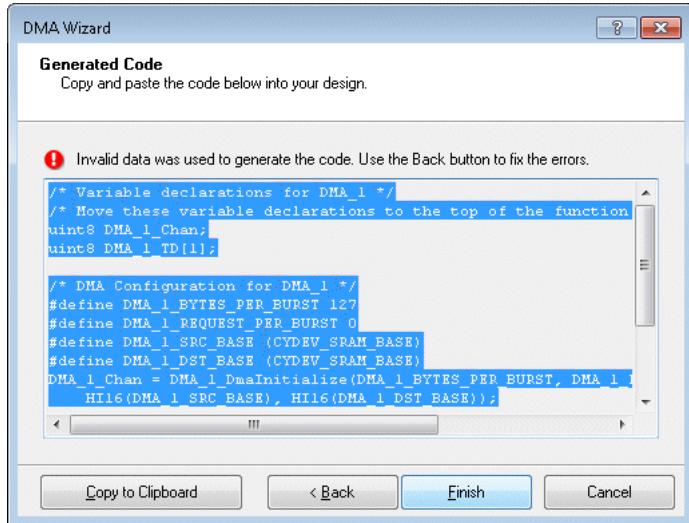
After configuring the appropriate settings, click **Next >** to proceed to the next step.

See Also:

- [DMA Wizard](#)
- [DMA Wizard Global Settings](#)
- [DMA Wizard Generated Code](#)
- [DMA Editor](#)

DMA Wizard Generated Code

The Generated Code step is where the code is displayed for you to copy and paste into your design.



The **Copy to Clipboard** button adds the code to the clipboard. You can also use standard keyboard shortcuts and the right-click menu, as needed.

Note If any errors were ignored in the previous steps, an error icon will display reminding you that the code will not work.

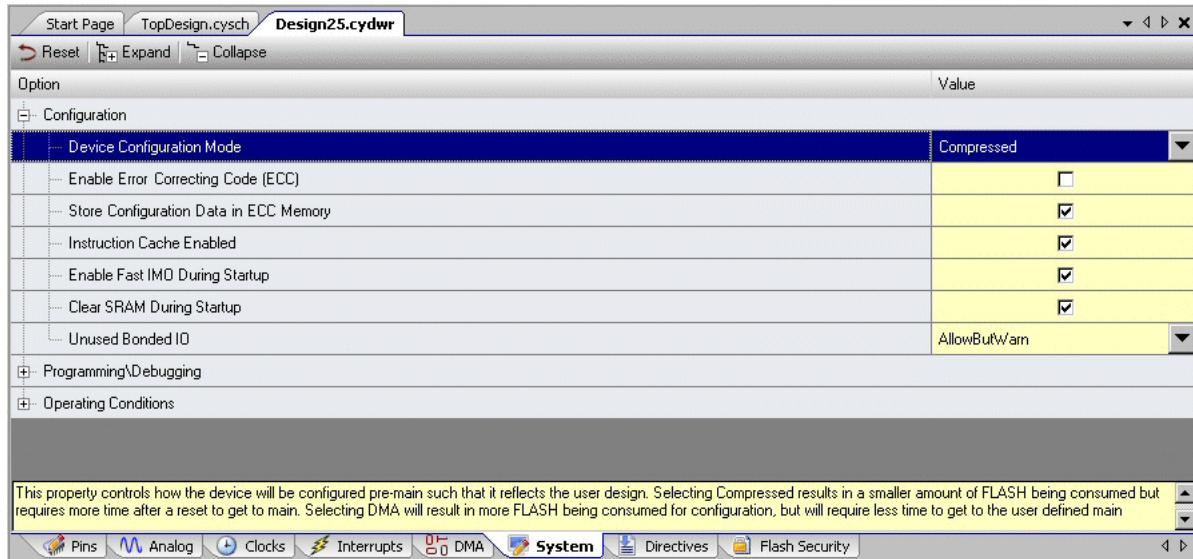
See Also:

- [DMA Wizard](#)
- [DMA Wizard Global Settings](#)
- [DMA Wizard Transaction Descriptors](#)
- [DMA Editor](#)

System Editor

The System Editor is used to edit various system properties. It contains a table with different categories of properties, such as Configuration, Programming/Debugging, and Operating Conditions. The available categories change based on your design.

Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.



To Open the System Editor:

Double click the System icon in the Design-Wide Resources tree, located in the **Source** tab of [Workspace Explorer](#).

The `<project>.cydwr` file opens as a [tabbed document](#) in the work area, with the System Editor (**System** tab) displayed on top.

To Edit a Property:

Click in the **Value** column for a property to edit.

Different properties have different methods of editing. Some properties have a check box to toggle on and off, some have a pull down menu to choose an option, and some have a text field in which to enter a value.

If you enter an invalid value, an error will display to indicate the invalid value and how you might correct the problem.

Property Descriptions:

Under each category, there are one or more rows of properties you can edit. When you highlight a particular property, its description displays in the text box at the bottom of the editor. The table contains the following columns:

- **Option** – This column displays the name of each row/level in the hierarchy.
- **Value** – This column displays the current value of a setting and allows you to change it when applicable.

The table contains the following options:

Option	Description	Note
Configuration		
Device Configuration Mode	<p>This property controls how the device will be configured pre-main, such that it reflects the design. The options available are Compressed, UnCompressed, and DMA. (The DMA option applies to PSoC 3 and PSoC 5LP devices only.)</p> <ul style="list-style-type: none"> • Compressed results in a smaller amount of FLASH being consumed but requires more time after a reset to get to main. • UnCompressed will use the same amount of FLASH as DMA, but will perform a software copy of configuration data. • DMA will result in more FLASH being consumed for configuration, but will require less time to get to the user-defined main function. 	Not applicable to FM0+ devices.
Enable Error Correcting Code (ECC)	<p>If true, the ECC will be used to detect and correct errors in the FLASH memory. Selecting this option hides the "Store Configuration Data in ECC Memory" option.</p> <p>WARNING Exercise caution changing this setting during development. Excessive re-programming of this setting may cause unexpected results.</p>	Applies to PSoC 3 and PSoC 5LP devices only.
Store Configuration Data in ECC Memory	If this option is enabled, device configuration data will be stored in ECC memory to reduce main FLASH memory usage. Error correction may not be used when this option is enabled.	Applies to PSoC 3 and PSoC 5LP devices only.
Instruction Cache Enabled	If true, the device will write data coming from the FLASH to the instruction cache SRAM.	Applies to PSoC 3 and PSoC 5LP devices only.
Enable Fast IMO During Startup	<p>If true, the fast IMO will be used. This configuration balances the need for rapid boot and configuration against peak power consumption. If true, the IMO will run at the faster speed of 48 MHz instead of 12 MHz during device startup.</p> <p>This configuration balances the need for rapid boot and configuration against peak power consumption. If true, the IMO will run at the faster speed of 48 MHz instead of 12 MHz during device startup and between CyPmSaveClocks() and CyPmRestoreClocks() functions calls. See System Reference Guide for more information about these functions.</p> <p>WARNING Exercise caution changing this setting during development. Excessive re-programming of this setting may cause unexpected results.</p>	Applies to PSoC 3 and PSoC 5LP devices only.
Clear SRAM During Startup	If enabled, writes zeros to SRAM before initializing variables. If disabled, variables that do not have an explicit initializer will not be initialized to zero. This option should only be disabled if the application requires a faster startup time and does not contain any code or libraries that depend on the value of un-initialized variables.	Applies to PSoC 3 devices only.
Read Accelerator Enabled	If true, the device will accelerate read operations using cached values.	Applies to PSoC 4 devices only.
Unused Bonded IO	<p>This option controls how unused bonded pins will be used for internal analog place and route.</p> <ul style="list-style-type: none"> • “Allow but warn” option will allow the analog router to make use of unused pin switches in the current design, but will give out warnings on the pins whose switches are used. • “Allow with info” option will allow the analog router to make use of unused pin switches in the current design, and will give out notes on the pins whose switches are used. • “Disallowed” option will disallow the analog router to use any of the unused pin switches in the current design. 	Not applicable to FM0+ devices.

Option	Description	Note
Heap Size	Defines the number of SRAM bytes to reserve for the Heap space.	Applies to PSoC 4 and PSoC 5LP devices only.
Stack Size	Defines the number of SRAM bytes to reserve for the Stack space.	Applies to PSoC 4 and PSoC 5LP devices only.
Include CMSIS Core Peripheral Library Files	The CMSIS (Cortex Microcontroller Software Interface Standard) Core Peripheral Library contains APIs to access core registers and peripherals. Checking this option will include the APIs from the current version of the standard in the project. The current version is tied to the version of the cy_boot Component in your project. Refer to the System Reference Guide for more information. Deselect the box and add the files manually if you wish to use a different version of the standard.	Applies to PSoC 4 and PSoC 5LP devices only.
Programming/Debugging		
Chip Protection	<p>The different chip protection levels provide limitations on what resources are accessible by the CPU and Debugger:</p> <ul style="list-style-type: none"> • Open: full access • Protected: no debugging • Kill: the device can never be reprogrammed <p>Note:</p> <ol style="list-style-type: none"> 1. Programming a PSoC 4/PSoC BLE device using PSoC Creator will set the chip protection to "Open" even it is selected as "Protected" or "Kill" mode. 2. In order to set the Chip protection to "Protected" or "Kill" mode, select the appropriate mode in PSoC Creator and program the PSoC 4/PSoC BLE device using PSoC Programmer after enabling "Chip Lock" feature in PSoC Programmer Options > Programmer Options. 	Applies to PSoC 4 devices only.
Debug Select	<p>Sets the Port 1 preferred program/debug interface (JTAG or SWD) that the chip enables by default for use after power up or reset.</p> <ul style="list-style-type: none"> • For PSoC 4/FM0+, JTAG is not available. • Setting to GPIO frees the pins for use as GPIOs but does not completely disable the debug interface for flash protection purposes. "Enable Device Protection" must be set for this purpose, or "Chip Protection" must be set to Open. <p>Note This setting must match how you intend to program your device in a 3rd party IDE. See also Integrating into 3rd Party IDEs.</p> <p>For more information about programming and debugging options see the device datasheet or Technical Reference Manual (TRM).</p> <p>WARNING Exercise caution changing this setting during development. Excessive re-programming of this setting may cause unexpected results.</p>	PSoC 3/PSoC 5LP/PSoC 6 options include: <ul style="list-style-type: none"> • 5-wire JTAG • 4-wire JTAG • SWD (serial wire debug) • SWD+SWV (serial wire debug and viewer) • GPIO PSoC 4/FM0+ options include: <ul style="list-style-type: none"> • SWD (serial wire debug) • GPIO
Enable Device Protection	When completing a production design, you may select the Enable Device Protection feature. Enabling this feature causes the part to disable debugging at run-time. It is still possible to connect a programmer, but debugging will be disabled. It is not recommended to enable it for multi-device JTAG chains, since it may break the chain.	Applies to PSoC 3 and PSoC 5LP devices only.

Option	Description	Note
	Note This setting does not affect flash protection. It is only used to disable debug access to the PSoC 3 or PSoC 5LP device.	
Embedded Trace (ETM)	Enables the Cortex ETM Trace capability for outputting real-time debug information while the processor is running. This will reserve the trace pins as debug pins. When not used for trace, PSoC Creator makes the pins available as a GPIO.	Applies to PSoC 5LP and PSoC 6 devices only.
Use Optional XRES	Enables hardware reset via the optional XRES pin (P1[2]). This is in addition to the dedicated XRES pin on this device. WARNING Exercise caution changing this setting during development. Excessive (over 1000 times) re-programming of this setting may cause the selection to become permanent.	Applies to PSoC 3 and PSoC 5LP devices with more than 48 pins.
Enable XRES	Enables hardware reset via the optional XRES pin (P1[2]). This device does not have a dedicated XRES pin, if disabled it is not possible to program the part without a power cycle or to issue a hard reset from the debugger. By default, the reset is done by toggling the XRES pin. The power cycle method is highly dependent on the design of your board, and may not be possible. If a XRES pin is not available and the Power Cycle does not work, the part cannot be reprogrammed. To ensure that the device can be programmed, this option should be enabled. WARNING Exercise caution changing this setting during development. Excessive (over 1000 times) re-programming of this setting may cause the selection to become permanent.	Applies to PSoC 3 devices with 48 pins or less.

Operating Conditions

These settings specify the various voltages and temperature ranges in which the device is used. Various Components in the device use these values for configuration information, so you should use correct values. There are numerous ways these settings can affect your design, including:

- The USB_Start function has an option to use Vddd to set the internal USB regulators for enumeration.
- The ADC_CountsTo_Volts() API uses these voltages for the Vssa to Vdda Input Range.
- The SAR_ADC Component uses the Vdda voltage setting for configuring the Input Range and Reference.
- The SC block Components (TIA, PGA, PGA_Inv, Sample_Hold, Mixer) enable boost clocks when the Vdda is set below 2.7 V.
- The VDAC8 Component generates a note explaining that the range of the VDAC is limited to the range from 0 V to Vdda.
- Static Timing Analysis uses smaller timing delays in the UDBs if Vddd is \geq 1.8 V (and you selected the 0 °C – 85 °C Temperature Range). IO delays are smaller if the voltage (VddioN) is 3.3 V or higher.

Option	Description	Note
Pin Voltages (VDDA, VDDD, VDDIO<X>, VDDR, VBUS, etc.)	Individual options are available for each power pin on the selected device. The input value indicates the input voltage on the corresponding pin.	The exact list of options will change based on the selected device.

Option	Description	Note
External PMIC Output	This option provides the ability to enable an external PMIC input to provide VDDD power to the device. Options include: <ul style="list-style-type: none"> • Disabled • Enabled 	Applies to PSoC 6 devices only.
Temperature Range	Specifies the temperature range at which the device will be operated. PSoC devices operate reliably across a wide temperature range. If your design will only ever run at typical room temperatures, however, the timing constraints are more easily satisfied. Selecting the narrower temperature range for your application helps the tool to find timing-compliant routing solutions.	Available selections are: <ul style="list-style-type: none"> • 0C – 85C (PSoC 3 and PSoC 5LP only) • -40C – 85C (all devices)
Variable VDDA	Option to create low voltage analog boost clock. Useful to allow low voltage boost when Vdda varies over time such as a battery powered application.	Not applicable to FM0+ devices.
Power Mode	This option provides the ability to switch between using the LDO Linear Regulator or the SMIO Buck to supply VCCD. Note There is no hardware protection when using the SMIO Buck. No configuration available for Ultra Low Power (ULP) devices.	Applies to PSoC 6 devices only.

Note: The voltage values are used by certain APIs (for example, ADC_CountsTo_Volts), as well as Component Configure dialogs (for example, the ADC_DelSig Component uses this Vdda information for showing input range if Vref is selected to Vdda/4 or Vdda/3). Although it is recommended to update the actual Vddx voltages in the DWR System Editor, the device will not get damaged if a different valid (as per the datasheet) supply voltage is given.

Analog Reference

For some devices, there is an additional section with the following options:

Option	Description	Note
Reference Source	This option specifies the system wide programmable reference source. Available selections are: <ul style="list-style-type: none"> • Bandgap (1.20 V) • VDDA (3.30 V) 	Applies to PSoC Analog Coprocessor and PSoC 4100PS series devices only.
Bandgap Reference Gain	This option controls the Bandgap multiplier used by all programmable references. Available selections are: <ul style="list-style-type: none"> • 1x (1.20 V) • 2x (2.40 V) 	Applies to PSoC Analog Coprocessor and PSoC 4100PS series devices only.
Voltage Reference Value	This option specifies the system wide programmable reference voltage. Voltage options are based on 1/16th increments of VDDA and Bandgap * Reference Gain. This voltage is accessible by the voltage reference Component. Different selections ranging from 0.08 V to 1.20 V.	Applies to PSoC Analog Coprocessor and PSoC 4100PS series devices only.
VDDA references active during DeepSleep	This option specifies whether or not VDDA-based references will remain active when the chip is in Deep Sleep. References which are based off of the bandgap reference (VBGR) are never active in DeepSleep.	Applies to PSoC Analog Coprocessor and PSoC 4100PS series devices only.
Bandgap Value	This option specifies the analog reference value. The system reference requires less power but provides lower performance than the local reference. This voltage is accessible by the voltage reference Component. Options: (System, Local, External Pin)	Applies to PSoC 6 devices only.
Opamp Reference Current	Selects the reference current available to the opamps. Selecting a lower value will reduce power consumption but will result in lower performance. See the opamp component datasheet for more details.	Applies to PSoC 6 devices only.

Option	Description	Note
Available in DeepSleep	This option specifies whether or not the bandgap reference will remain active when the chip is in Deep Sleep.	Applies to PSoC 6 devices only.

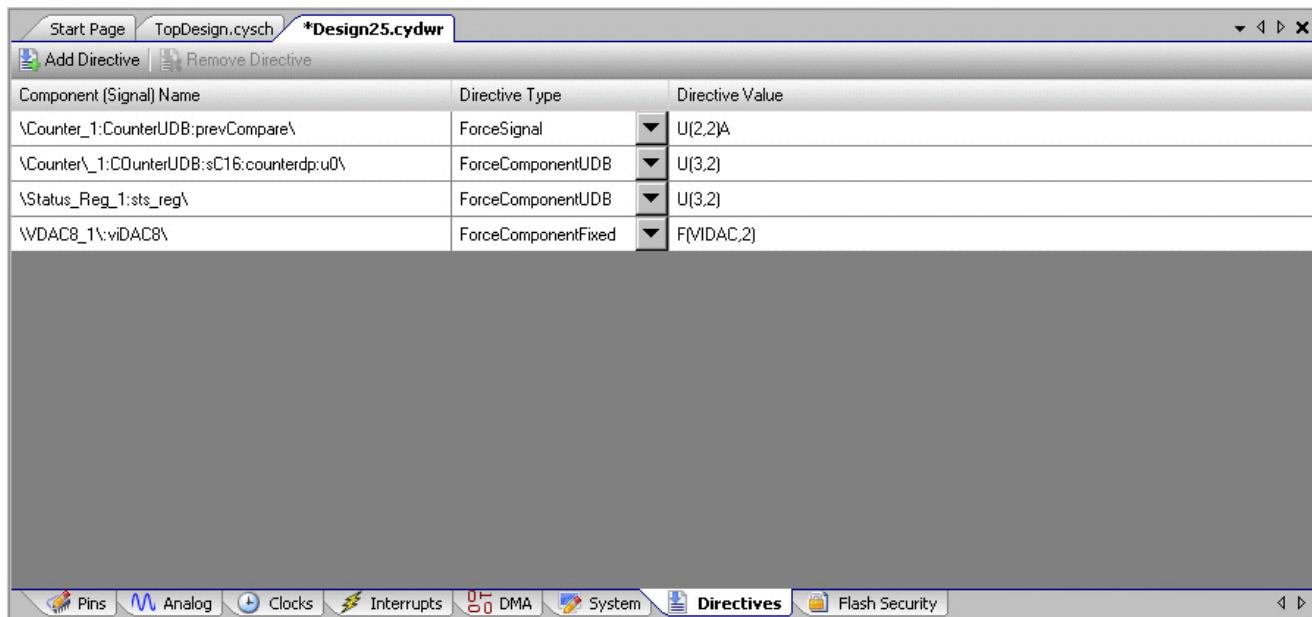
See Also:

- [Design-Wide Resources](#)

Directives Editor

The Directives Editor is used to add, remove, and edit directives. See [Directives](#) for more information about the directives available in PSoC Creator.

Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.

**To Open the Directives Editor:**

Double click the Directives icon in the Design-Wide Resources tree, located in the **Source** tab of [Workspace Explorer](#).

The `<project>.cydwr` file opens as a [tabbed document](#) in the work area, with the Directives Editor (**Directives** tab) displayed on top.

To Add a Directive:

Click the **Add Directive** button on the top left of the editor.

To Edit a Directive:

Enter a **Component/Signal name**, **Directive Type**, and **Directive Value** as appropriate in each column.

- **Component (Signal) Name** - Use this column to enter the signal or Component name. Currently there is no validation check for this column, except you must enter a name in this column if you enter a value in the Directive Value column.
- Note** The Component Name for this field is the fully elaborated name of the Component as specified in the `<project>.rpt` file after a successful build (e.g, "\Counter_1:CounterUDB:sC8:counterdp:u0\"). You can find the `<project>.rpt` file under the **Output** tab in the [Workspace Explorer](#).
- **Directive Type** - Use this column to select a valid directive type from the drop down list. The initial value for this column is INVALID. If you enter a value in the Directive Value column, you must change the Directive Type to the appropriate type. Valid types include:
 - **ForceSignal**: Maps to the 'placement_force' directive format used for arbitrary logic. Allows the assignment of a UDB PLD location to the output signal of a block of logic to tell the placer to put that output in that UDB PLD. This generates a rule of the form:


```
attribute placement_force of [signal name] : signal is "[UDB spec]"
```

where [UDB spec] : U(x,y)[A|B] or U(x,y,[A|B])i
 - **ForceComponentUDB**: Maps to the 'placement_force' directive format for UDB Components. Assigns the location of UDB Component listed to the location specified. This generates a rule of the form:


```
attribute placement_force of [Component name] : label is "[UDB spec]"
```

where [UDB spec]: U(3,2)
 - **ForceComponentFixed**: Maps to the 'placement_force' directive format for Fixed Function blocks. Assigned the location of the fixed block listed to the location specified. This generates a rule of the form:


```
attribute placement_force of [Component name] : label is "[fixed spec]"
```

where [fixed spec] : F([fixed block],i)

where [fixed block] : CAN, Comparator, I2C, SC, Timer, VIDAC, ...
 - **Group**: Maps to the 'placement_group' directive format. Groups the specified signal name into the specified group. This generates a rule of the form:


```
attribute placement_group of [signal name] : signal is "[group name]"
```
 - **Directive Value** - Use this column to specify the value for the directive, based on the selected Directive Type. Each type has certain rules for its value. If the value does not follow the rules of the type, an error icon will display next to the text (for example, see signal_5 in the image). Mouse over the error icon to show the reason of the error.

To Delete a Directive:

Click a row in the table and click the **Delete Directive** button.

See Also:

- [Design-Wide Resources](#)

- [Directives](#)

Flash Security Editor

The Flash Security Editor allows you to control the read/write access to the flash memory. This feature is designed to secure proprietary code.

Flash rows are displayed as a table where each editable cell in the table represents a single row of flash (64/128/256 bytes depending on the PSoC 4 device; 256 bytes for PSoC 3 and PSoC 5LP devices). Each row of flash can have its protection level independently set.

Note This feature is not applicable to PSoC 6 devices, for which flash security is handled by the MPU/SMPU/PPU as part of the firmware.

OFFSET:	000	100	200	300	400	500	600	700	800	900	A00	B00	C00	D00	E00	F00	Row
BASE ADDR: 0000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	0-15
1000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	16-31
2000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	32-47
3000	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	48-63
4000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	64-79
5000	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	80-95
6000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	96-111
7000	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	112-127
►	8000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	128-143
9000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	144-159
A000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	160-175
B000	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	176-191

Flash memory is organized as rows with each row of flash having 256 bytes. Each flash row can be assigned one of 4 protection levels:

Pins Analog Clocks Interrupts DMA System Directives Flash Security EEPROM

Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.

Protection Levels:

The tool offers four levels of protection, as follows. You can assign one of four protection levels (two levels for PSoC 4) to each row; see the table below. Flash protection levels can only be changed by performing a complete flash erase. For more information on PSoC flash and security features, refer to a device datasheet or Technical Reference Manual (TRM).

- **Unprotected (U)** – No protection.
- **Factory Upgrade (F)** – Read protected. No device external to the PSoC device can read a flash block that is read-protected. The SPC Read commands cannot be used to read a block that is read protected. Only the processor and the PHUB can access a block of flash that is read protected. (This option is not available for PSoC 4 devices.)
- **Field Upgrade (R)** – External write protection. No device external to the PSoC device can erase or write a row of flash that is external write protected. Includes all Read Protect restrictions. (This option is not available for PSoC 4 devices.)

- **Full Protection (W)** – Fully protected. Neither the PSoC CPU nor any device external to PSoC can erase or write a block of flash that is fully protected. Includes all protections from lower levels of flash data protection. This level is used when a block of flash should never be modified by an internal process or external device.

	PSoC 3 and PSoC 5LP		PSoC 4	
Protection Setting	Allowed	Not Allowed	Allowed	Not Allowed
Unprotected	External read and write, Internal read and write	–	External read and write, Internal read and write	–
Factory Upgrade	External write, internal read and write	External read	n/a	n/a
Field Upgrade	Internal read and write	External read and write	n/a	n/a
Full Protection	Internal read	External read and write, Internal write	Internal read	External write, Internal write (see Note below)

Note To protect the PSoC 4 device from external read operations, you must change the device protection settings to “Protected” in the [DWR System Settings](#). You must also enable “Chip Lock” from [Options > Programmer Options](#) before programming the device for these settings to take effect. You must use the PSoC Programmer tool to program the device.

To protect the bootloader portion of flash, set the corresponding rows to “full protection.” PSoC Creator lets you easily select the protection setting for each row.

Note The Full Protection level cannot be used on the last two rows of flash for Bootloader or Bootloadable projects. These rows are used for application metadata and require internal write access.

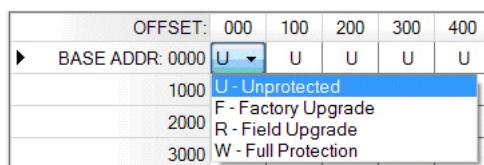
To Open the Flash Security Editor:

Double click the Flash Security icon in the Design-Wide Resources tree, located in the **Source** tab of [Workspace Explorer](#).

The `<project>.cydwr` file opens as a [tabbed document](#) in the work area, with the Flash Security Editor (**Flash Security** tab) displayed on top.

To Change a Single Row of Flash:

To change the protection level on a single flash row, click the corresponding cell in the table and make your selection from the drop-down list.



To Change Multiple Rows of Flash:

To change the protection level on multiple rows of consecutive flash at once, proceed as follows:

1. Select your starting and ending flash row numbers using the fields provided at the top of the editor.

From row: 0 to 255 U - Unprotected

By default the starting and ending values are set to include all rows of flash.

2. Select your desired protection level from the drop-down list located to the right of the previous fields.
3. Click the **Set** button.

Generated Hex (PSoC 3 Only):

There are two generated hex files with flash security levels in them. During a build, the flash protection data is gathered from the CyFlashSecurityModel and outputted as a hex file (located at *Generated_Source/[Architecture]/protect.hex*). This data is then combined into the overall hex file for the project (located at *[Platform (DP8051-Keil_Generic/Debug, ...)]/[Configuration (Debug, Release)]/[Prj Name].hex*).

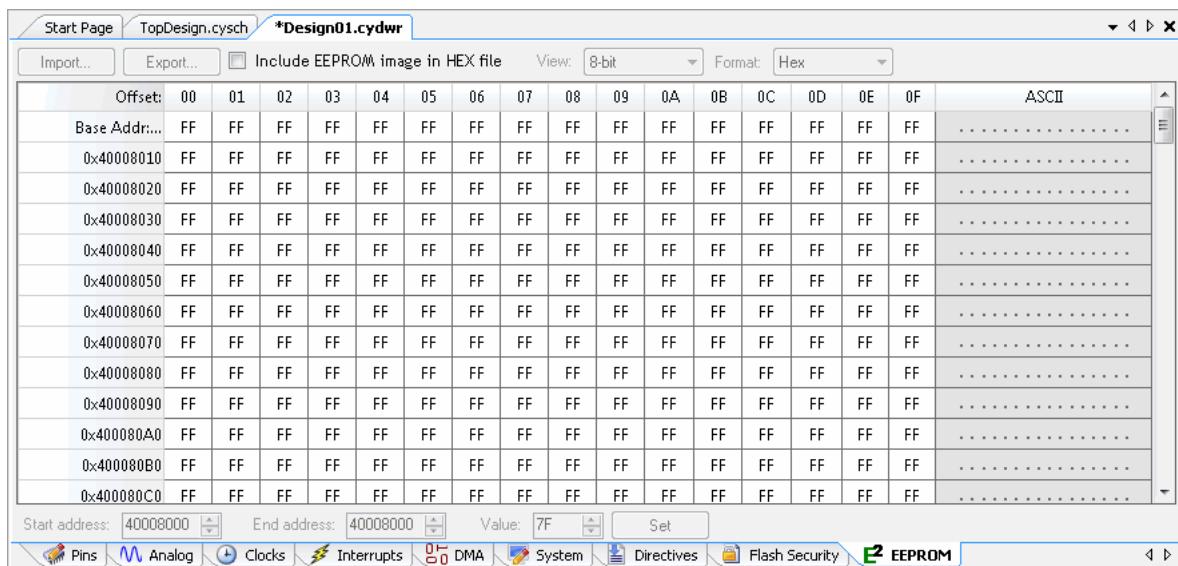
See Also:

- [Design-Wide Resources](#)

EEPROM Editor

The EEPROM Editor allows you to set up EEPROM data from PSoC Creator without requiring any code to run in the PSoC application. It is a grid that displays the EEPROM memory according to the display options. Each cell is editable, with a default value of 0xFF.

Note PSoC Creator collects DWR information dynamically. Depending on the complexity of the design, it may take a few seconds to update the DWR information.



Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
Base Addr:...	FF															
0x40008010	FF															
0x40008020	FF															
0x40008030	FF															
0x40008040	FF															
0x40008050	FF															
0x40008060	FF															
0x40008070	FF															
0x40008080	FF															
0x40008090	FF															
0x400080A0	FF															
0x400080B0	FF															
0x400080C0	FF															

Start address: 40008000 End address: 40008000 Value: 7F Set

Pins Analog Clocks Interrupts DMA System Directives Flash Security EEPROM

The left column shows the base addresses. The top row shows the offsets for each column. The ASCII display on the right is similar to that shown in the [Memory Window](#).

To Open the EEPROM Editor:

Double click the EEPROM icon in the Design-Wide Resources tree, located in the **Source** tab of [Workspace Explorer](#).

The `<project>.cydwr` file opens as a [tabbed document](#) in the work area, with the EEPROM Editor (**EEPROM** tab) displayed on top.

To Change Single Cell Value:

To change the value of a single cell, click in the cell and either type the desired value or use the menu to select a value from the list.

To Change Multiple Cell Values:

To change the value of multiple cells, use fields below the grid. Enter or select the appropriate values for **From address**, **to**, and **Value**. Then click **Set**.

To Import/Export Data:

You can write a sequence of bytes in an external editor in comma-separated value (CSV) format, save it as a CSV file, and import the file into PSoC Creator. Click **Import**, navigate to the CSV file location, select it and click **Open**. The changes are applied in the EEPROM Editor.

You can also make edits in the EEPROM Editor, export the data as a CSV file, then open the file in an external editor. Click **Export**, navigate to the location to save the file, and click **Save**.

To Include EEPROM Image in Hex file:

Select this check box to include the EEPROM image in the hex file. If selected, programming time increases. The default is not selected.

Note If this check box is not selected, there is no functional change to your project.

To Change the Display:

The following settings are stored in the user configuration file and change the display of all projects for that user. These values change the display only; they do not modify the actual values of the data.

- Use the **View** pull-down menu to change the size of the data displayed in the editor: 8-bit, 16-bit, or 32-bit.
- Use the **Format** pull-down menu to change the format of the data displayed in the editor: Hex, Signed Int, Unsigned Int.

Bootloader Support:

You can use the EEPROM Editor on any type of project. However, you cannot use it for both bootloader and bootloadable. PSoC Creator will indicate an error if such a condition occurs.

For multi-application bootloaders, the EEPROM is equally divided amongst all bootloadables.

Rules:

1. If you have set up a bootloader/multi-application bootloader project to use EEPROM, then no bootloadable project using that bootloader is allowed to use EEPROM (the DWR in the bootloadable project will not allow you to turn on EEPROM for that project).

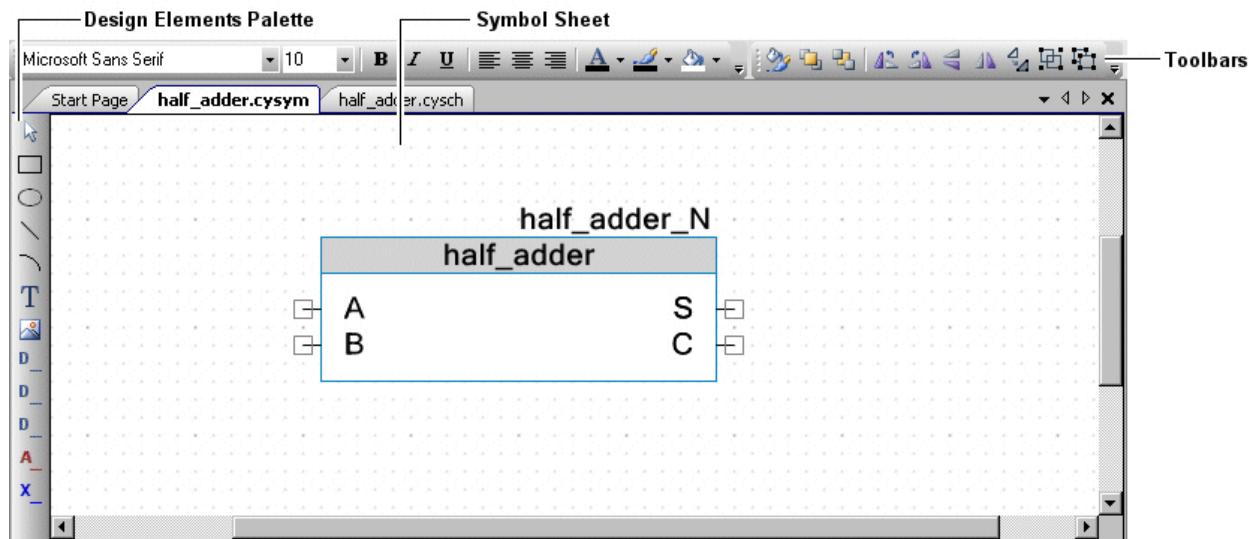
2. If the bootloader is not using EEPROM and it is not a multi-application bootloader, then any bootloadable projects using that bootloader may enable EEPROM in their DWR and use the full EEPROM memory region.
3. If you have a multi-application bootloader and that bootloader is not using EEPROM, then bootloadables using that bootloader may use EEPROM. The EEPROM will be divided evenly amongst the bootloadable .cyacd files (e.g. *_1.cyacd has the first half of EEPROM and *_2.cyacd has the upper half of the EEPROM data).

See Also:

- [Design-Wide Resources](#)
- [Memory Window](#)
- Bootloader/Bootloadable Component datasheet (available from the [Component Catalog](#))

Symbol Editor

The Symbol Editor allows you to create and edit Components that can then be used in your designs.



The process of creating Components can be complex. These topics are provided as a help if you press [F1] for the various dialogs you may encounter. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).

The main topics of the Symbol Editor include:

- symbol sheet – the canvas on which you draw Components
- [Design Elements Palette](#)
- [Symbol Editor Context Menus](#) – commands specific to the Symbol Editor
- [Common toolbars](#) – commands common to the design entry tools

This section also contains various topics related to working with the Symbol Editor:

- [Creating a Symbol](#)
- [Symbol Wizard](#)
- [Symbol Editor Context Menus](#)
- [Working with Component Terminals](#)
- [Defining Catalog Placement](#)
- [Creating Symbol Parameters](#)
- [Creating Parameter Validators](#)

Creating a Symbol

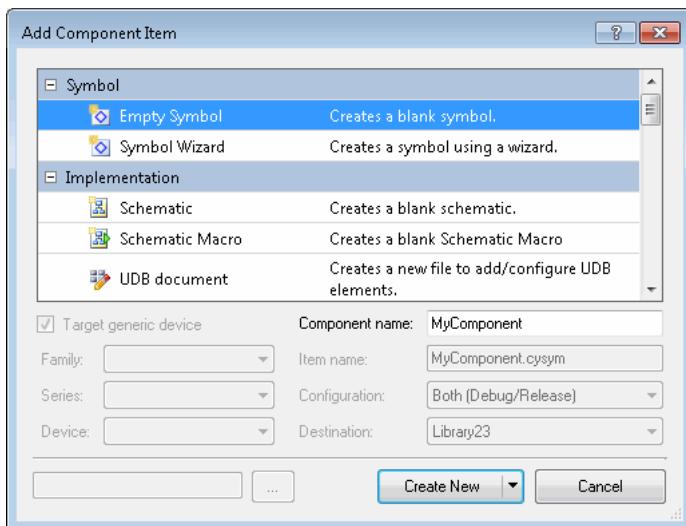
Creating a symbol is only one aspect of creating a Component. The process of creating Components can be complex. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).

You create a symbol within a project. So first, either [create a new project](#) or [open an existing one](#).

1. Once you have opened a project, select the **Components** tab in the [Workspace Explorer](#).
2. Right-click on a Component or project, and select **Add Component Item...**

You can also select this option from the **Project** menu.

The Add Component Item dialog displays.



3. Select the Empty Symbol icon.
 4. Enter a Component name.
 5. Select the **Destination**.
- Note** This option is only available if you open this dialog from the **Project** menu.
6. Click Create New.

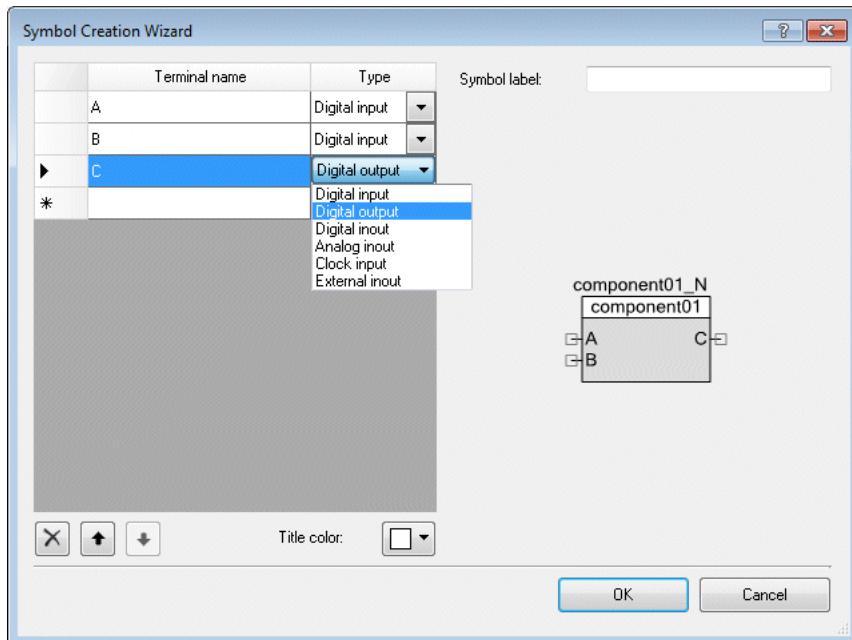
The symbol is added to the existing Component or a new Component is created with the new symbol.

See Also:

- [Component Author Guide](#)
- [Library Component Project](#)
- [Adding a Component Item](#)
- [Symbol Wizard](#)
- [Create Symbol from Schematic](#)

Symbol Wizard

The Symbol Wizard allows you to create a basic symbol and specify the names and types of terminals. The wizard also offers a preview of the symbol, as well as the option to change the title color.



Creating a symbol is only one aspect of creating a Component. The process of creating Components can be complex. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).

To Open the Symbol Wizard:

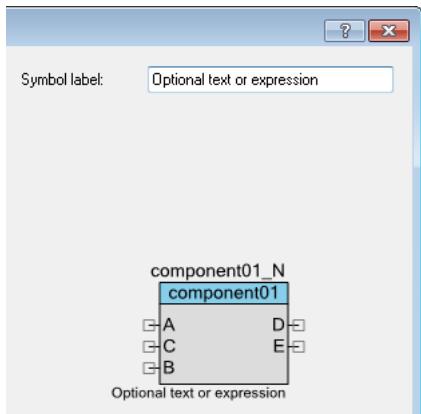
1. Open a project (or create a new project) and select the **Components** tab in the [Workspace Explorer](#).
2. Right-click on a Component or project, and select **Add Component Item...** to open the [Add Component Item](#) dialog.
3. Select the Symbol Wizard icon and click **Create New**.

To Add a New Terminal:

Enter the **Terminal name** and select a **Type** for each Component terminal; create separate terminals using different rows.

To Add a Symbol Label:

Use the **Symbol label** text box to add optional text or an expression to be displayed under the symbol.



Note This field is normally used to enter an expression to display the value of a single parameter for the Component in the symbol (for example, Param name = `=\$ParamName`). See [Using Text Substitution](#) for more information about expressions.

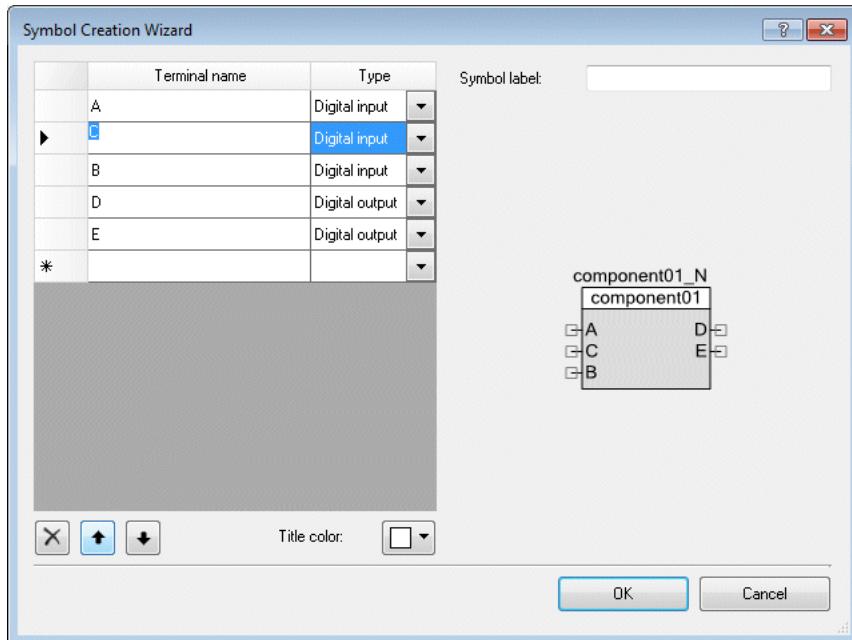
To Delete a Terminal:

Double-click the row header cell for a terminal. A dialog will display to confirm the deletion.

You can also select a row and click the **Delete**  button.

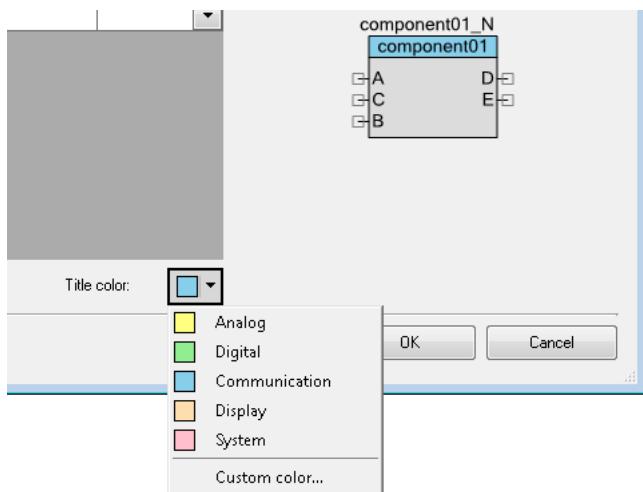
To Change Terminal Order:

Select a row and use the **Up Arrow** and **Down Arrow** buttons to rearrange the terminal order in the table. This will also affect how the terminals will appear on the symbol, as shown in the preview.



To Change the Title Color:

To change the color of the symbol title, select a color from the **Title color** pull-down menu. There are several pre-defined Cypress colors. You may also choose a custom color.



See Also:

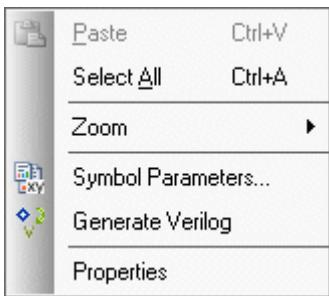
- [Component Author Guide](#)
- [Library Component Project](#)

- [Adding a Component Item](#)
- [Creating a Symbol](#)
- [Create Symbol from Schematic](#)
- [Using Text Substitution](#)

Symbol Editor Context Menus

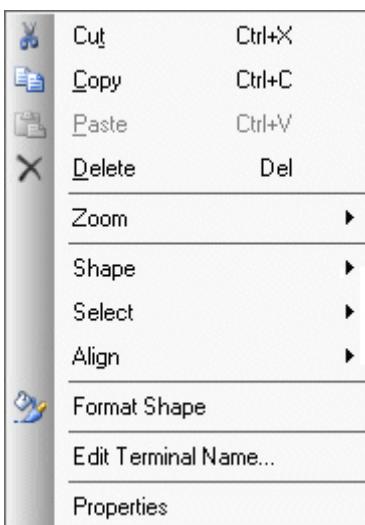
The [Symbol Editor](#) contains various commands available on right-click – or context – menus. The commands available will vary depending on whether you right-click on the canvas itself or on a Component/element. The following are the commands available:

On Canvas:



- **Paste** – Same as command from the [Standard Toolbar](#).
- **Select All** – Selects everything on the canvas.
- **Zoom** – Same as zoom commands from the [View Menu](#).
- **Symbol Parameters** – Opens the Parameters Definition dialog. See [Creating Symbol Parameters](#).
- **Generate Verilog** – Generates a Verilog file based on the Symbol's definition. See [Generate Verilog](#).
- **Properties** – Opens the [Properties dialog](#).

On Selected Object(s):



- **Cut, Copy, Paste, Delete** – Same as commands from the [Standard Toolbar](#).
- **Select All** – Selects everything on the canvas.
- **Zoom** – Same as zoom commands from the [View Menu](#).
- **Shape** – Same as shape commands from the [Common Design Entry Toolbars](#).
- **Select** – Allows you to select a specific object when two or more objects are drawn on top of each other.
- **Align** – When two or more objects are selected, this command allows you to align selected shapes: left, right center, top middle, and bottom.
- **Edit Name and Width** – For terminals only, opens [Terminal Name dialog](#).
- **Format Shape** – Opens the [Format Shape](#) dialog to change various characteristics for the selected shape(s).

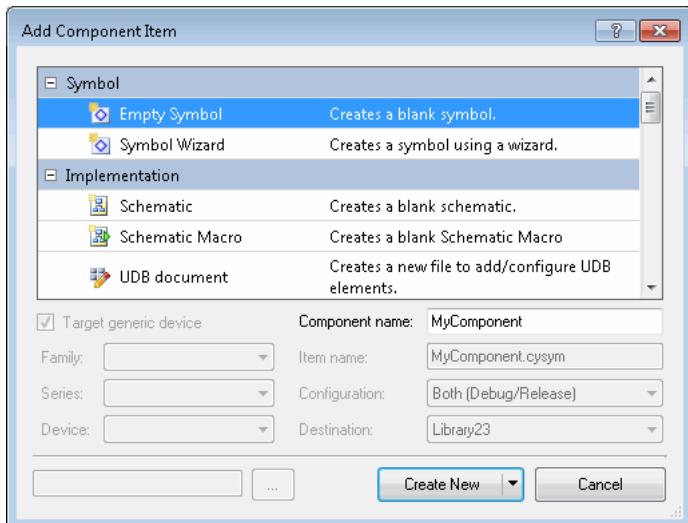
- **Properties** – For terminals only, opens the [Properties dialog](#).

See Also:

- [Symbol Editor](#)
- [Common Design Entry Toolbars](#)
- [Design Elements Palette](#)
- [Standard Toolbar](#)

Adding a Component Item

The primary purpose of the Add Component Item dialog is to add new items to a Component. It can also be used to create a new Component within a library project. Adding Component items and creating Components can be a complex set of instructions and procedures. This topic is provided as a help for this dialog. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).



The dialog provides templates of the different types of items you can add to a Component. The options available will vary depending on the type of item you select and whether you are adding an item to a Component or creating a new Component.

To Open the Dialog:

In the [Workspace Explorer](#), under the **Components** tab, right-click on a Component or project or select **Add Component Item...** 

- Select a Component  to add a new Component item within the Component.
- Select a project  to create a new Component within the project.

You can also use the **Project** menu to open this dialog; however, the **Destination** field does not get selected automatically as it does for the context menu.

To Use the Dialog:

The Add Component Item dialog contains the following sections:

Templates

The **Templates** area displays icons for the different types of available Component items. Currently the available templates include:

- **Symbol** – Contains the symbol templates to create an empty symbol or to use the Symbol Wizard. See also [Creating a Symbol](#).
- **Implementation** – Contains the implementation templates. See also [Creating a New Schematic](#).
- **API** – Contains the API templates to create C, header, and assembly files.
- **Library** – Contains a set of library templates to allow Component authors to add libraries for different compiler tool-chains and configurations (DEBUG/RELEASE).
- **Misc** – Contains miscellaneous templates to create a documentation file, [Control File](#), XML file or any other type of miscellaneous file.

Beside each template, there is a brief description for each Component item.

Target generic device

The **Target generic device** check box disables/enables the **Family**, **Series**, and **Device** pull-down menus. These options determine where in the project hierarchy the new item will be stored. This field is only available for certain types of Component items, such as implementations, API files, etc. The check box will become active for different template items you select.

- Select the **Target generic device** check box to create the selected Component item at the top-level of the Component and disable the other options; de-select to enable them.
- Choose a **Family** to create the Component item in a subfolder for a family (e.g., PSoC3, PSoC4, or PSoC5.).
- Choose a device **Series** to create the Component item in a subfolder for a series of devices (e.g., CY8C32, PSoC 4000, PSoC 4200 BLE, CY8C52LP, etc.).
- Choose a specific **Device** part number to create the Component item in a subfolder for a specific device.

Component Name

The **Component Name** field allows you to specify a name for the Component when you create a new Component for a project. This field is disabled when adding Component items at the Component level.

Item Name

The **Item Name** field allows you to specify a name for some Component items. The following Component items derive their name from the **Component Name**:

- Symbol
- Schematic

- Control file
- Verilog file

Configuration

The **Configuration** field only applies to Library template files. It allows you to specify if the library is for debug mode, release mode, or both.

Destination

The **Destination** field is only active when you open this dialog using the **Add Component Item** command from the **Project** menu. This field is a pull-down menu that allows you to select the project or Component to which you want to add the Component item. Notice that if you select a project, the **Component Name** field becomes active. Conversely, if you select a Component, the **Component Name** field becomes inactive.

Create New vs. Add Existing

The **Create New** button displays by default. If you click this button, PSoC Creator will create a new file of the selected type and it to the Component.

This button contains a pull-down toggle  that switches it to **Add Existing**. If you switch it, a navigation button [...] becomes active to select an existing file of the selected type. Then when you click the **Add Existing** button, the selected file will be copied to the Component folder and added to the Component. Files may be renamed during the copy process to ensure they comply with file naming restrictions imposed on certain file types.

See Also:

- [Component Author Guide](#)
- [Library Component Project](#)
- [Symbol Wizard](#)
- [Create Symbol from Schematic](#)
- [Workspace Explorer](#)
- [Creating a New Schematic](#)
- [Creating a Symbol](#)

Working with Component Terminals

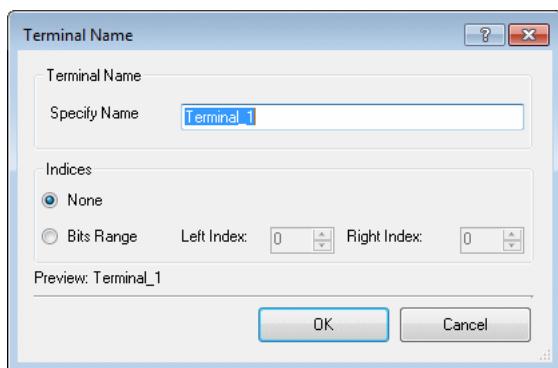
The terminal tools in the [Design Elements Palette](#) for schematics allow you to draw digital input, output, and inout, as well as analog and external terminals.



Use Component terminals when you are creating a symbol in the [Symbol Editor](#), as part of creating a Component. For more information about creating Components, refer to the [Component Author Guide](#).

To Place a Terminal:

Select the appropriate **Terminal** tool from the Design Elements Palette and click on the canvas. The Terminal Name dialog will display.



To Rename a Terminal:

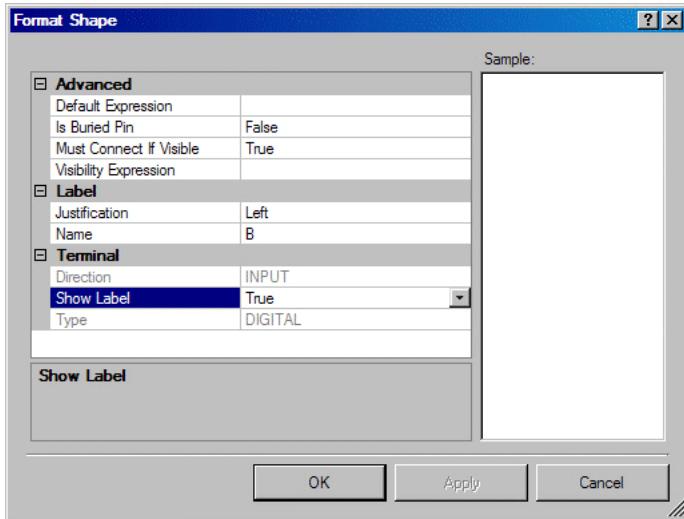
Right-click a terminal and select **Edit Terminal Name**.

The Terminal Name dialog will display.

Use the dialog to specify the terminal name and/or indices, as appropriate.

To Show/Hide a Terminal Label:

1. Right click the terminal and select **Format Shape** to open the Format Shape dialog.



2. Change the **Show Label** property to true/false to show/hide the label, respectively.
3. Click **OK**.

To Delete a Terminal:

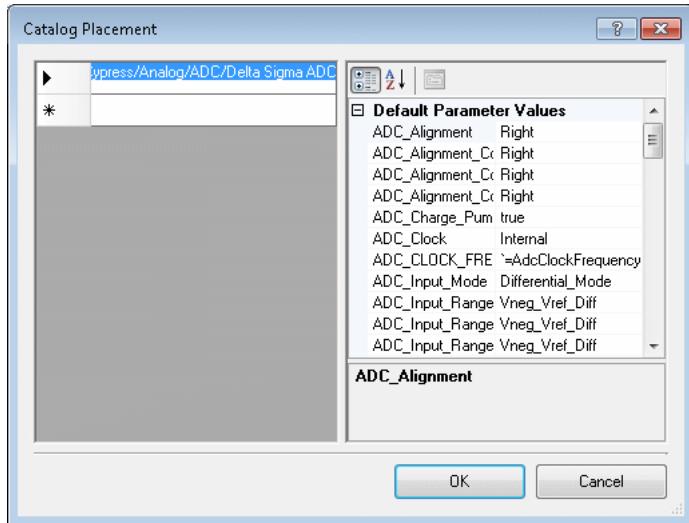
Select the terminal and press [**Delete**] or click .

See Also:

- [Component Author Guide](#)
- [Symbol Editor](#)
- [Design Elements Palette](#)
- [Terminal Name](#) dialog
- [Format Shape](#) dialog

Defining Catalog Placement

As part of creating a Component, you can use the Catalog Placement dialog to define how symbols are displayed in the [Component Catalog](#) under various trees and tabs.

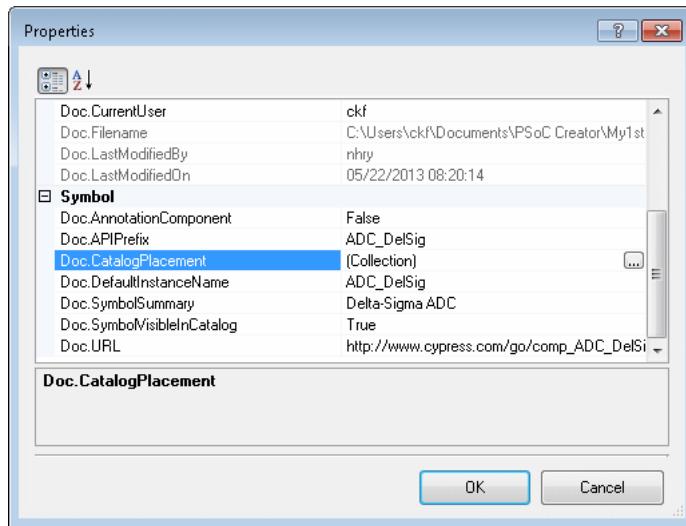


The process of creating Components can be complex. This topic is provided as a help if you press [F1] for this dialog. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).

To Open the Catalog Placement Dialog:

1. Right-click on the symbol canvas and select **Properties**.

The Properties dialog displays.



2. Click the [...] button in the **Doc.CatalogPlacement** field to open the Catalog Placement dialog.

To Define Catalog Placement:

Use the following syntax:

t/x/x/x/x/d

Each Component of the syntax has the following meaning:

- t – The tab name. The tab order displayed in the Component Catalog is alphabetical and case insensitive.
- x – A node in the tree under the tab. You must have at least one node.
- d – The display name for the symbol (optional).

If you do not specify the display name, the symbol name will be used instead; however, you must use the t/x/ syntax.

If you do not define the **Doc.CatalogPlacement** property for a given symbol, it will display by default under the **Default** tab.

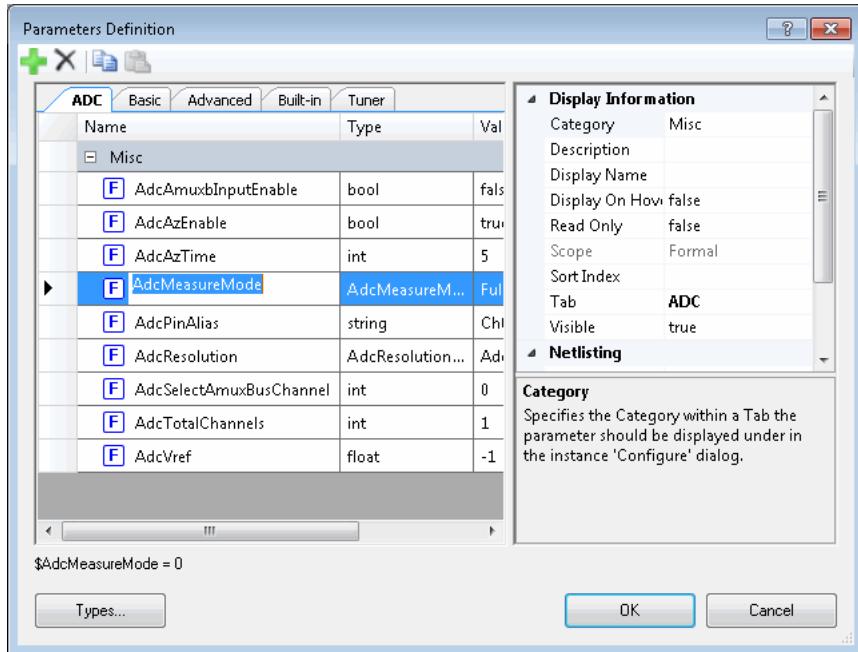
If you want to show the symbol in multiple catalog trees, enter separate syntax strings in different rows of the dialog.

See Also:

- [Component Author Guide](#)
- [Library Component Project](#)
- [Component Catalog](#)
- [Properties dialog](#)

Creating Symbol Parameters

When you create a Component, you can create one or more parameter collections for your symbol using the Parameters Definition dialog.



The dialog presents the parameters in the same order (tabs, categories, and parameters) that they will be presented to the user when editing the instance, with a few minor exceptions related to visibility. All parameters are visible in this dialog no matter what the visibility is set to.

- If a parameter is set to not be visible in the instance Configure dialog, it will not be displayed to the user when configuring the instance.
- If there are no visible parameters in a tab or category, that tab or category will not be displayed.
- If there is only a single visible category on a tab, the category will not be displayed; instead the parameters will be displayed without hierarchy.

The process of creating Components can be complex. This topic is provided as a help if you press [F1] for this dialog. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).

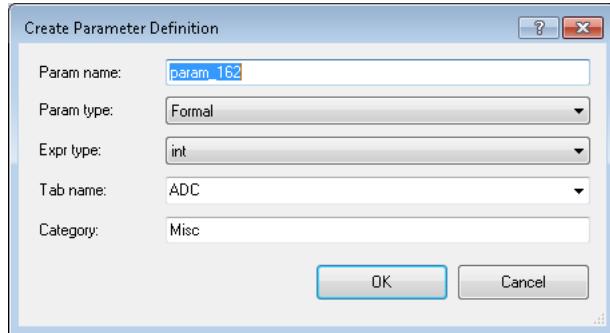
To Open the Dialog:

Right-click on the symbol canvas and select the **Symbol Parameters** icon .

To Create Parameters:

For each symbol, you can create any number of parameters.

1. Click the **Add** button  to create a new parameter definition.



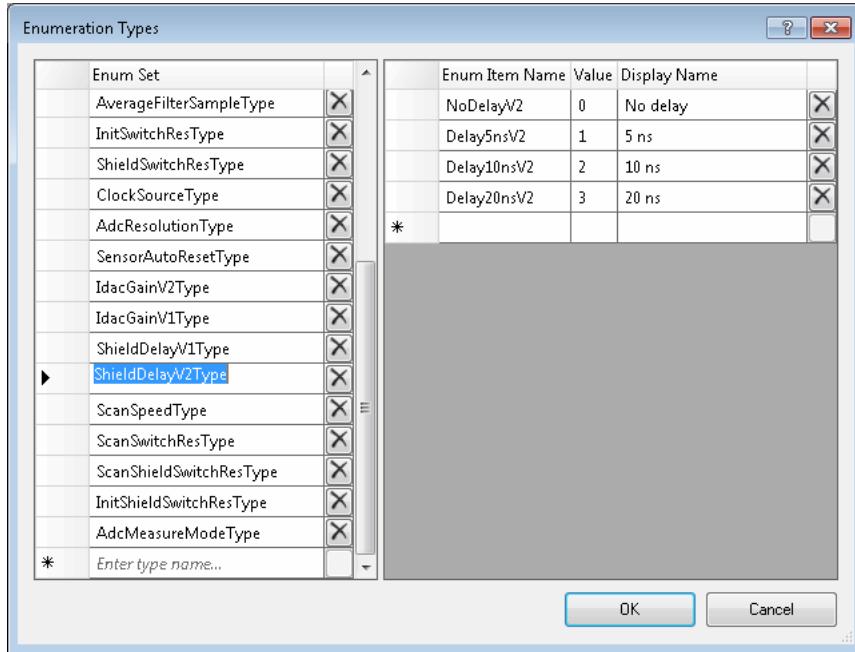
2. For each parameter, define the properties on the right side of the dialog.

Note Multiple parameters can have their properties edited at once by using [Ctrl] or [Shift] while clicking in the table to select multiple rows at once.

3. To add another parameter, repeat the process or use the Copy/Paste buttons.

To Create Enumerated Types:

Click the **Types...** button to open the Enumeration Types dialog.



See [Enumeration Types](#) for more information.

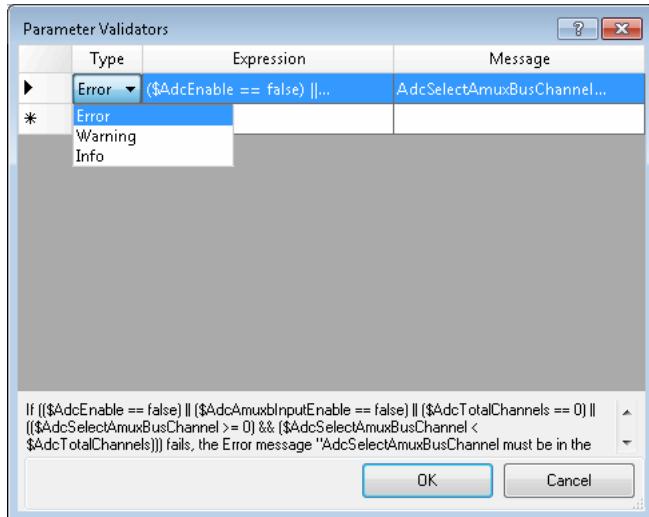
See Also:

- [Component Author Guide](#)

- [Symbol Editor](#)
- [Enumeration Types](#)

Creating Parameter Validators

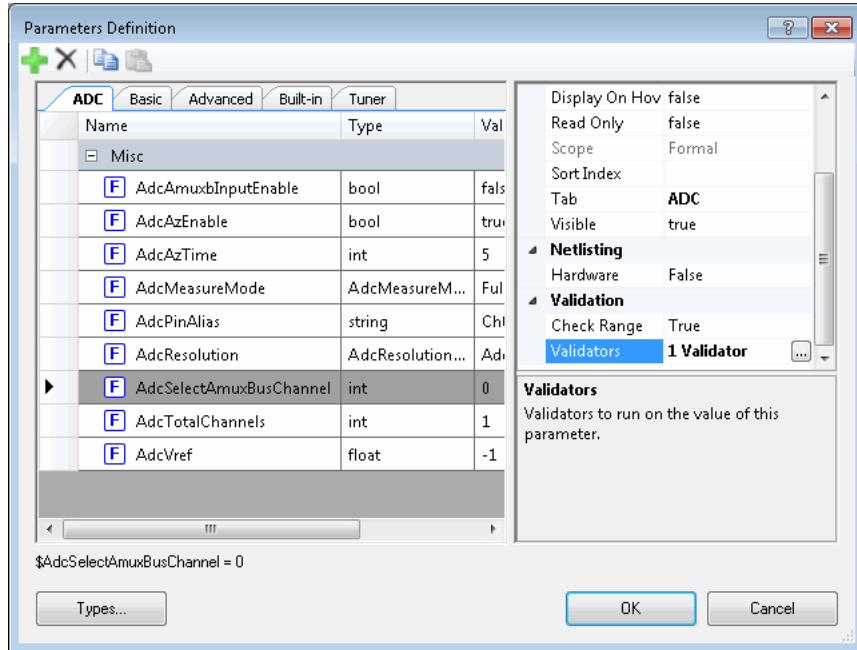
When you create a Component, you can create one or more parameter validators using the Parameter Validators dialog.



The process of creating Components can be complex. This topic is provided as a help if you press [F1] for this dialog. For more in depth discussion regarding creating Components, refer to the [Component Author Guide](#).

To Open the Parameter Validators Dialog:

1. On the [Parameters Definition dialog](#), select the parameter for which to add a validator.
2. Then select the **Validators** field under the **Validation** section, and click the ellipsis [...] button.



To Add a Validator:

1. Select the message type from the drop down in the first row, either Error, Warning, or Info.
2. Type an expression in the **Expression** field. Refer to the [Component Author Guide](#).
3. In the **Message** field, type in the message to display if the validation check is not met.
4. To add another validator, select the message type from the drop down in the next row, and repeat the process.
5. Click **OK** to close the dialog.

See Also:

- [Component Author Guide](#)
- [Symbol Editor](#)
- [Creating Symbol Parameters](#)

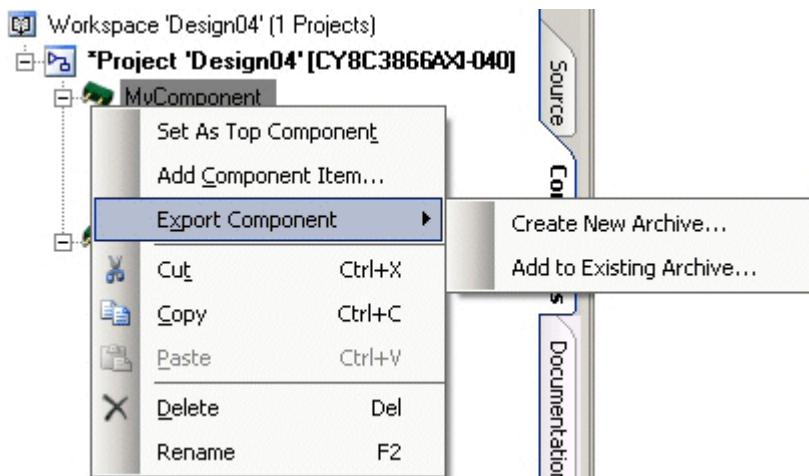
Exporting a Component

After you have finished developing one or more Components, you can export them without providing the entire project or library. This allows for a library-free distribution method of Components and allows you to more easily control your own libraries and dependencies.

If you prefer to include the Component as part of a project/library, see [Archiving a Workspace/Project](#).

To Export a Component:

1. In the [Workspace Explorer](#), under the **Components** tab, right-click on the Component to view the context menu, and select **Export Component**.



2. Select **Create New Archive...** to export it as a new Component archive.
 - This option opens the Save As dialog to save the Component with a default .cycomp file name of *comp_archiveXX*, where XX is a number that will increment each time you export a Component.
 - Rename the file as appropriate and select a desired location.
 - Click **Save**.
3. Select **Add to Existing Archive...** to update an existing exported Component archive. This option allows you to update an existing Component, and it allows you to add additional Components to an existing .cycomp file.
 - This option opens the Save As dialog.
 - Navigate to the location of the existing .cycomp file to be updated.
 - Select the .cycomp file and click **Save**.

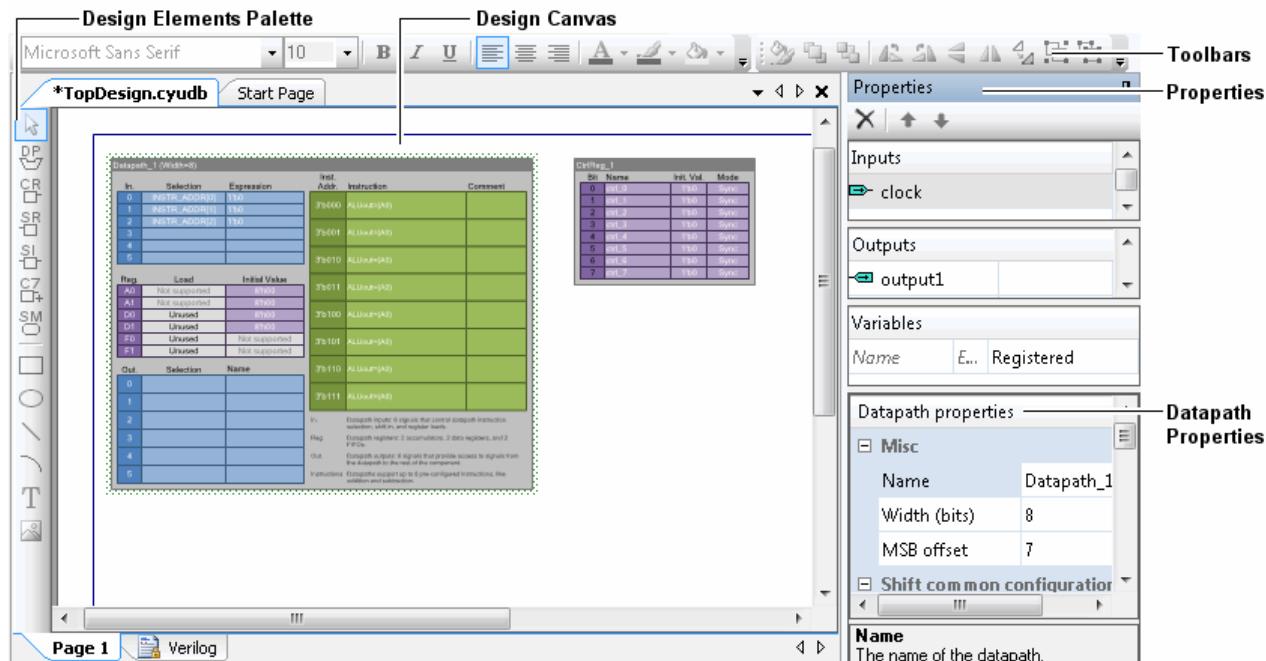
The .cycomp file contains all of the files included with the selected Components, and it is used by PSoC Creator during the [Import Component](#) process.

See Also:

- [Archiving a Workspace/Project](#)
- [Workspace Explorer](#)
- [Import Component](#)

UDB Editor

The Universal Digital Block (UDB) Editor is a graphic tool to create PSoC Components. This simple-to-use editor provides an approachable way to implement and configure UDB resources.



Note The UDB Editor is not intended to enable 100% of UDB functionality or to make the most optimal designs possible. Rather, its purpose is to increase your ability to use UDB resources. This editor does not replace the Datapath Config Tool, which is more of an expert-level tool for working with UDBs. Refer to the [Component Author Guide](#) for more information about the Datapath Config Tool and UDB resources. Refer to the UDB Editor Guide for more information about the UDB Editor.

The main Components of the UDB Editor include:

- Design Canvas – the canvas on which you draw designs
- Verilog – Displays Verilog code generated from the design canvas. The UDB Editor currently supports Verilog; additional languages may be added in the future. All expressions must confirm to Verilog syntax.
- [UDB Design Elements Palette](#)
- [UDB Properties](#)
- [Common Design Entry Toolbars](#) – commands common to the design entry tools
- [Context Menus](#) – commands available by right-clicking

The UDB Editor provides the following UDB features as editable blocks:

- [Datapath \(DP\)](#)
- [Control Register](#)
- [Status Register](#)

- [Status Interrupt Register](#)
- [Count7](#)
- [State Machines](#)

See Also:

- [Component Author Guide](#)

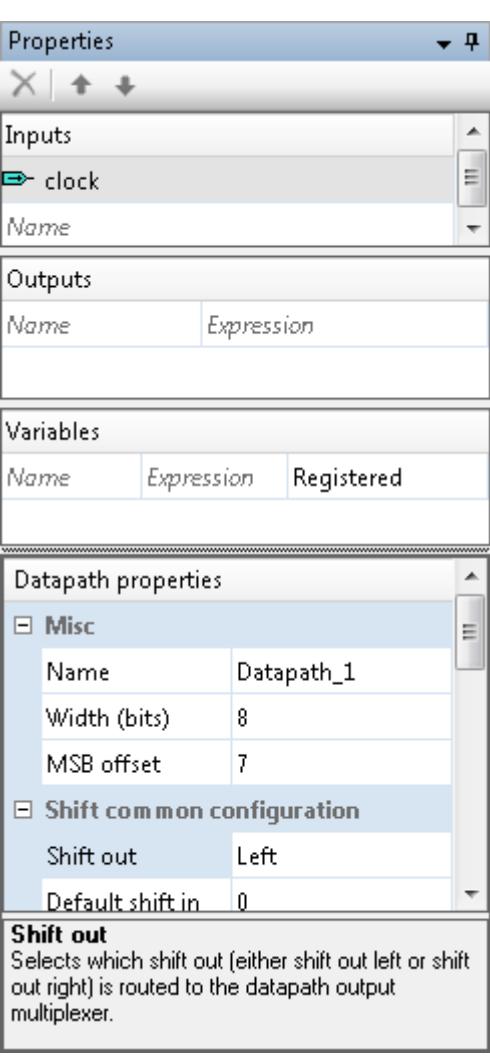
UDB Design Elements Palette

This vertical toolbar is the same as the [Design Elements Palette](#) used for other design entry tools. The main difference is that this toolbar contains the UDB resources to place on the [UDB Editor](#). The buttons are as follows:

-  [UDB Datapath](#)
-  [UDB Control Register](#)
-  [UDB Status Register](#)
-  [UDB Status Register Interrupt](#)
-  [UDB Count7](#)
-  [UDB State Machine](#)

UDB Properties

On the right side of the UDB Editor, there are several sections for properties.



The screenshot shows the UDB Properties panel with the following sections:

- Inputs:** A table with one row named "clock".
- Outputs:** A table with two columns: "Name" and "Expression".
- Variables:** A table with three columns: "Name", "Expression", and "Registered".
- Datapath properties:**
 - Misc:** Table with rows for Name (Datapath_1), Width (bits) (8), and MSB offset (7).
 - Shift common configuration:** Table with rows for Shift out (Left) and Default shift in (0).
 - Shift out:** A note explaining the function of the shift out setting.

- **Inputs** – This is a list of inputs to the Component (equivalent to [schematic input terminals](#)). The clock input is always present and cannot be modified. All other inputs may be added, removed, or renamed as needed.
- **Outputs** – This is a list of outputs to connect to the Component (equivalent schematic output terminals). They operate exactly like inputs, except there are no fixed / unchangeable defaults.
- **Variables** – These allow you to create named and reusable combinatorial and registered logic. Separate icons are used to indicate whether the variable is combinatorial or registered.
- **Datapath Properties** – These apply specifically to the [datapath](#) as a whole (and not to a particular instruction). These properties include:
 - Misc
 - Shift common configuration
 - Shift configurationA / configurationB
 - Configurable comparator inputs
 - Masks
 - FIFOs

Toolbar:

The toolbar contains the following commands:

- **Delete Row** – Deletes the selected row in one of the properties tables.
- **Move Up/Down** – Moves the selected row up or down in the table.

To Add an Input/Output/Variable:

Click in the cell showing "Enter Name", type a name and press [**Enter**].

To Delete an Input/Output/Variable:

Click on a row and press the [**Delete**] key or click the [**Delete Row**] button.

To Edit Datapath Properties:

1. You must have a datapath resource placed on the canvas. Select it to enable the properties.
2. Click in a cell for a particular property, and either select a value from the pull-down menu or enter a value in the field.

See Also:

- [Component Author Guide](#)
- [UDB Editor](#)
- [UDB Datapath](#)
- [Working with Schematic Terminals](#)

UDB Datapath

The UDB datapath element is used to configure datapath resources in the PSoC device. This element contains several tables to edit Inputs, Registers, Outputs, and Instructions.

Datapath_1 (Width=8)					
In.	Selection	Expression	Inst. Addr.	Instruction	Comment
0	INSTR_ADDR[0]	1'b0	3'b000	ALUout=(A0)	
1	INSTR_ADDR[1]	1'b0	3'b001	ALUout=(A0)	
2	INSTR_ADDR[2]	1'b0	3'b010	ALUout=(A0)	
3			3'b011	ALUout=(A0)	
4			3'b100	ALUout=(A0)	
5			3'b101	ALUout=(A0)	
			3'b110	ALUout=(A0)	
			3'b111	ALUout=(A0)	
Reg.	Load	Initial Value			
A0	Not supported	8'h00			
A1	Not supported	8'h00			
D0	Unused	8'h00			
D1	Unused	8'h00			
F0	Unused	Not supported			
F1	Unused	Not supported			
Out.	Selection	Name			
0					
1					
2					
3					
4					
5					

In. Datapath inputs: 6 signals that control datapath instruction selection, shift in, and register loads.

Reg. Datapath registers: 2 accumulators, 2 data registers, and 2 FIFOs.

Out. Datapath outputs: 6 signals that provide access to signals from the datapath to the rest of the component.

Instructions Datapaths support up to 8 pre-configured instructions, like addition and subtraction.

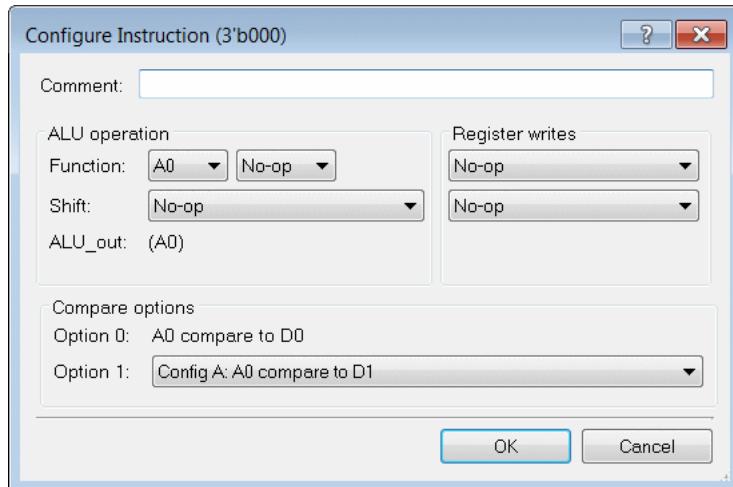
For more information about UDB datapaths, refer to the [Component Author Guide](#).

To Place a Datapath:

Click on the Datapath icon  in the [UDB Design Elements Palette](#), and drag the instance to the canvas.

To Configure Inputs, Registers, Outputs, and Instructions:

- Double-click one of the areas within the Datapath instance to open a Configure dialog for that area.



Note Each area of the Datapath instance contains different configuration dialogs. See [Configure Dialog Descriptions](#) for more information.

- Select a value from the pull-down menu or enter a value in the field, as appropriate.
- Click **OK** to close the dialog.

To Configure General Datapath Properties:

To change the datapath name and other properties, make sure the datapath instance is selected to enable the [properties](#) on the right side on the design canvas.

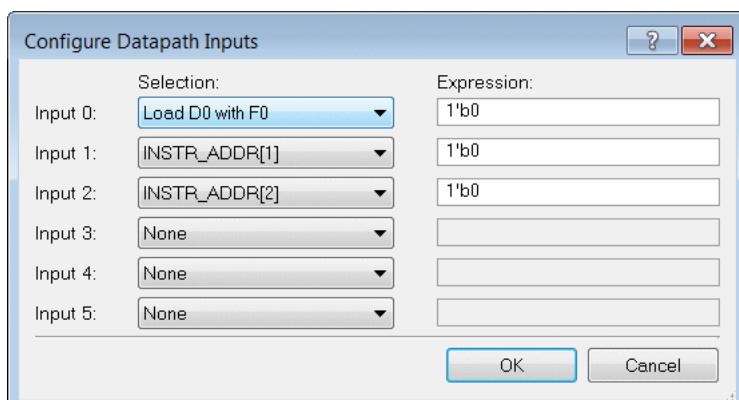
There you can edit each of the properties by selecting or typing a value.

Configure Dialog Descriptions:

The Datapath element contains the following Configure dialogs:

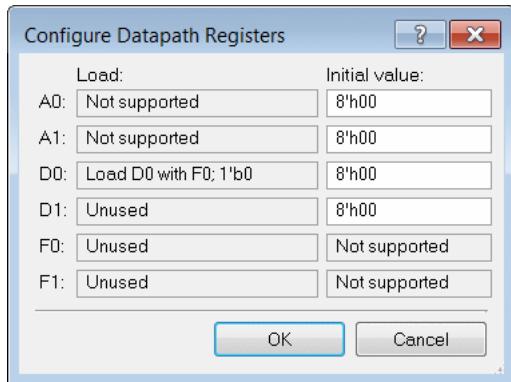
Inputs

This dialog contains fields to select a datapath input and assign an input expression. There can be a total of six inputs for a datapath. All supported values can be entered at the same time using this dialog.



Registers

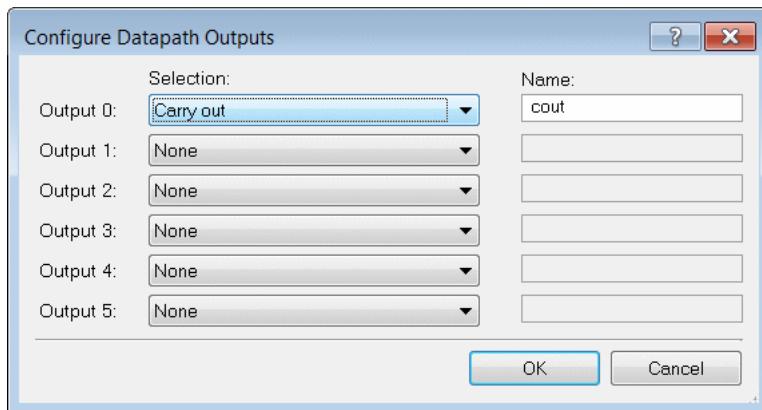
This dialog contains fields to enter initial values for load registers. All supported values can be entered at the same time using this dialog.



Note The values in the **Load** column come from the Input dialog selections; they are included as read-only fields for reference.

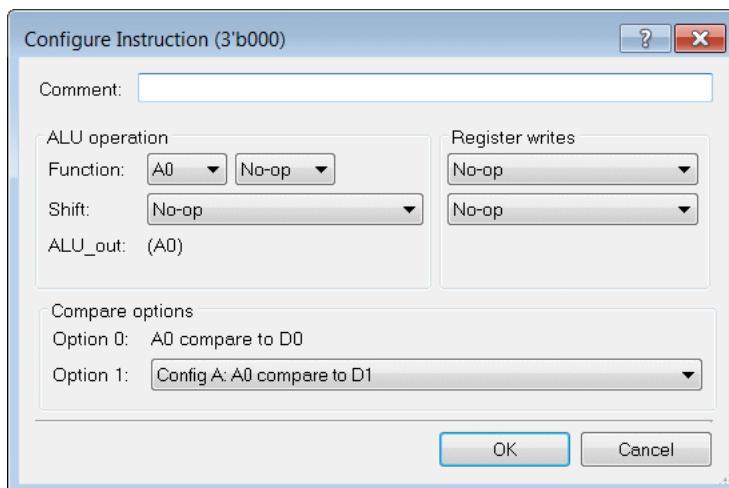
Outputs

This dialog contains fields to enter datapath output and assign a condition. There can be a total of six outputs for a datapath. All supported values can be entered at the same time using this dialog.



Instructions

This dialog contains fields to enter various datapath instructions. There can be a total of eight instructions for a datapath, and each instruction must be configured individually.



Each instruction is divided into three parts: ALU operation, Register writes, and Compare options. The ALU operation determines what arithmetic or Boolean operation is performed for that instruction cycle. Register writes are used to load A0 and A1 with values for the next instruction cycle. Compare options are used to set the comparisons being made using comparator0 and comparator1. Refer to the Component Author Guide for more details.

See Also:

- [Component Author Guide](#)
- [UDB Editor](#)

UDB Control Register

Control Registers are used by the CPU to send commands to the Component. Each control register has eight available bits that can be used throughout the design to control the various aspects of the Component operation.

CtrlReg_1			
Bit	Name	Init. Val.	Mode
0	ctrl_0	1'b0	Sync
1	ctrl_1	1'b0	Sync
2	ctrl_2	1'b0	Sync
3	ctrl_3	1'b0	Sync
4	ctrl_4	1'b0	Sync
5	ctrl_5	1'b0	Sync
6	ctrl_6	1'b0	Sync
7	ctrl_7	1'b0	Sync

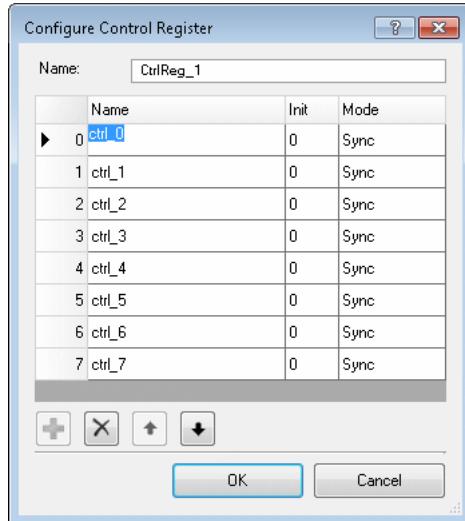
Refer to the [Component Author Guide](#) for more information about control registers and the UDB Editor.

To Place a Control Register:

Click on the Control Register icon  in the [UDB Design Elements Palette](#), and drag the instance to the canvas.

To Configure Bits:

1. Double-click on the Control Register instance to open the Configure dialog.



2. Enter a **Name**, and select the **Init** and **Mode** values.
3. Click **OK** to close the dialog.

See Also:

- [Component Author Guide](#)
- [UDB Editor](#)
- Control Register Component datasheet (available from the [Component Catalog](#))

UDB Status Register

Status Registers are used by the CPU to read hardware signals from the Component. Eight bits are available in a single status register and allows signals from the Component to be seen by the CPU.

StatusReg_1		
Bit	Expression	Mode
0	1'b0	Sticky
1	1'b0	Sticky
2	1'b0	Sticky
3	1'b0	Sticky
4	1'b0	Sticky
5	1'b0	Sticky
6	1'b0	Sticky
7	1'b0	Sticky

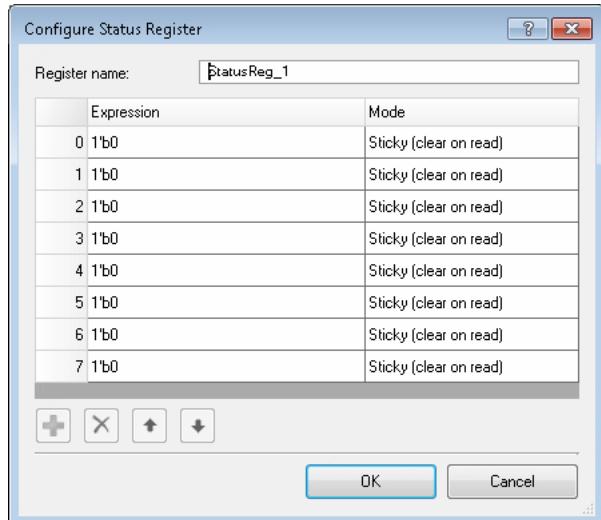
Refer to the [Component Author Guide](#) for more information about status registers and the UDB Editor.

To Place a Status Register:

Click on the Status Register icon  in the [UDB Design Elements Palette](#), and drag the instance to the canvas.

To Configure Bits:

1. Double-click on the Status Register instance to open the Configure dialog.



2. Enter an **Expression**, and select the **Mode** value.
3. Click **OK** to close the dialog.

See Also:

- [Component Author Guide](#)
- [UDB Editor](#)
- Status Register Component datasheet (available from the [Component Catalog](#))

UDB Status Interrupt Register

Status Interrupt Registers are used to generate a maskable interrupt from the status bits. Seven bits are used as the inputs and one bit is used as the interrupt output.

StatusIntReg_1			
Bit	Expression	Mode	Mask
0	1'b0	Sticky	1
1	1'b0	Sticky	1
2	1'b0	Sticky	1
3	1'b0	Sticky	1
4	1'b0	Sticky	1
5	1'b0	Sticky	1
6	1'b0	Sticky	1

Output	Name
Interrupt	StatusIntReg_1_int

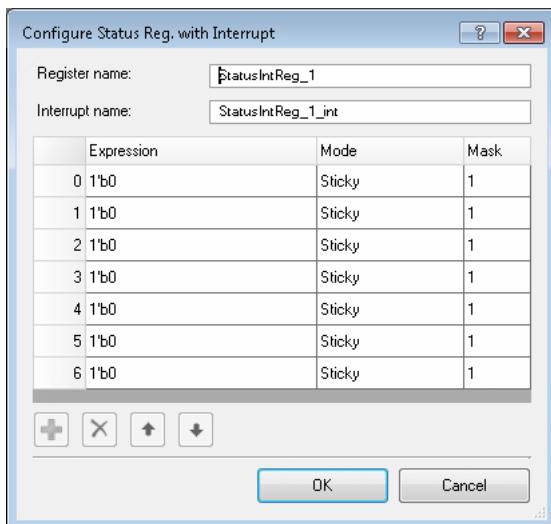
Refer to the [Component Author Guide](#) for more information about status interrupt registers and the UDB Editor.

To Place a Status Interrupt Register:

Click on the Status Interrupt Register icon  in the [UDB Design Elements Palette](#), and drag the instance to the canvas.

To Configure Bits:

1. Double-click on the Status Interrupt Register instance to open the Configure dialog.



2. Enter an **Expression**, and select the **Mode** and **Mask** values.
3. Click **OK** to close the dialog.

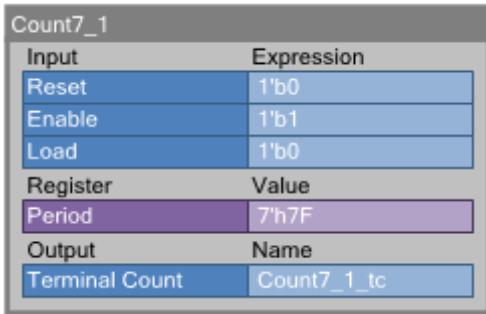
See Also:

- [Component Author Guide](#)

- [UDB Editor](#)
- Status Register Component datasheet (available from the [Component Catalog](#))

UDB Count7

The Count7 counter is a 7-bit down counter that should be used when a counter of three to seven bits is needed. This provides resource savings compared to PLDs or datapath-based counter designs.



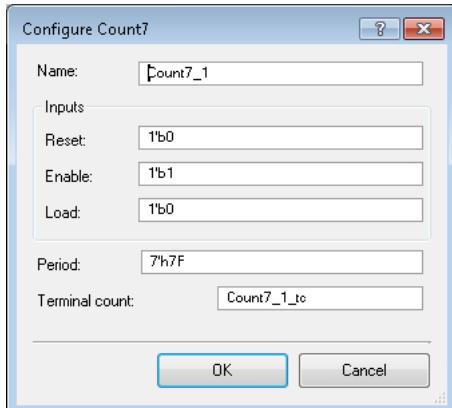
Refer to the [Component Author Guide](#) for more information about the Count7 and the UDB Editor.

To Place a Count7:

Click on the Count7 icon  in the [UDB Design Elements Palette](#), and drag the instance to the canvas.

To Configure Bits:

1. Double-click on the Count7 instance to open the Configure dialog.



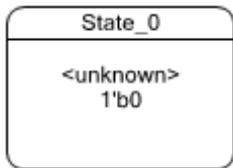
2. Enter a **Name**, **Inputs**, **Period**, and **Terminal** count values.
3. Click **OK** to close the dialog.

See Also:

- [Component Author Guide](#)
- [UDB Editor](#)
- Count7 Component datasheet (available from the [Component Catalog](#))

UDB State Machine

A state machine is control logic implemented using PLDs. It is used to send control signals to the elements in your design and to keep track of the operations happening in your hardware.



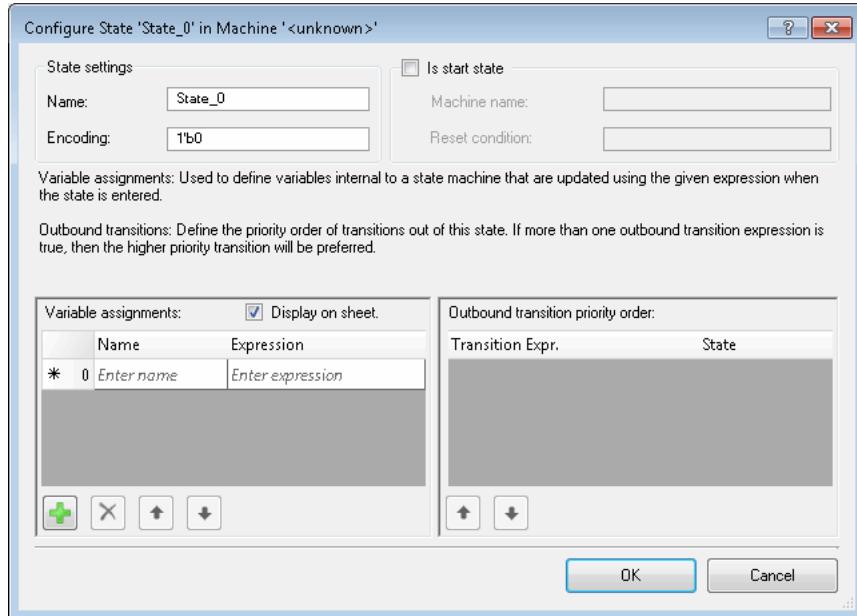
Refer to the [Component Author Guide](#) for more information about state machines and the UDB Editor.

To Place a State Machine:

Click on the State Machine icon  in the [UDB Design Elements Palette](#), and drag the instance to the canvas.

To Configure a State Machine:

1. Double-click on the State Machine instance to open the Configure dialog.



2. Enter information in the various fields.

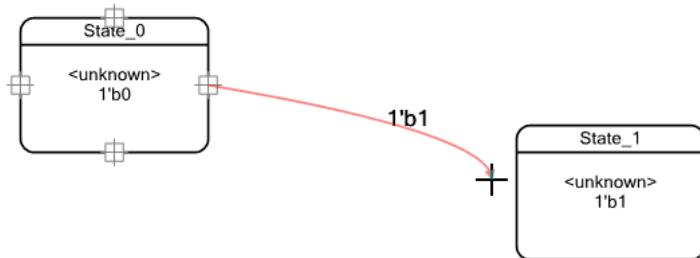
Refer to the Component Author Guide for a complete descriptions of these fields. A few notes:

- **State Name** must be globally unique.
- **Encoding** must be unique within a state machine.
- **Machine name** must be globally unique; it is auto-populated when not a start state.

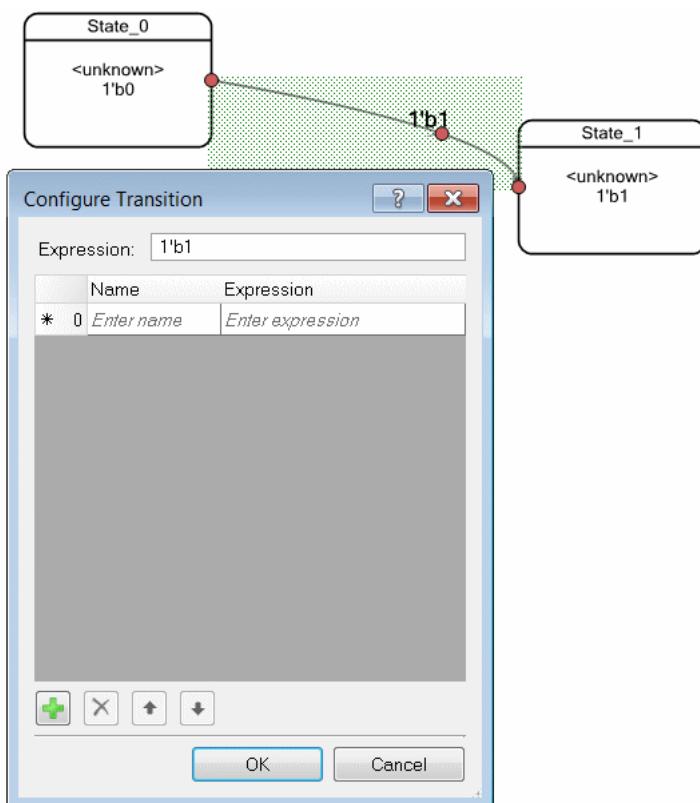
3. Click **OK** to close the dialog.

To Add a State Transition:

1. Hover the cursor over one of the states to view the transition anchor point.
2. Click and drag from an anchor point to begin drawing the state transition.



3. Continue dragging to another state to make the connection. When you release the mouse, a Configure dialog will display.

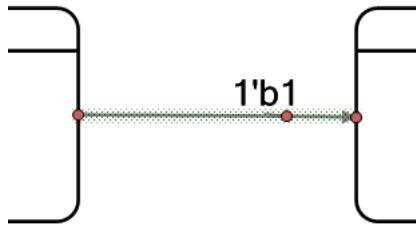


4. Enter the Expression, and add more expressions as needed.
5. Click **OK** to close the dialog.

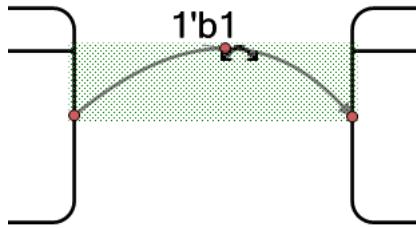
To open the dialog again, double-click on the transition line.

To Adjust the Transition Arc:

1. Click on the transition line to view the arc points.



2. Drag the middle arc point to adjust the arc as appropriate.



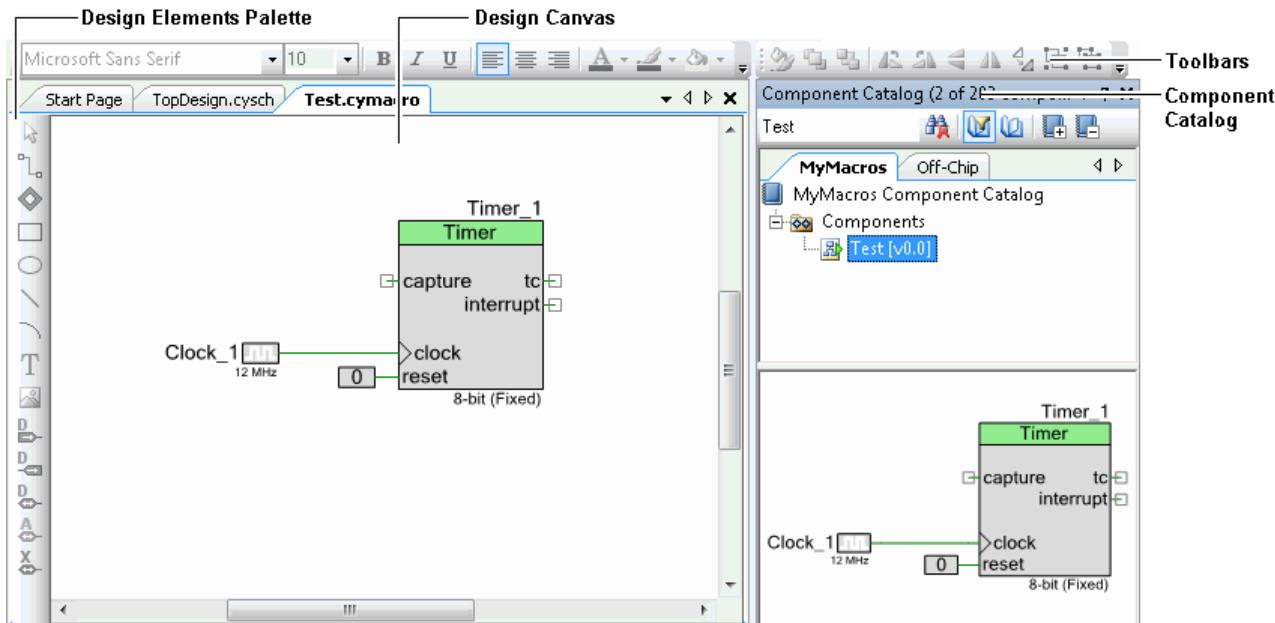
See Also:

- [Component Author Guide](#)
- [UDB Editor](#)

Other Tools

Schematic Macro Editor

The Schematic Macro Editor is similar to the [Schematic Editor](#) in that you use it to create schematics with access to the [Component Catalog](#) and other associated schematic tools. However, the Schematic Macro Editor is also similar to the [Symbol Editor](#) in that you use it to create Component macros that will display in the Component Catalog.



Schematic macros are typically created by Component authors to simplify usage of the Components they build. Typical uses of the Components are prepared and made available as macros. End users use these macros instead of using base Components.

Schematic macros are mini schematics. A Component can have multiple macros. Macros can be at the generic, architecture, family and device levels. Macros can be dragged from the Component catalog and dropped into schematics. Macros can have instances (including the Component for which the macro is being defined), terminals, and wires.

The main Components of the Schematic Macro Editor include:

- design canvas – the canvas on which you draw designs
- [Design Elements Palette](#)
- [Common Design Entry toolbars](#) – commands common to the design entry tools
- [Component Catalog](#) – library of Components to use in your schematic

To Create a Schematic Macro:

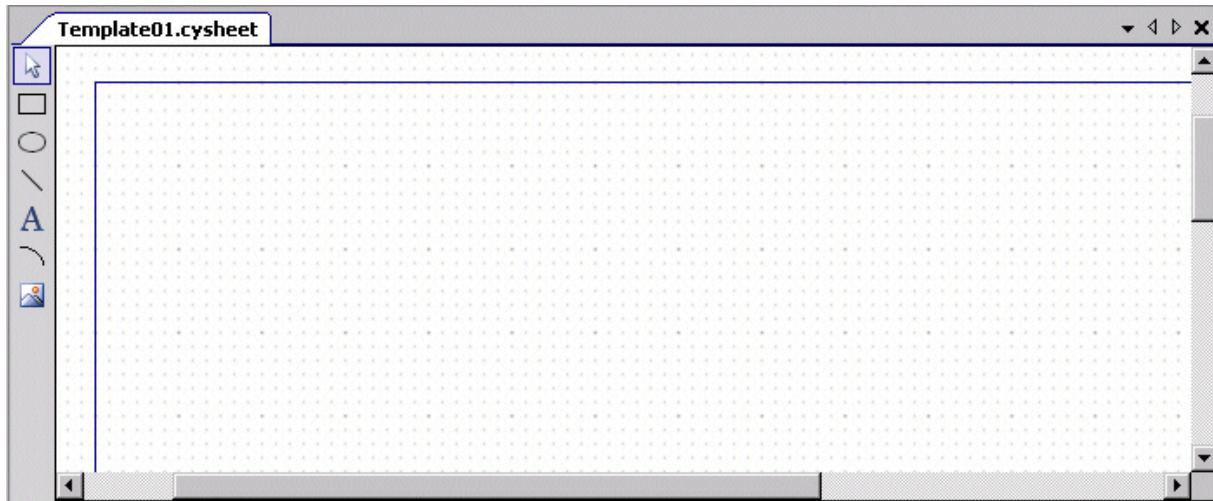
Use the [Add Component Item](#) dialog to add a Schematic Macro to a Component.

See Also:

- [Schematic Editor](#)
- [Symbol Editor](#)

Sheet Template Editor

The Sheet Template Editor allows you to create, edit, and save sheet templates for your schematics.



By creating your own sheet templates, you can add logos, company information, and so on to your template file, and have them display on your designs.

Creating a Sheet Template:



1. Click **File > New File**.
2. On the New File dialog, select the Sheet Template icon, and click **OK**.
The [Sheet Template Page Setup dialog](#) displays.
3. Select the page size, select the page orientation, enter margins, and click **OK**.
A blank sheet template file (.cysheet) opens.

Designing a Template:

You can use all the drawing tools to design the template to meet your specifications.

- Use the Import Graphic tool to import a logo or other images.
- Use the Text tool to type your Company Name, Address, and other important information.
- Use the shape tools to create legends, frame your information, or for any other shapes you might need.

Using Template Properties:

The [Properties dialog](#) contains various fields for information about your template. Right-click on the canvas and select **Properties** to open the dialog.

- The address fields (Addr1, Addr2, Addr3) can be used to enter your company's address.
- The name field can be used to enter your company's name.
- The current user field can be used to enter your name or company ID.
- The DisplayName is where you enter the name of this template. It is also the name displayed in the Sheet Catalog when you create a new executable project.

Saving a Template:

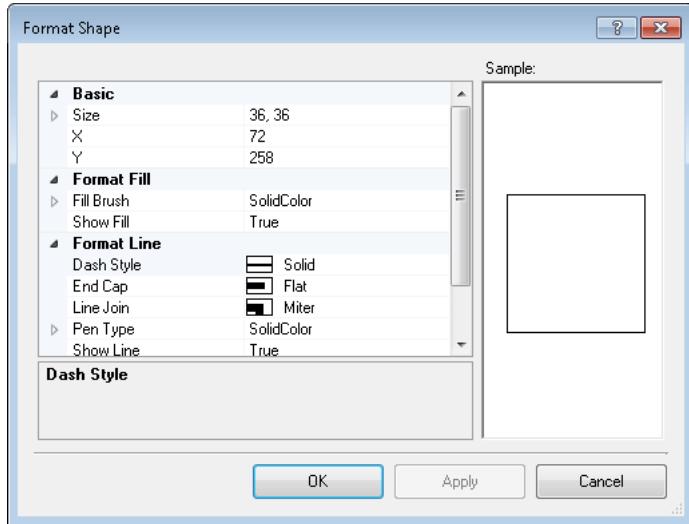
You can save your template anywhere on your PC. The default location for PSoC Creator to locate your template is <INSTALL_PATH>\templates\sheets. If you save your template in another location, specify that location using the PSoC Creator [Design Entry Options](#), under Sheet Templates

See also:

- [Working with Shapes](#)
- [Working with Text](#)
- [Design Entry Options](#)
- [Properties](#)

Format Shape

The Format Shape dialog allows you to change various properties for any shape (or set of shapes) on your canvas.



The types of properties available will vary depending on the selected shape(s). For example, text offers font, color, size, etc., while a line offers width, pen type, end cap, etc.

Common Shape Properties:

Most of the common shape properties in this dialog are self-explanatory, and they are similar to the shape formatting you will see in word-processing or drawing programs. For most of these properties, you select a value from a pull-down menu.

Advanced Shape Properties:

For some shapes, such as instance terminals, there are some advanced properties, including:

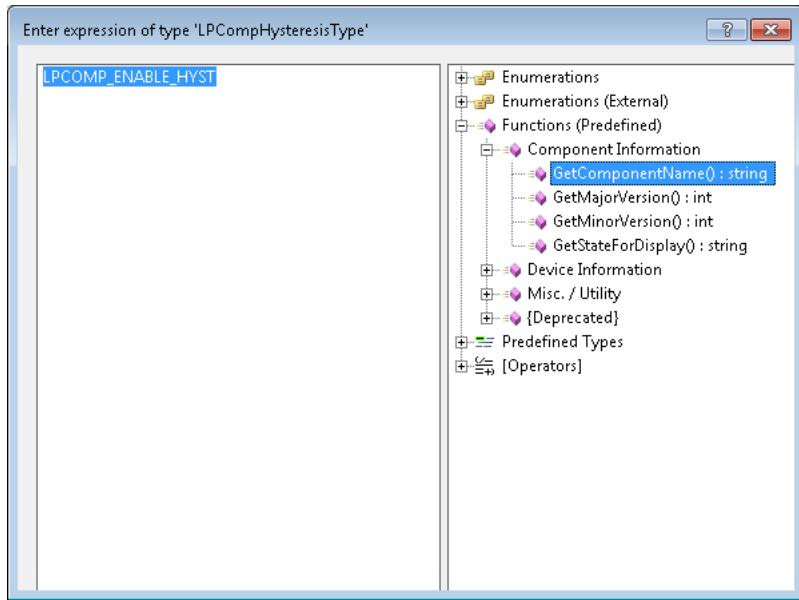
- **Default Expression** – Defines the default value for the shape expressed as <size>'b<value>, where <size> = number of bits, and <value> = the binary value.
- **Shape Tags** – Defines one or more tags to be associated with one or more shapes for use with the shape customization code; refer to the [Component Author Guide](#).
- **Visibility Expression** – Defines an expression to evaluate whether or not the selected shape is visible. For example in the UART Component, this property is defined with the variable \$FlowControl for the rts_n and cts_n terminals. If \$FlowControl is set to true in the instance, then these terminals display in the schematic; if false, they are hidden.

See Also:

- [Component Author Guide](#)
- [Working with Shapes](#)
- [Working with Lines](#)
- [Common Design Entry Toolbars](#)

Expression Editor

This dialog is used to help enter complex expressions.



The text entry on the left supports multiline and the available expression entries are presented in a tree on the right. Hovering over an item in the tree will provide its description. Double-clicking an item in the tree will replace the selected text in the left with the expression that was clicked on. Note that the expected type of the expression is displayed in the title area of the dialog.

To Open the Dialog:

Click the **Expression** button  from either the [default instance Configuration dialog](#) or from the symbol file [Parameters Definition dialog](#).

Common Design Entry Toolbars

There are two common toolbars used with graphical design-entry editors, such as the [Schematic Editor](#) and [Symbol Editor](#): Formatting and Shape Formatting.



Note Many of these commands are also available from the [Schematic Editor Context Menus](#) and [Symbol Editor Context Menus](#).

Formatting:

The following table lists and describes the design entry formatting commands.

Note These commands apply to text labels only. To adjust text properties for the code editor, see the [Options Dialog](#).

Icon	Command	Description
--	Font Style	Menu to select the font family and size.
--	Font Size	Menu to select the font size.
B	Bold	Make selected text bold.
<i>I</i>	Italic	Make selected text italic.
<u>U</u>	Underline	Underline selected text.
	Align Left	Align selected text left.
	Align Center	Align selected text center.
	Align Right	Align selected text right.
	Font Color	Choose color for selected text.
	Line Color	Choose line color (or none) for selected object(s).
	Fill Color	Choose fill color (or none) for selected object(s).

Shape Formatting:

The following table lists and describes the design entry shape formatting commands:

Icon	Command	Description
	Format Shape	Open Format Shape dialog to define various characteristics of the selected object(s).
	Bring to Front	Brings selected objects to the top of the canvas.
	Send to Back	Sends selected objects to the bottom of the canvas.

Icon	Command	Description
	Rotate Left	Rotates selected objects to the left.
	Rotate Right	Rotates selected objects to the right.
	Flip Vertical	Flips the orientation of selected objects vertically.
	Flip Horizontal	Flips the orientation of selected objects horizontally.
	Convert to Closed Shape	Groups selected lines and arcs into a closed shape.
	Group	Groups selected objects into a single shape.
	Ungroup	Removes the grouping from a previously grouped set of objects.

See Also:

- [Schematic Editor](#)
- [Symbol Editor](#)
- [Working with Lines](#)
- [Working with Shapes](#)
- [Design Elements Palette](#)

Design Elements Palette

The design elements palette is a vertical toolbar on the left side of the canvas of graphical design-entry editors, such as the [Schematic Editor](#) and [Symbol Editor](#). The palette contains various design elements you can place on your canvas.

Common Elements:

The following table lists and describes the common design entry shape commands available on both the Schematic Editor and Symbol Editor:

Icon	Command	Shortcut **	Description
	Select	[Esc]	Default; allows items to be selected. Also escapes other tools.
	Draw Rectangle	[R]	Used to draw squares and rectangles .
	Draw Ellipse	[E]	Used to draw circles and ovals .
	Draw Line	[L]	Used to draw lines .
	Draw Arc	[C]	Used to draw curved lines .
	Draw Text	[T]	Used to draw text .
	Insert Image	[M]	Used to insert an image onto the canvas. Allowed image formats are: BMP, GIF, EXIF, png, PNG, and TIFF.

** Keyboard shortcuts for shape drawing tools are only active while the sheet canvas document is active.

Schematic Editor Elements:

The Schematic Editor palette contains the following elements:

- **Terminals** – Used to draw digital input, output, and inout, as well as analog and external schematic terminals. See [Working with Schematic Terminals](#) and [Keyboard Shortcuts](#).

Note Schematic terminals are hidden from the DEP when you are editing a top-level schematic. They only appear when creating a schematic implementation for a Component.

- **Wire** – Used to draw wires on the schematic; the shortcut is [W]. See [Working with Wires](#).
- **Sheet Connector** – Used to draw connectors between commonly named wires on multiple sheets; the shortcut is [S]. See [Using Multiple Pages and Connectors](#).

Symbol Editor Elements:

- **Terminals** – Used to draw digital input, output, and inout, as well as analog and external Component terminals. See [Working with Component Terminals](#) and [Keyboard Shortcuts](#).

Normal vs. Sticky Mode:

All commands on the palette except **Select** have two modes: normal and sticky. Normal mode allows you to place the element on the canvas once, while sticky mode allows you to repeat placing an element indefinitely.

- To activate normal mode, single-click an element; to activate sticky mode, double-click an element.
- To escape sticky mode, press [**Esc**] or click **Select** .

See Also:

- [Schematic Editor](#)
- [Symbol Editor](#)
- [Keyboard Shortcuts](#)

Working with Text

The Text tool allows you to write text on your symbol and schematic documents. For basic shapes, such as rectangles, circles, and lines, etc., you can add text to label them. This section covers the basics for creating a text, and it provides more advanced techniques for using text substitutions.

Note Wires and terminals may also have text that is referred to as labels. These wire and wire labels are controlled using specific dialogs. See [Signal Name](#), [Wire Labels and Names](#) and [Terminal Name](#) for more information.

To Create Text:

1. From the [Design Elements Palette](#), select the **Text Tool**. 
2. Click on your schematic or symbol canvas.

A text box appears:



3. Type the text and click on the canvas.

Your text displays as you typed it, surrounded by a dashed box.



To Edit Text:

1. Double-click the text label to edit.

The text becomes selected in the same manner as creating new text.



2. Edit the text as appropriate and click on the canvas.

Your text displays as you edited it, surrounded by a dashed box.

Edited Text

See Also:

- [Schematic Editor](#)
- [Symbol Editor](#)
- [Design Elements Palette](#)
- [Wire Labels and Names](#)
- [Working with Schematic Terminals](#)
- [Working with Component Terminals](#)
- [Using Text Substitution](#)

Using Text Substitution

When creating and editing text boxes in symbols and schematics (see [Working with Text](#)), you can use substitutions to replace the string you type with different types of information. You can enter various expressions (for example, `=1+1`) that will expand to the appropriate value.

To Use Text Substitution:

1. Create a text box.
2. Type in the box, using the following format:
`=<expression>`

The format must use a backward apostrophe [`], equals sign [=], the expression, and closed with another backward apostrophe [`]. If the expression is or contains a parameter, use the dollar symbol [\$] in front of the parameter name.

`=\$<parameter>`

Available Substitution in Documents:

You can have as many substitution points in your text as you would like. You have access to parameters, document properties, and instance names. You also have access to enums, as well as the full expression language. For more information about expressions, refer to the [Component Author Guide](#).

The following are the available parameters visible/available in the various documents:

- Text access to parameters and functions really only makes sense in symbols. There may be a corner case that would make sense in schematics. Access to parameters and functions is not allowed in macros.
- Text in symbols can see all parameters defined in the symbol. It doesn't matter if they're built-in (such as \$INSTANCE_NAME) or user-defined.

- Text in schematics can see all parameters defined in the schematic's symbol. For example, the schematic that implements a UART_v1_20 Component can use `=\$PARAM` to expand UART parameters.
- Text substitution has no semantic influence on the design. It is only available so it can be read by users.
- Text in a [Schematic Editor](#) only has access to the default value of the parameters.
- Text in schematics that do not have symbols cannot see any parameters. You can only use simple expressions, such as `=1+1`.
- Text in [Schematic Macros](#) do not have access to any parameters.
- Text expressions have access to Component-specific functions created via the ICyExprEval_v1 or ICyExprEval_v2 customizer APIs (see Customizer API Reference Guide). The exact same rules apply here as with parameters:
 - in symbols can call functions defined by that Component being defined
 - in schematics can call functions defined by the Component being implemented
 - in schematic macros cannot be called at all

Substitution Examples:

The following sections provide different examples for variables you can use.

Symbol Parameters

On symbols, you can refer to any parameter you define (but you'll get the default value). When your symbol is instantiated in a schematic, you'll see the parameters set by the user on that instance. All parameters must use the \$ sign. For example, parameter p1, type = int, default value = 42, enter the following:

```
`=$p1`
```

The text displayed for this label will be 42.

Document Properties

On symbols and schematics you can refer to a document property. For example:

```
`=$Doc.CurrentUser`
```

The text displayed for this label will be the current user's name or ID. See the [Properties dialog](#) for more information about the different properties available.

Instance Name

On symbols you can refer to the instance name. You'll always get "Inst_N" in the symbol document, but when dropped in a schematic it will have that instance's proper name.

```
`=$INSTANCE_NAME`
```

Note There is a setting in the [Design Entry Options dialog](#) to show or hide unevaluated expressions

Enumerations

On symbols and schematics you can refer to any enumeration. For example, if you define the following enum type:

```
enum Foo
{
    Foo_VAL_1 = 1,
    Foo_VAL_2 = 2
};
```

You would enter the variable for this label as:

```
`=Foo_VAL_1`
```

The text displayed for this label will be Foo_VAL_1.

Notice that the enumeration was converted to a string, and the default string conversion uses the textual name of the enum. If you want to display the numeric value you can do one of two things:

```
`=Foo_VAL_1 + 0`
```

- or -

```
`=cast(int, Foo_VAL_1)`
```

The first method takes advantage of the fact that the + operator forces the left and right to be a number and that 0 is the additive identity. The second method uses the explicit casting operation and forces the enumeration to convert to an int.

Complicated Expressions

You can also do complicated expressions. For example:

```
`="p1=" . $p1 . " Current User=" . $Doc.CurrentUser . " Foo_VAL_1=" . cast(int,
Foo_VAL_1)`
```

The text displayed for this label will be: p1=42 Current User=xxx Foo_VAL_1=1

This example used casting and string concatenation. You can also embed as many substitution strings in your edit text as you want. For example:

```
Life is short, so eat `=$1` donuts and `=Foo_VAL_1 + 0` fig newtons.
```

The text displayed for this label will be: Life is short, so eat 42 donuts and 1 fig newtons.

See Also:

- [Schematic Editor](#)
- [Symbol Editor](#)
- [Working with Text](#)
- [Component Author Guide](#)
- Customizer API Reference Guide
- [Properties dialog](#)
- [Design Entry Options](#)

Working with Lines

The Schematic Editor and Symbol Editor both provide a line tool to draw lines. This section describes various ways to draw and work with lines, including:

- [Drawing a single line](#)
- [Drawing multiple lines](#)
- [Moving a line](#)
- [Resizing a line](#)

See also [Working with Shapes](#).

To Draw a Single Line:

1. Click the **Draw Line** tool. 
2. Move the cursor to the position to start the line.
3. Click and **hold** the mouse button, and drag the cursor to the end point of the line.
4. Release the mouse button.

To Draw Multiple Lines:

1. Double-click the **Draw Line** tool  to enter sticky mode.
2. Move the cursor to the position to start the line.
3. Click and **hold** the mouse button, and drag the cursor to the end point of the first line.
4. Release the mouse button.
5. Move the cursor to the start of the second line.
6. Click and **hold** the mouse button, and drag the cursor to the end point of the second line.
7. Continue clicking dragging and until you are done drawing lines.
8. Press **[Esc]** or click **Select**  to exit sticky mode.

To Move a Line:

1. Click the **Select** tool. 
2. Click and hold the mouse button on the line to move, and drag it to the desired position.

When you hover your mouse over the line, your cursor changes to a finger.

To Resize a Line:

1. Click the **Select** tool. 

2. Click the mouse button on the line to resize.

Notice that its handles display.

- To change either the width or the height, drag a side handle.
- To change both the width and the height, drag a corner handle.

See Also:

- [Using Design Entry Tools](#)
- [Design Elements Palette](#)
- [Working with Shapes](#)

Working with Shapes

When using the Symbol Editor or the Schematic Editor, you can draw many different kinds of shapes. There are many similarities between shapes and lines; however, lines have a few differences. This section covers the basics of working with shapes, including:

- [Drawing a shape](#)
- [Moving a shape](#)
- [Resizing a shape](#)

See also [Working with Lines](#).

To Draw a Shape:

1. Click the tool for the shape to draw. See [Common Design Entry Toolbar](#).
2. Move the cursor to the position to start the shape.
3. Click and **hold** the mouse button, and drag the cursor to the end point of the shape.
4. Release the mouse button.

To Move a Shape:

1. Click the **Select** tool. 

2. Click and hold the mouse button on the shape to move, and drag it to the desired position.

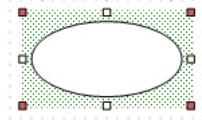
When you hover your mouse over the line, your cursor changes to a finger.

To Resize a Shape:

1. Click the **Select** tool. 

2. Click the mouse button on the shape to resize.

Notice that its handles display.



- To change either the width or the height, drag a side handle.
- To change both the width and the height, drag a corner handle.

See Also:

- [Using Design Entry Tools](#)
- [Design Elements Palette](#)
- [Working with Lines](#)

Zooming

There are different windows, such as the [Schematic Editor](#) and [Symbol Editor](#), that allow you to zoom in and out. For these types of windows there are several useful zoom features you may wish to use:

- [Toolbar](#)
- [\[Ctrl\] Key](#)
- [Right-Click](#)

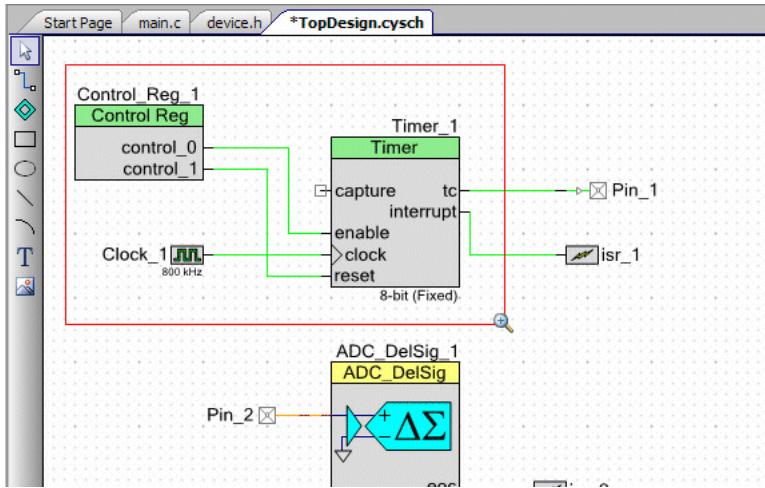
Use the Toolbar:



- To pick a specific zoom percentage, click the **Zoom** pull down menu and select the zoom level you want. You can also manually type any percentage; the available zoom range is 2% to 2038%.
- To zoom incrementally, click **Zoom In** or **Zoom out**, as appropriate.

Use the [Ctrl] Key:

- Press and hold the **[Ctrl]** key and drag a box around the area you wish to magnify. This action draws a red rectangle as you drag the mouse.



Immediately upon releasing the mouse, the canvas will zoom to the selected area.

- Press and hold the [Ctrl] key and use the scroll wheel on your mouse to zoom in and out incrementally.
- Press and hold the [Ctrl] key and press the [+] key to zoom in and the [-] key to zoom out.

Use the Right-Click Menu

Right-click on a canvas and select the appropriate **Zoom** options.

See Also:

- [Standard Toolbar](#)
- [Keyboard Shortcuts](#)

Scrolling

Various windows provide scroll bars to view more information. There are a few ways to scroll:

- Drag the vertical or horizontal scroll bar, as needed.
- Use your mouse scroll wheel to scroll up and down.
- Press and hold the [Shift] key with the mouse scroll wheel to scroll left and right.
- Use the left and right arrows to scroll vertically; up and down arrows scroll horizontally.

Design Entry Reserved Words

The following words are reserved by PSoC Creator Design Entry tools. They cannot be used for names of design elements, such as wires, terminals, instances, parameters, etc.

C/C++	Verilog	VHDL
asm	always	abs
auto	and	access
bool	assign	after
break	attribute	alias
case	begin	all
catch	buf	and
char	bufif0	architecture
class	bufif1	array
const	case	assert
const_cast	casex	attribute
continue	casez	begin
default	cmos	block
delete	deassign	body
do	default	buffer
double	defparam	bus
dynamic_cast	disable	case
else	edge	Component
enum	else	configuration
explicit	endattribute	constant
export	endcase	disconnect
extern	endfunction	downto
false	endmodule	else
float	endprimitive	elsif
for	endspecify	end
friend	endtable	entity
goto	endtask	exit
if	event	file
inline	for	for
int	force	function
long	forever	generate
mutable	fork	generic
namespace	function	group
new	highz0	guarded
operator	highz1	if
private	if	impure
protected	ifnone	in
public	initial	inertial
register	inout	inout
reinterpret_cast	input	is
restrict	integer	label
return	join	library
short	medium	linkage
signed	module	literal
sizeof	large	loop

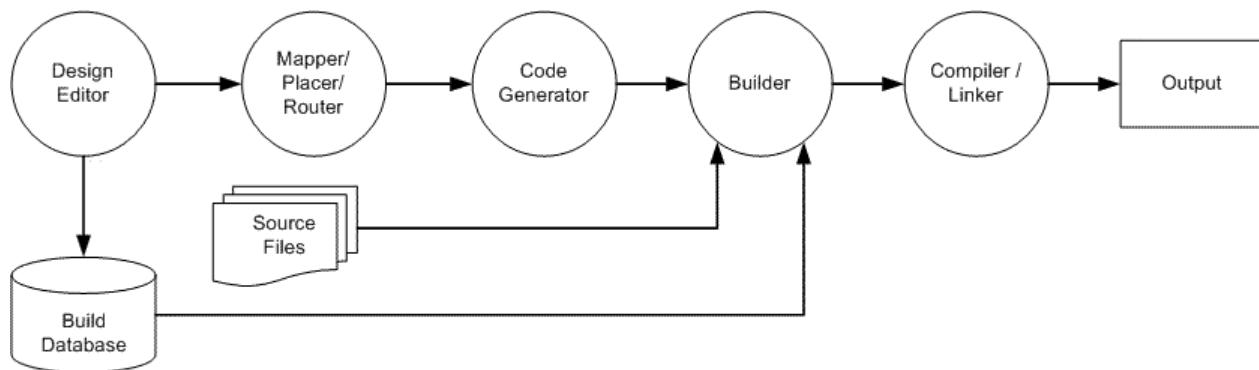
C/C++	Verilog	VHDL
static	macromodule	map
static_cast	nand	mod
struct	negedge	nand
switch	nmos	new
template	nor	next
this	not	nor
throw	notif0	not
true	notif1	null
try	or	of
typedef	output	on
typeid	parameter	open
typename	pmos	or
union	posedge	others
unsigned	primitive	out
using	pull0	package
virtual	pull1	port
void	pulldown	postponed
volatile	pullup	procedure
while	rcmos	process
wchar_t	real	pure
_Bool	realtime	range
_Complex	reg	record
_Imaginary	release	register
	repeat	reject
	rnmos	rem
	rpmos	report
	rtran	return
	rtranif0	rol
	rtranif1	ror
	scalared	select
	signed	severity
	small	signal
	specify	shared
	specparam	sla
	strength	sll
	strong0	sra
	strong1	srl
	supply0	subtype
	supply1	then
	table	to
	task	transport
	time	type
	tran	unaffected
	tranif0	units

C/C++	Verilog	VHDL
	tranif1	until
	tri	use
	tri0	variable
	tri1	wait
	triand	when
	trior	while
	trireg	with
	unsigned	xnor
	vectored	xor
	wait	
	wand	
	weak0	
	weak1	
	while	
	wire	
	wor	
	xnor	
	xor	

5 Building a PSoC Creator Project



When you build a project, PSoC Creator goes through series of processes to produce an output file. For a design project, the output is a hex file used to program a device. For a library project, the output is a library file used to specify Component information. The following image shows the processes at a high level for a design project. A library project would not necessarily include all of these processes.



When you create a project, PSoC Creator sets the tool chain with which to build the output. The tool chain is a collection of tools (code generator, compiler, assembler, linker, etc.) that transforms a project's contents into the appropriate output for the project's type.

Build Configurations:

PSoC Creator provides Debug and Release configurations for these tool chains. Changing between build configurations can help when developing and testing a design. For example, while first developing code it is easiest to use a 'Debug' configuration that typically has fewer optimizations and produces more debug information. This can help in tracking down exactly what is causing an issue. As the code becomes stable and is getting ready for release, using a 'Release' configuration becomes preferable as additional optimizations are often desired. These optimizations help cut down the size of the program and allow it to run faster.

Having multiple configurations available allows for quickly switching between 'Debug' and 'Release' modes if issues are discovered that need investigation. The Build Configuration on the main toolbar allows for changing between different configurations. If this option is not visible, right click on the toolbar area and select Build Configuration. Additionally, the build options for any configuration can be adjusted using the [Build Settings](#) dialog.

Section Topics:

This section covers various aspects of the PSoC Creator build system. It includes the following topics:

- [Build Toolbar Commands](#)

- [Build Menu](#)
- [Build Settings](#)
- [Mapper, Placer, Router](#)
- [Control File](#)
- [Directives](#)
- [Generated Files](#)
- [Source Code Control](#)
- [Static Timing Analysis](#)
- [CyPrjMgr Command Line Tool](#)
- [Keil C51 Compiler](#)
- [Reentrant Code in PSoC 3](#)

Build Toolbar Commands

The Build toolbar contains many of the common commands you will use while building your designs:



The pull-down menu allows access to different build and clean options, shown in the following table.

The Build toolbar contains the following commands:

Menu Item	Icon	Shortcut	Description	See Also
Build (Named) Project		[Shift]+[F6]	Build the selected project.	Building a PSoC Creator Project
Clean and Build (Named) Project			Clean and build the selected project. Note During the clean process, PSoC Creator cleans the build output of only those files that are still part of the project. Any output files generated using source files that are no longer part of the project (deleted or simply removed from project), will be left untouched.	
Clean (Named) Project			Clean the selected project.	
Build All Projects		[F6]	Build all projects in the workspace.	
Clean and Build All Projects			Clean and build all projects in the workspace.	
Clean All Projects			Clean all projects in the workspace.	
Cancel Build		[Ctrl]+[Break]	Cancel the build process.	

Menu Item	Icon	Shortcut	Description	See Also
Compile File		[Ctrl]+[F6]	Compile the selected file.	
Generate Application			Generate API source code files.	Generated Files
Program		[Ctrl]+[F5]	Program the selected device with the code generated from the selected project.	
Debug		[F5]	Start the debugger.	Using the Debugger

See Also:

- [Build Menu Commands](#)

Build Menu

The Build menu contains the following commands:

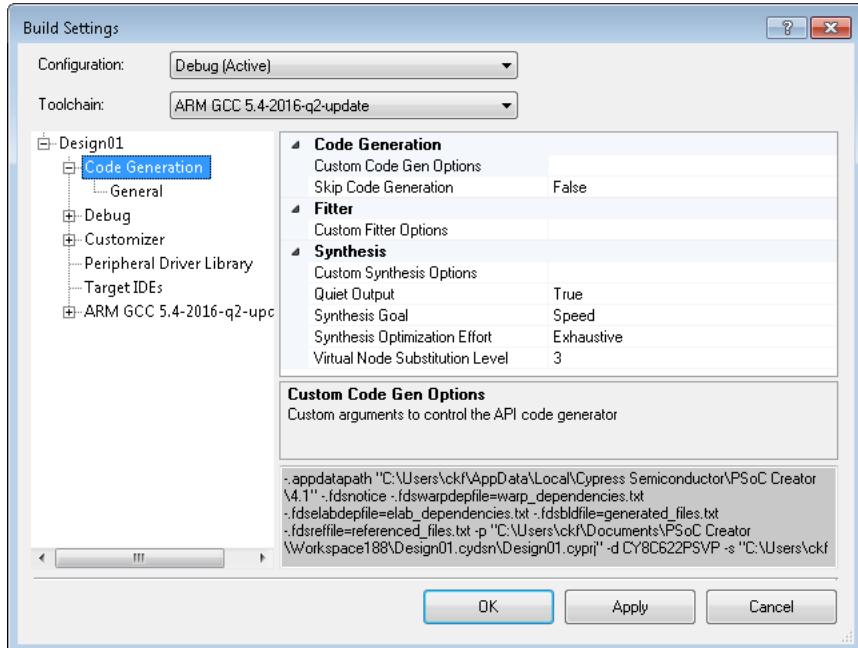
Menu Item	Icon	Shortcut	Description	See Also
Build All Projects		[F6]	Build all projects in the workspace.	Building a PSoC Creator Project
Clean All Projects			Clean all projects in the workspace. Note During the clean process, PSoC Creator cleans the build output of only those files that are still part of the project. Any output files generated using source files that are no longer part of the project (deleted or simply removed from project), will be left untouched.	
Clean and Build All Projects			Clean and build all projects in the workspace.	
Build (Named) Project		[Shift] + [F6]	Build the selected project.	
Clean (Named) Project			Clean the selected project.	
Clean and Build (Named) Project			Rebuild the selected project.	
Cancel Build		[Ctrl] + [Break]	Cancel the build process.	
Compile File		[Ctrl] + [F6]	Compile the selected file.	
Generate Application			Generate API source code files.	Generated Files
Generate Project Datasheet			Generate a project datasheet.	Generating a Project Datasheet

See Also:

- [Build Toolbar Commands](#)

Build Settings

The Build Settings dialog lets you define various build settings on a per-project basis. You can also select a different toolchain for the selected project, and in turn set different settings for the compiler, assembler, and linker.



Build Settings Categories:

This dialog may contain several different categories of settings, depending on the specified PSoC device and project type. These categories include:

- [Code Generation \(this topic\)](#)
- [Debug](#)
- [Customizer](#)
- [Peripheral Driver Library](#) (PSoC 6)
- [Target IDEs](#) (PSoC 6)
- [Toolchain](#)
 - [Assembler](#)
 - [Compiler](#)
 - [Linker](#)

- [User Commands](#)
- [Library Generation](#)

To Open the Dialog:

Right-click on your project and select **Build Settings**.

Build Settings Macros:

The Build Settings dialog contains macros that can be used to substitute names for directories and files. You can use these macros in any Build Settings field where you can type.

Macro	Description	Example Values
<code> \${Config}</code>	Build Configuration.	"Debug" or "Release"
<code> \${ProcessorType}</code>	Name of the processor type.	"CortexM0", "CortexM3", ...
<code> \${Platform}</code>	Name of Toolchain.	"Arm_GCC_541", "Arm_GCC_Generic", ...
<code> \${ProjectShortName}</code>	Name of the project without Extension.	"Design01", Design02", ...
<code> \${ProjectDir}</code>	Directory of the project file relative to current directory.	".\Design01.cyprj", ".\Design02.cyprj", ...
<code> \${ProjectFile}</code>	Path of the project file relative to current directory.	"."
<code> \${WorkspaceDir}</code>	Directory of the workspace file relative to current directory.	"..", "..", ...
<code> \${WorkspaceFile}</code>	Path of the workspace file relative to current directory.	".\Workspace01.cywrk", ".\Workspace02.cywrk", ...
<code> \${OutputDir}</code>	Path of the output directory as specified by in the toolchain's "General" page.	"\${ProjectDir}\\${ProcessorType}\\${Platform}\\${Config}"

Note There is also a `$(CompileFile)` macro that corresponds to the file being compiled when running compile/assemble commands. It is ignored everywhere else, and it should not be used.

Settings Options:

The top of the dialog contains the following pull-down menus:

- **Configuration** – Use this menu to set build setting preferences for the different types of builds: Debug or Release. Debug mode adds logging information for debugging purposes; Release mode does not.

Note This option does not change the build type for your project. It is only used to set preferences. The currently active build type is shown in parentheses. To change the active build type, close the Build Settings dialog, and use the **Configuration** pull-down menu located on the main toolbar.
- **Toolchain** – Use this menu to select the toolchain for the selected project. To specify a default toolchain for all new projects, see [Selecting a Default Compiler](#).
- **Processor Type** – For library projects, use this menu to select the processor type.

Code Generation Category:

The Code Generation category displays by default. This section allows you to specify various code generation options.

Code Generation:

- **Custom Code Gen Options** – Custom arguments to control the API code generator. At this time, there are no arguments exposed to users. This field is internal to Cypress only.
- Skip Code Generation – true or false.

Note Skipping code generation effectively locks the design such that future changes to the schematic, design-wide resources, and so on are not incorporated into the build output.

Fitter:

- **Custom Fitter Options** – Specify custom arguments to control how the design fits into the PSoC device. Some of the options include the following:

- | | |
|-----------------|---|
| -q | : The -q ("quiet") option suppresses the printing of status messages during compilation. This leads to a less cluttered screen when compilation and synthesis are finished. |
| -xor2 | : The -xor2 option passes along any XOR operators found in the design to the fitter to implement as it sees fit. |
| -f (O D T) | : The -f option enables certain global fitter options. -f must be followed (without an intervening space) by one of the arguments O, D or T. The options define the type of flip-flop that should be used by UDB logic (d-type or t-type). O stands for optimal, and will allow warp to pick optimal flip-flop type for your device. |
| -f (P K) | : The -fK option forces the fitter to preserve the user-specified polarity for all outputs. This is the opposite of the -fP option, which optimizes for the optimal polarity. The -fK option is not recommended for most designs, but is useful in certain cases when the user is able to determine the proper polarity for all the signals. |
| -m | : The -m option enables a smart compile of the project Verilog files. Generally, without this option, Warp compiles all the files. When this option is specified, Warp compiles only those files that have been modified since the last compile. |
| -w# | : The -w option specifies the maximum number of warnings that can appear as a result of a single Warp run before Warp quits. |
| -e# | : The -e option specifies the maximum number of non-fatal errors that can occur on a single Warp run before Warp exits. |
| -yg (a s c) | : The -yg option causes Warp to synthesize the design so that UDB Components are optimized for area (a), speed (s) or as combinatorial equations (c). |
| -yv# | : The -yv option controls the amount of information that is reported in the report file. The -yv option should be followed by a digit. The default is 0. Numbers higher than zero produce more verbose report files useful for debugging. By default (with a value of 0), the report file only indicates major events during synthesis. |
| -v# | : The -v option has a numeric argument that controls the aggressiveness of the virtual substitution algorithm. The range of numbers allowed is 0 to 11, where a value of 0 does not perform any virtual substitution and a value of 11 performs virtual substitution even against the better judgement of the algorithm to isolate large combinatorial and compact them in a UDB. |

Synthesis:

- **Custom Synthesis Options** – Specify custom arguments to control HDL synthesis. See also [Directives Editor](#).

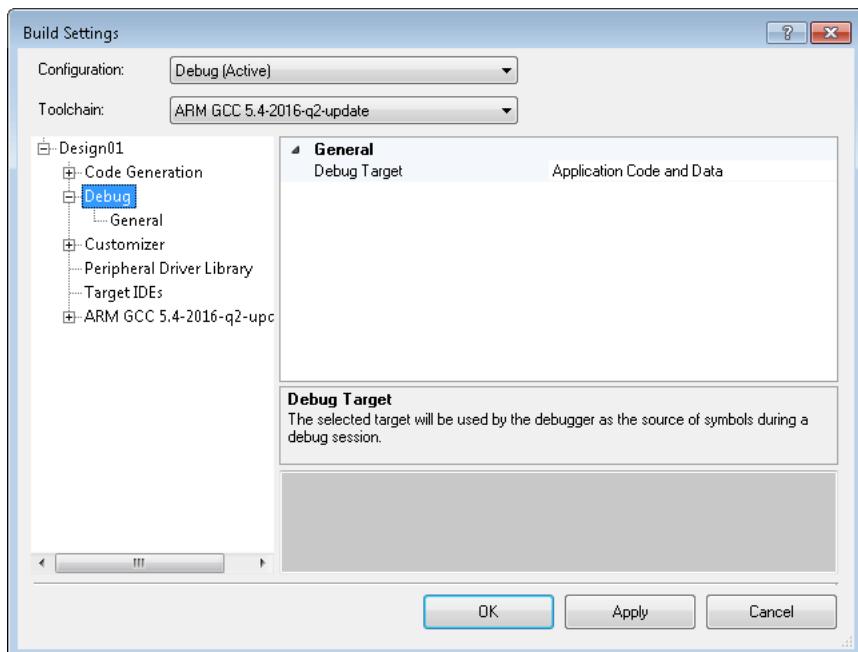

```
-f1 -- no_factor
-o -- opt_level
```
- **Quiet Output** – Control the level of output from the synthesis tool: true or false.
- **Synthesis Goal** – Select the efficiency of the synthesizer: speed or area.
- **Synthesis Optimization Effort** – Select how much effort the synthesizer should put into optimizing the design: none, normal, or exhaustive.
- **Virtual Node Substitution Level** – Used for optimizing designs for size and speed: 0-11.

See Also:

- [Workspace/Project](#)
- [Project Types](#)
- [Directives Editor](#)

Debug Build Settings

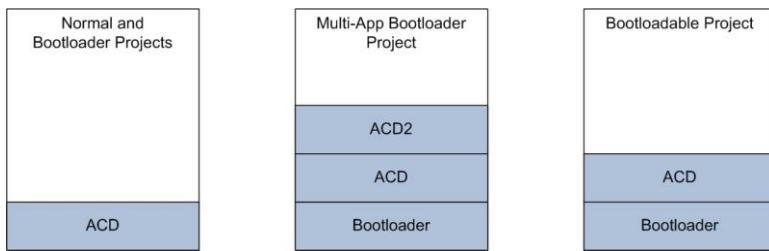
The Debug section of the Build Settings dialog allows for selecting project settings that influence the debugging experience.



Debug Target:

This controls what portion of the design is to be debugged. It contains the following options:

- Application Code and Data – This is the default value; valid for all project types.
 - In normal projects and Bootloader projects, this refers to the application flash.
 - In a Multi-App-Bootloader project, this refers to the first application in Bootloadable flash.
 - In a Bootloadable project, this refers to the Bootloadable flash
- Application Code and Data 2 – Use this option for a Multi-App Bootloader project for debugging the second application of the Bootloadable project.
- Bootloader – Use this option only for debugging the Bootloader part of a Bootloadable project.



Note Because the Bootloader performs a software reset to start the Bootloadable application, there is no way to debug a Bootloadable application from the standard **Execute Code** menu option. Instead, it is recommended that you debug your application before making it a Bootloadable project. If you have already made it a Bootloadable project, you can still use the debugging functionality by programming the design onto the board. Then, reset/power cycle the device and use the [Attach to Target](#) option to debug the Bootloadable application.

Debugging Bootloader/Bootloadable Projects

When App 1 is running, you can debug App 1 by using the **Attach to Target** option. However, the debugger will disconnect if any software reset happens.

When App 2 is running, you can still use the **Attach to Target** option. However, you will not be able to use **Step Into** or **Step Out** commands from the Bootloadable project, because the debugger by default assumes that the code placement is in the App 1 region. You can only use **Halt** and **Run** commands when App 2 is running.

It is possible to debug App 2 of a Bootloadable project using the following steps:

1. Right-click on the Bootloadable project and select **Build Settings** to open the Build Settings dialog.
2. In the "Debug" settings, change the value of **Debug Target** to "Application Code and Data 2" and close the dialog.
3. When "Application Code and Data 2" is set as the **Debug Target**, then commands such as **Step Into** or **Step Out** will work in App2.

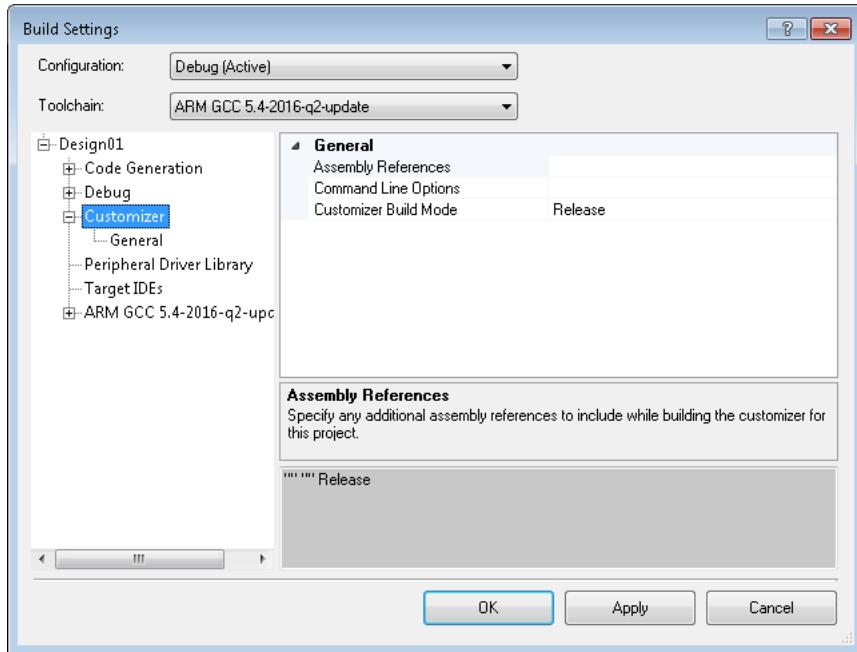
See Also:

- [Build Settings](#)
- [System Reference Guide](#)
- [Attach to Target](#)

- [Debugger Toolbar Commands](#)

Customizer Build Settings

The Customizer section of the Build Settings dialog is used to control various options that determine how the source based customizers for the selected project are built.



General:

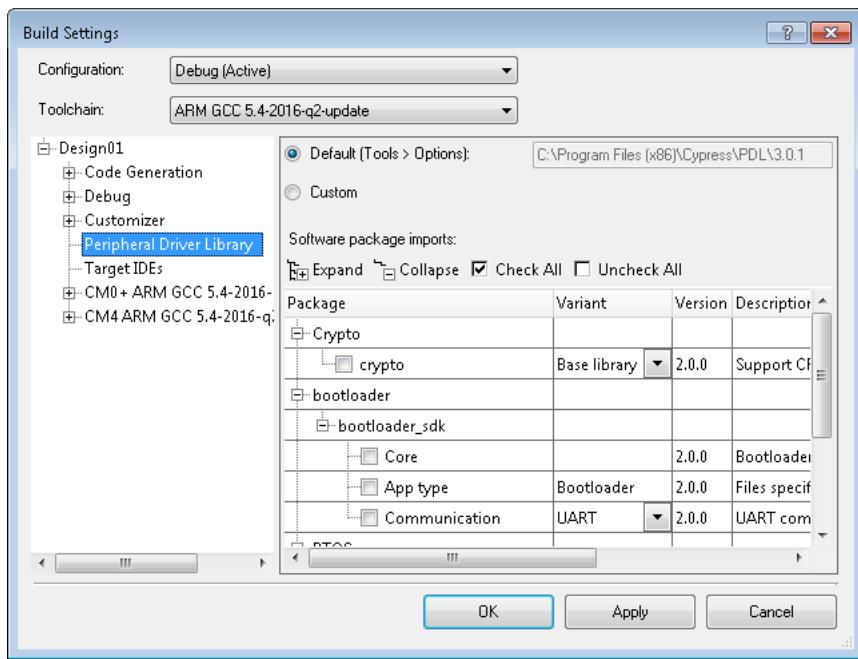
- **Assembly References** – Specify any additional .NET assemblies which need to be included while building the customizer code for the selected project.
- **Command Line Options** – Specify any additional command line options to be given to the compiler while building the customizers for the selected project. The default command line option only defines the "TRACE" constant, which is useful while debugging.
- **Customizer Build Mode** – Specify whether the customizers for this project should be built in "Debug" or "Release" mode. In the "Debug" mode, the customizer will be built with debugging information. This mode should be chosen while debugging customizers.

See Also:

- [Build Settings](#)

Peripheral Driver Library Build Settings

The Peripheral Driver Library section of the Build Settings dialog **applies to PSoC 6 and FM0+ devices only**. Use the options on this page to select the **Default** or **Custom** Peripheral Driver Library (PDL) located on your computer. This allows PSoC Creator to locate the correct copy and version of PDL you wish to include in the build of your design. You can also select any Software Package Imports (i.e., middleware) that may be available.



Default PDL Installation

PDL 3.0.1 is used for PSoC 6 devices and PDL 2.1.0 is used for FM0+ devices. Both versions of PDL may be installed on your computer, depending on the installation options you chose.

The **Default** field on this dialog displays the path of the default PDL that will be used for projects, as specified on the [Options dialog](#) in the **Peripheral Driver Library location** field. If the path is displayed and it is correct, then there is nothing more to be done. However, if the path is not displayed or if it is incorrect, you can select the correct path using the Options dialog.

Custom PDL Installation

In some cases, you may make a local copy of your PDL to modify a driver. And, for a specific design, you may want to use that copy of PDL instead of the default PDL installation. In that situation, select the **Custom** option and navigate to the alternate location of the PDL copy you wish to use.

Software Package Imports

This section is used to select software packages (Middleware, RTOSes) to include in your build. They don't have schematic Component representations, so they can be selected here.

Under **Package**, select the check box for one or more middleware packages to include.

Under **Variant**, select the appropriate variant of the middleware to use, if applicable.

Note Selected software packages require certain drivers to be included in your design. These software packages will display an informational icon and provide a tooltip detailing what PDL driver(s) must be in your design.

This section has the following buttons:

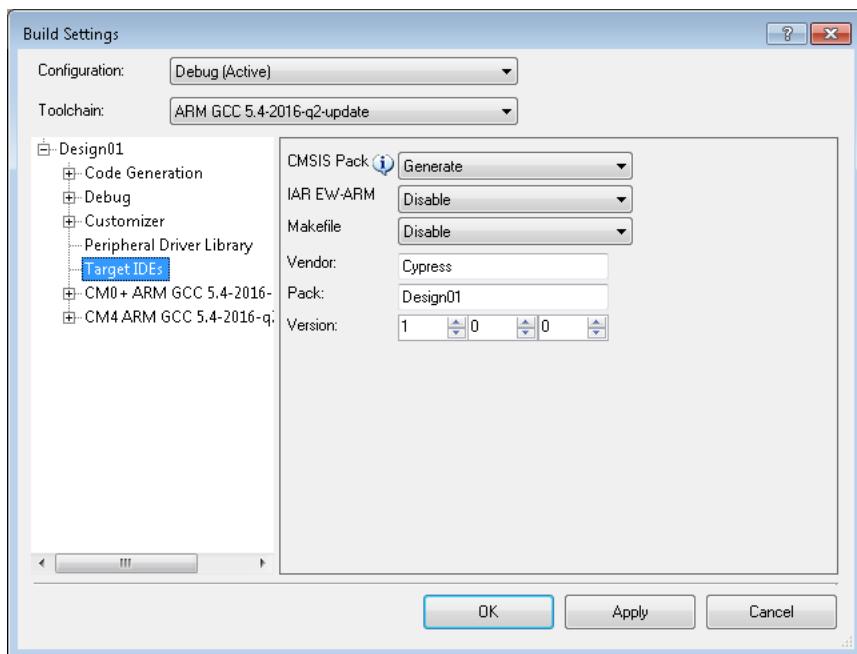
- **Expand/Collapse** – Used to expand/collapse displayed packages.
- **Check All/Uncheck All** – Used to select or unselect all packages.

See Also:

- [Build Settings](#)
- [Options Dialog](#)

Target IDEs Build Settings

The Target IDEs section of the Build Settings dialog applies to **PSoC 6 devices only**. Use the pull-down menus to select one or more IDEs for which to generate files. You will then use these files to further develop the PSoC Creator design in those selected IDEs. For more information, refer to [Integrating into 3rd Party IDEs](#).



IDE Options

This section provides a set of pull-down menus to select one or more IDE for which to generate files. All menus are set to **Disable** by default. The IDEs include:

- **CMSIS Pack** – Applies to Eclipse and Arm MDK. If set to **Generate**, this option generates Component Instance firmware for use in Eclipse. Application firmware is not included. There are also additional fields to define the pack as follows:
 - **Vendor:** Enter the Vendor name to be shown in the pack. If you provide a custom value for the Vendor field, it will change the name of the generated pack, as well as its installation location with your CMSIS Pack root folder. However, when creating a new project within a third-party IDE, users will select their project device from under the Cypress heading. See the appropriate instructions under [PSoC 6 Designs](#) for the appropriate 3rd party IDE.

- Pack:** Enter the Project name of the pack. The pack name will be combined with the device part number to present the device name in the µVision environment. µVision restricts the device name to be a maximum of 48 characters. So, the pack name must be short enough to be combined with the other characters to make total number of characters equal to 48 or less.
 - Version:** Enter the major version, minor version, and revision number for the pack, as appropriate for your development environment.
- **IAR EW-Arm** – If set to **Generate**, this option generates Application firmware, Component Instance firmware, and PDL Firmware for use in IAR. This menu also includes **Generate without copying PDL files**, which generates a separate .ipcf file to include PDL files in the generated output.
- **Makefile** – If set to **Generate**, this option generates Application firmware, Component Instance firmware, and PDL firmware for use in Make.

Types of PSoC Creator Project Firmware

A PSoC Creator project consists of different types of firmware:

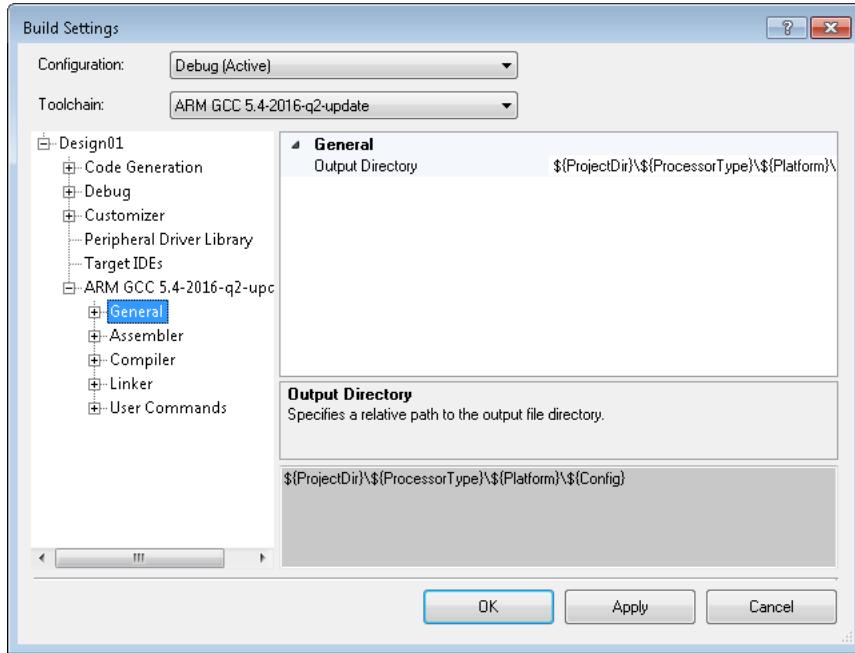
- **Application firmware** – Firmware files written and maintained by the user (e.g., main.c, cyapicallbacks.h).
- **Component Instance firmware** – Firmware generated by PSoC Creator during the build process. Every instance of a Component in the design has its own copy of its firmware. (e.g., UART_1.c, UART_1.h).
- **PDL firmware** – This firmware is a new driver-style firmware supported by PSoC 6 devices. A peripheral driver is a collection of related C functions shared by all instances of a peripheral.

See Also:

- [Build Settings](#)
- [Integrating into 3rd Party IDEs](#)
- [Generating PSoC 6 Files for 3rd Party IDEs](#)

Toolchain Build Settings

The Toolchain section of the Build Settings dialog provides additional toolchain-specific properties to define compiler, assembler, and linker options. These settings apply only to the selected **Configuration**, **Toolchain**, and **Processor Type**, and the settings can be set differently for each option.



General Category:

The top level of this category contains only one option under General: **Output Directory**. This option allows you to specify the relative path to the output file directory. The default is:

```
$(ProjectDir)\$(ProcessorType)\$(Platform)\$(Config)
```

This defines the project directory, processor type, platform, and configuration.

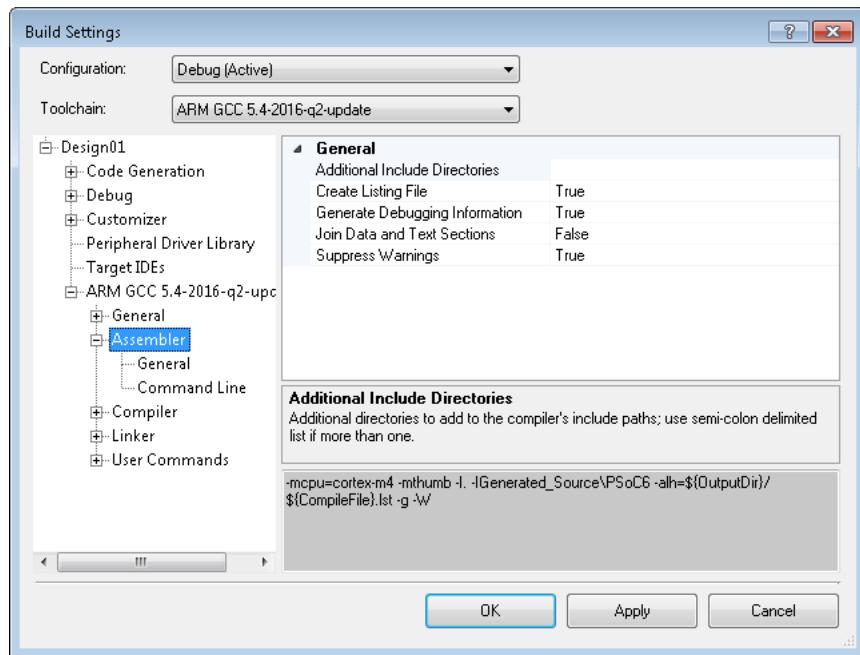
See Also:

- [Build Settings](#)

Assembler Build Settings

The Assembler section of the Build Settings dialog is used to control various options depending on the CPU and compiler.

Arm Options:



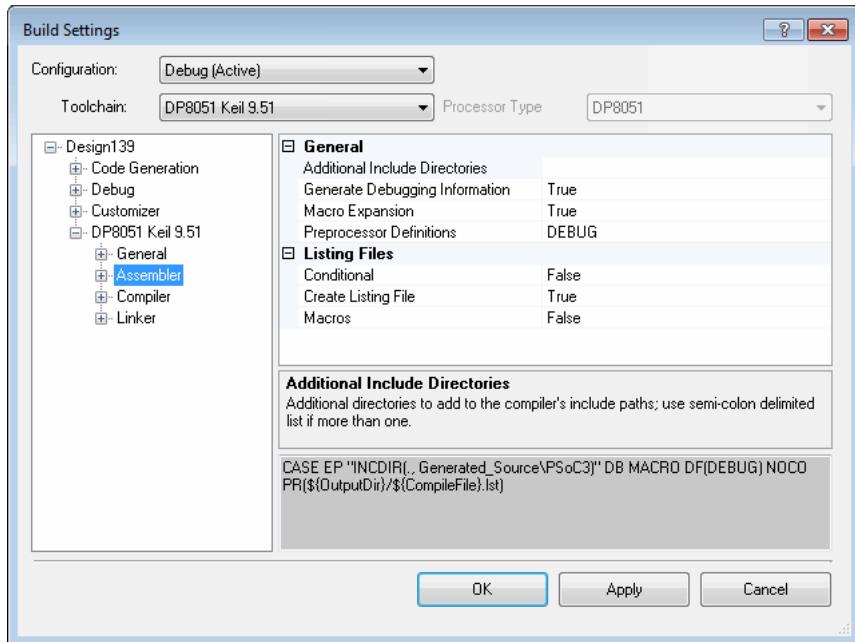
General

- **Additional Include Directories** – Specify additional directories to the compiler's include path. If you wish to specify more than one, separate them with semi-colons.
- **Create Listing File** – Create file containing high-level source and assembly: true or false.
- **Difference Tables** – Enable to issue warning when assembler alters the code emitted for directives: true or false.
- **Generate Debugging Information** – Produce debugging information to work with GDB: true or false.
- **Join Data and Text Sections** – Enable this option to generate shorter address displacements: true or false.
- **Suppress Warnings** – Enable this option to suppress all warnings: true or false.

Command Line

- **Custom Flags** – Allows you to enter any flags that are understood by the compiler. Refer to the appropriate compiler documentation. These flags are added to the flags generated by PSoC Creator based on the selections made in other sections.

Keil Options:



General

- **Additional Include Directories** – Specify additional directories to the compiler's include path. If you wish to specify more than one, separate them with semi-colons.
- **Generate Debugging Information** – Produce debugging information to work with GDB: true or false.
- **Macro Expansion** – Enable macro expansion: true or false.
- **Preprocessor Definitions** – Opens the Preprocessor Definitions dialog to add define directives to your source code.

Listing File

- **Conditional** – Include conditional assembler source code: true or false.
- **Create Listing File** – create file containing high-level source and assembly: true or false.
- **Macros** – Include all macro expansions in the listing file: true or false.

Command Line

- **Custom Flags** – Allows you to enter any flags that are understood by the compiler. Refer to the appropriate compiler documentation. These flags are added to the flags generated by PSoC Creator based on the selections made in other sections.

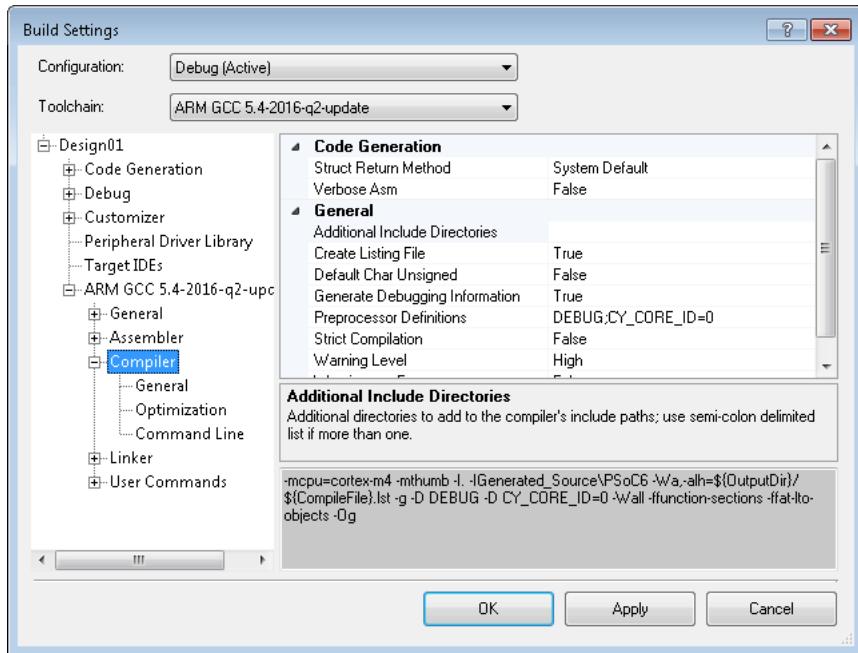
See Also:

- [Build Settings](#)

Compiler Build Settings

The Compiler section of the Build Settings dialog is used to control various options that will vary depending on the CPU and compiler.

Arm Options:



Code Generation

- **Struct Return Method** – Specify the method used for returning short structs/methods: system default, register, or memory.
- **Verbose Asm** – Enable extra commentary information in the generated assembly code to make it more readable: true or false.

General

- **Additional Include Directories** – Specify additional directories to the compiler's include path. If you wish to specify more than one, separate them with semi-colons.
- **Create Listing File** – Create file containing high-level source and assembly: true or false.
- **Default Char Unsigned** – Set the default char type to be unsigned: true or false.
- **Generate Debugging Information** – Produce debugging information to work with GDB: true or false.
- **Preprocessor Definitions** – Opens the Preprocessor Definitions dialog to add define directives that can impact how your C source code is compiled.
- **Strict Compilation** – Enable strict ISO C/C++ compilation: true or false.
- **Warning Level** – Determine the warning level to use: 0, 1, 2.
- **Warnings as errors** – Report all warnings as errors: true or false.

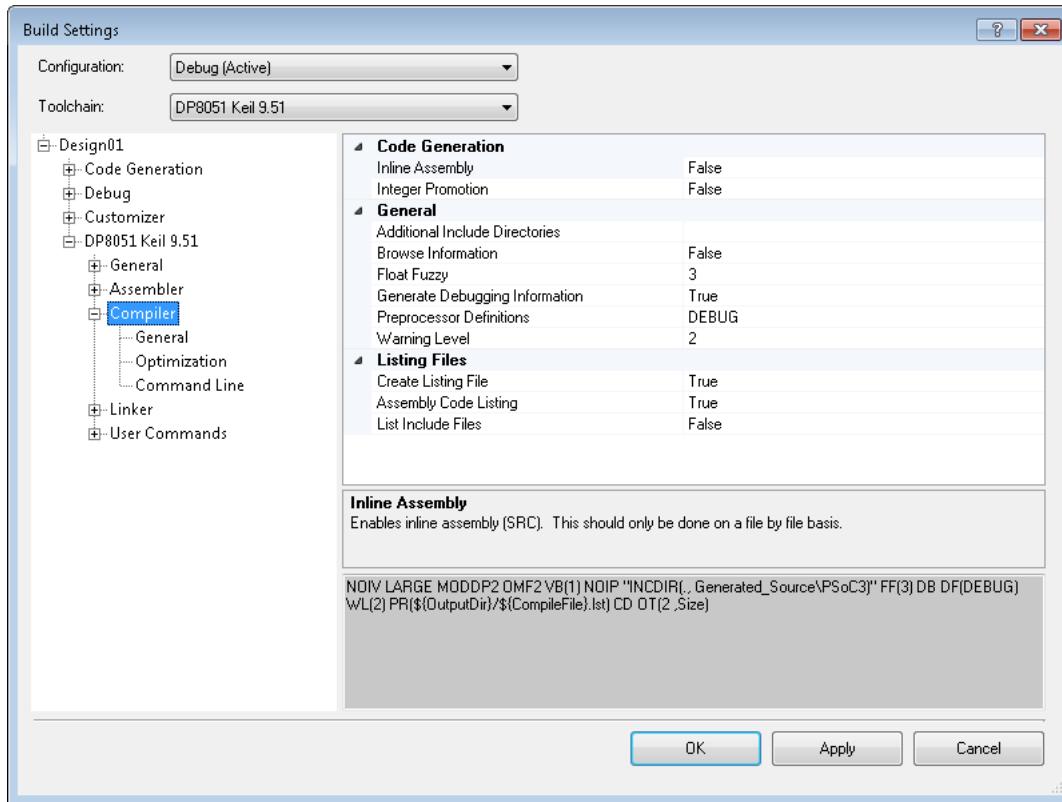
Optimization

- Create Function Sections – true or false.
- **Inline Functions** – Enable function inlining during compilation: true or false.
- **Optimization Level** – Options for code optimization. These values correspond with the toolchain command line code (in parentheses): None (-O0), Debug (-Og), Minimal (-O1), High (-O2), Speed (-O3), Size (-Os). Refer to the toolchain manual for details.

Command Line

- **Custom Flags** – Allows you to enter any flags that are understood by the compiler. Refer to the appropriate compiler documentation. These flags are added to the flags generated by PSoC Creator based on the selections made in other sections.

Keil Options:



Code Generation

- **Inline Assembly** – Enable inline assembly: true or false.
- **Integer Promotion** – Enable integer promotion: true or false.

General

- **Additional Include Directories** – Specify additional directories to the compiler's include path. If you wish to specify more than one, separate them with semi-colons.
- **Browse Information** – Include browser information in the generated object module: true or false.
- **Float Fuzzy** – Specify the number of bits rounded before comparing floating-point numbers: 0-7.
- **Generate Debugging Information** – Include debugging information in the object file: true or false.
- **Preprocessor Definitions** – Opens the Preprocessor Definitions dialog to add define directives that can impact how your C source code is compiled.
- **Warning Level** – Determine the warning level to use: 0, 1, 2.

Listing Files

- **Assembly Code Listing** – Append an assembly mnemonics list to the listing file: true or false
- **Create Listing File** – create file containing high-level source and assembly: true or false.
- **List Include Files** – Print a list of the #include files in the listing file: true or false.

Optimization

- **Linker Code Packing** – Include information in the object file for linker-level program optimizations: true or false.
- **Optimization Emphasis** – Indicate the emphasis of the optimization done by the compiler: none, size, or speed.
- **Optimization Level** – Options for code optimization: 0-11.

Command Line

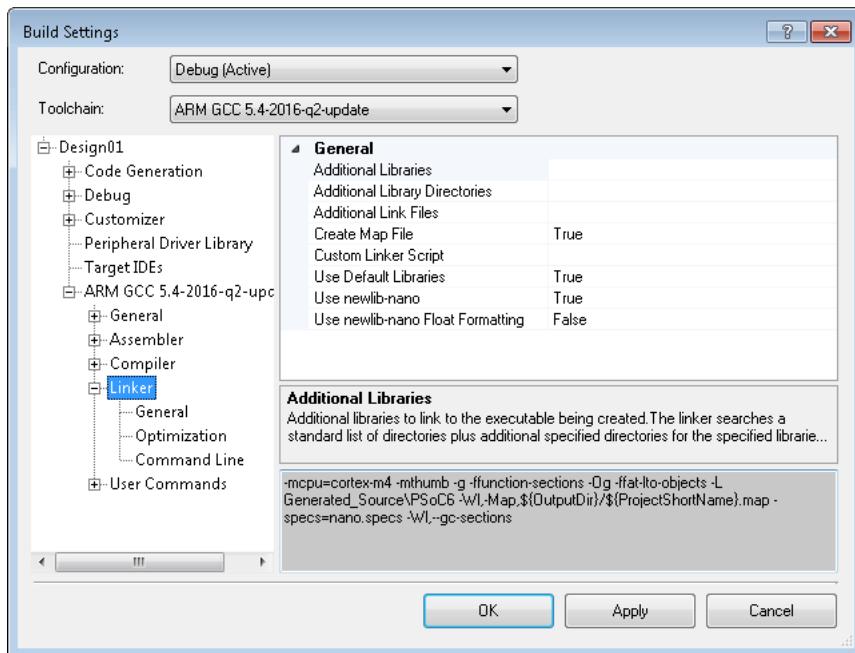
- **Custom Flags** – Allows you to enter any flags that are understood by the compiler. Refer to the appropriate compiler documentation. These flags are added to the flags generated by PSoC Creator based on the selections made in other sections.

See Also:

- [Build Settings](#)

Linker Build Settings

The Linker section of the Build Settings dialog is used to control various options depending on the CPU and compiler.



Arm Options:

General

- **Additional Libraries** – Specify additional libraries to link to the executable. Use semi-colons to separate more than one.
- **Additional Library Directories** – Specify additional libraries to add to the linker's library path. Use semi-colons to separate more than one.
- **Additional Link Files** – Specify additional files to link to the executable. Use semi-colons to separate more than one.
- **Create Map File** – Generate an updated listing file derived from the relocated addresses and data from the linker: true or false.
- **Custom Linker Script** – Specify the path to a custom linker script to use when building the project instead of the default script provided with the cy_boot Component.
- Remove Unused Functions – true or false
- **Use Debugging Information** – Enable to use the debugging information generated by gcc during compilation of the source code: true or false.
- **Use Default Libraries** – true or false

Command Line

- **Custom Flags** – Allows you to enter any flags that are understood by the compiler. Refer to the appropriate compiler documentation. These flags are added to the flags generated by PSoC Creator based on the selections made in other sections.

printf, sprintf, and floating point values

The newlib nano C library used with the Arm GCC toolchain does not include support for floating point like %f in format strings by default. This saves valuable flash memory. If you need to use floating point values you can tell newlib nano to include floating point support in format strings by adding a custom command line argument to the linker. Enter the following in the **Linker > Command Line** field:

```
-u _printf_float
```

Keil Options:

General

- **Additional Link Files** – Specify additional files to link to the executable. Use semi-colons to separate more than one.
- **Create Code Listing** – Create code listing file that contains program source/assembly: true or false.
- **Disable Unreferenced Segments Warnings** – Disable linker warnings about unreferenced code segments: true or false.
- **Recursions** – Control the number of recursions allowed in the linker before an abort: integer.
- Remove Unused Segments – true or false
- **Warning Level** – Determine the warning level to use: 0-2.

Listing File

- **Create Map File** – Generate an updated listing file derived from the relocated addresses and data from the linker: true or false.
- **Cross Reference Report** – Include a cross reference report in the listing file: true or false.
- **Generate Memory Map** – Generate the memory map and overlay map in the listing file: true or false.

Debugging

- **Generate Debug Lines** – Include line number information in the linker output and map files: true or false. If false, source-level debugging will not be possible.
- **Generate Local Symbols** – Include local symbol information in the linker output and map files: true or false. If false, source-level debugging will not be possible.
- **Generate Public Symbols** – Include public symbol information in the linker output and map files: true or false. If false, source-level debugging will not be possible.
- **Symbol Types** – Include symbol type information in the output file: true or false.

Command Line

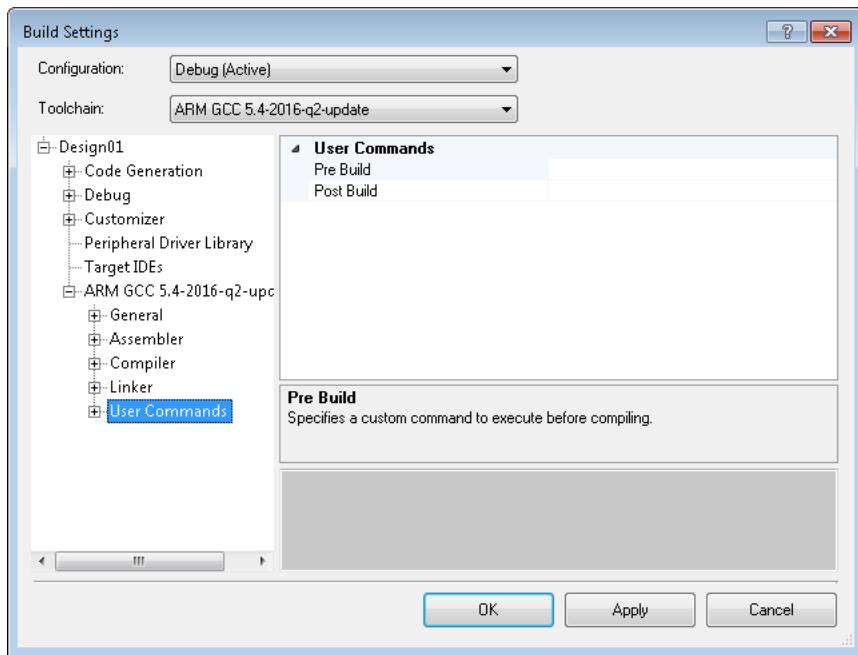
- **Custom Flags** – Allows you to enter any flags that are understood by the compiler. Refer to the appropriate compiler documentation. These flags are added to the flags generated by PSoC Creator based on the selections made in other sections.

See Also:

- [Build Settings](#)

User Commands Build Settings

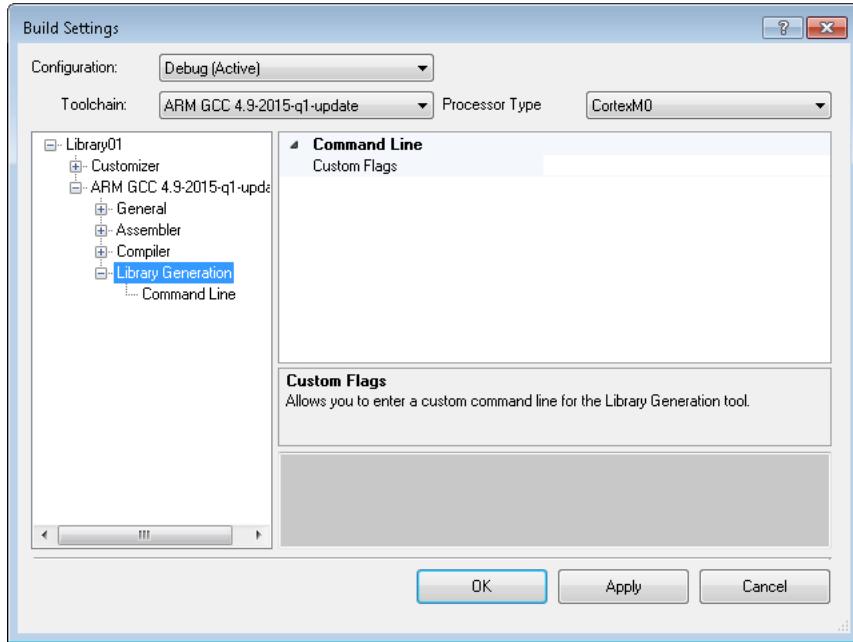
The User Commands section of the Build Settings dialog allows you to specify commands to run before compiling and after linking. The commands will be rooted from the project's cysdn directory.



- **Pre Build** - This command is run after code generation but before the compile step. If more than one command is needed, a batch script must be used.
- **Post Build** - This command is run after the link step but before the hex-file generation step. If more than one command is needed, a batch script must be used.

Library Generation Build Settings

The Library Generation section of the Build Settings dialog applies to Library projects only.



At this time there is only one property:

- **Custom Flags** – Allows you to enter any flags that are understood by the compiler. Refer to the appropriate compiler documentation. These flags are added to the flags generated by PSoC Creator based on the selections made in other sections.

See Also:

- [Build Settings](#)

Mapper, Placer, Router

When you use the **Build** function, PSoC Creator converts the schematic and Verilog specifications of your design into configuration information. That information can be used to program the digital and analog Components of the device. The first phase of the conversion process is called the mapper. Once mapping completes, the placer and router phases begin.

The mapper phase is where PSoC Creator maps the logic described in your design into Components that represent functional blocks of the PSoC device. The placer phase positions the functional blocks into available locations of the selected device. The placer will attempt to optimize the locations for the different Components based on the connectivity in the design, as well as following any restrictions on locations you may have specified. After locations for all the Components have been determined, the router phase computes the optimal signal path between the Components through the switching fabric of the device. If there are not enough resources to support the Components necessary for the design, or if PSoC Creator determines that routing is not possible, it will generate an error and the build will fail.

Note Due to the relationships between the domains of the part, PSoC Creator will attempt to place and route all the analog Components first. Once that is complete, it will attempt to place and route the digital portion of the design.

Once the design has been mapped, placed and routed, the settings to create that configuration are generated and written out to the [Generated Files](#) folder to be used when building the entire application for the project. You can influence the results of the place and route phase by specifying where Components should be located in the design. For more information, see [Pin Editor](#) and [Directives Editor](#) sections under [Design-Wide Resources](#), as well as the [Directives](#) topic.

Migrating from Older PSoC Creator Versions

When migrating from older versions of PSoC Creator, it is possible for a design that was previously building successfully to fail with errors during Mapping, Placing, or, Routing. There are two reasons this can occur:

- a new rule has been added to the PSoC Creator backend that was previously missing
- changes to the backend have resulted in the project "moving" into a corner of the solution that cannot be solved

If a new rule has been added to PSoC Creator, the design was invalid, but PSoC Creator was not flagging the design. If the error message is not about a rule violation, but about PSoC Creator being unable to map, place, or, route the design, this can be worked around.

To work around the issue, open the design in the previously successful version of PSoC Creator and follow the instructions for adding a [Control File](#). Then use the control file to force the design to have the same placement as the design previously had. Then save and close the design. When you open the design in the new version of PSoC Creator, the design will keep the control file and work as in the previous version.

See Also:

- [Design-Wide Resources](#)
- [Pin Editor](#)
- [Directives Editor](#)
- [Directives](#)
- [Generated Files](#)
- [Control File](#)

Control File

A control file is an optional file that provides a common location for setting global directives for a given design. This provides detailed control over many aspects of synthesis while maintaining a device and vendor independent hardware description language (HDL) source file. The control file allows the user to attach directives via the attribute mechanism, and the file supports the Warp specific HDL syntax for these (extended) attributes to allow the cutting and pasting of these directives between the HDL source and the control file. The file can also be used for back-annotating pinout and internal placement information from fitting and place and route results automatically.

During the process of synthesis, optimization, and factoring, Warp derives many new signal and node names to realize the design. For example, Warp separates buses into individual signals. Even though objects such as buses make HDL design entry much simpler, no VHDL-based way exists to assign attributes to portions of a bus. In other cases, Warp produces brand new signal names which may not have any direct correlation to any single VHDL/Verilog object within a design. This situation occurs during factorization where factors are produced by examining the design globally.

Only one control file is allowed per design, and the file must have the same base name as the top-level design file name. For example, for a top-level design whose name is *mydesign.vhd* or *mydesign.v*, the control file must be called *mydesign.ctl*.

The creation and editing of the control file is an iterative process, typically done to refine, improve, or constrain the results of synthesis.

The control file is applied during parsing/analysis, during synthesis, and during fitting. Names created during parsing/analysis are not qualified. Therefore, if the control file pattern matches the name, the attribute is applied. This occurs at any level of the design hierarchy.

For example, consider a top level signal called *c*. An attribute applied to *c* will be applied to the top level signal, but it will also be applied to any internal signal named *c*, including those in the Warp library. So if the control file says:

```
attribute placement_force of c : signal is "U(1,2)A";
```

it will attempt to force all signals named *c* into that programmable logic block.

Likely, there will be too many signals, and the placer will complain. The work around is to rename the top level signal something unique, like *top_level_c*, or to use an attribute in the source file, not the control file.

To Create a Control File:

1. Click the **Components** tab of the Workspace Explorer.
2. Right-click on the **TopDesign** Component of the active project and select **Add Component Item...**

The Add Component Item dialog opens.

3. Scroll down to the **Misc** group, select **Control File**, and click **OK**.

A *TopDesign.ctl* file is created and added to the Workspace Explorer.

4. Double-click it to open the file.

See Also:

- [Directives](#)
- Attribute, CSAttribute, and FixedAttribute
- [Control File Format](#)
- [Control File Pattern Matching](#)

Attribute, CSAttribute, and FixedAttribute

Attribute:

Use Attribute for most applications of attributes.

The attribute keyword treats all patterns that do not begin with a backslash in a case-insensitive manner. Any pattern that begins with a backslash is treated in a case-sensitive manner (similar to CSAttribute). The attribute keyword is also allowed within the text of VHDL designs and will adhere to the syntax and semantics as defined in the VHDL language. However, the meaning of attribute is still the same with the exception of applying wildcard patterns.

CSAttribute:

For the rare situation when there are multiple objects which have the same identifier differing only in case (eg, Fred_Astaire and fred_astaire), CSAttribute (case sensitive attribute) can be used to select the object with the same name as the pattern provided.

The CSAttribute keyword treats all patterns in a case-sensitive manner.

FixedAttribute:

FixedAttribute is intended for use by tools. The pattern on the FixedAttribute line is actually a case sensitive name in which any meta characters are treated as actual characters in the identifier.

The FixedAttribute keyword accepts an identifier, not a pattern, and does a case-sensitive match.

Validity Checking:

For both the CSAttribute and Attribute directives in the control file, the validity of the attribute name and the legality of the value must be checked for both Verilog and VHDL based designs. For string valued attributes, the actual check is done by the destined client of the attribute (for example the placement_force value will only be checked by the appropriate fitter). The following example shows how Attribute, CSAttribute and FixedAttribute match with Verilog and VHDL source code.

Pattern	Source Identifier	VHDL Source Match	Verilog Source Match
attribute ff_type of mysig : signal is ff_d	mysig Mysig MYSIG mySig	Yes Yes Yes Yes	Yes Yes Yes Yes
csattribute ff_type of mysig : signal is ff_d	mysig Mysig MYSIG mySig	Yes Yes Yes Yes	Yes No No No
fixedattribute ff_type of mysig : signal is ff_d	mysig Mysig MYSIG mySig	Yes Yes Yes Yes	Yes No No No

See Also:

- [Control File](#)

Control File Format

The format of the control file is as follows:

- A comment begins with "--" and terminates at the end of the line.
- The lines in the control file contain attribute, csattribute, or fixedattribute statements.
- The line must start with one of the keywords: attribute, csattribute, or fixedattribute.

The syntax for the attribute, csattribute and fixedattribute statements in the control file are as follows:

```
attribute attribute_name [of] pattern [:] [object-class] [is] value ;
csattribute attribute_name [of] pattern [:] [object-class] [is] value ;
fixedattribute attribute_name [of] identifier [:] [object-class] [is] value ;
```

All the words in the above lines except for the pattern and identifier are treated in a case-insensitive manner. The pattern is interpreted as described in [Control File Pattern Matching](#). The default object-class is a SIGNAL. However object-class can be:

- label
- entity
- module
- architecture
- signal

For VHDL designs, attribute statements can be present in both the control file and in the source. A control file attribute will take precedence over the source file attribute. If there is a specific attribute in the source and a matching attribute line in the control file, the attribute in the control file will take precedence.

A specifically applied attribute takes precedence over a hierarchically applied attribute.

Attribute_name:

Attribute_name is the directive name.

Patterns/Identifiers:

The name of the object upon which the directive is being placed can be specified as an identifier (simple, extended/escaped) or as a pattern. Individual bits of an array are represented by enclosing the integer in parentheses.

In the control file, Warp accommodates both VHDL extended identifiers and limited Verilog escaped identifiers. An extended or escaped identifier always starts with a backslash. It is terminated with the first unescaped backslash if it exists, or the first encountered white space if there is no trailing backslash terminator. A backslash within a VHDL extended identifier is escaped by preceding it with another backslash.

In the case of Verilog escaped identifiers, no embedded backslashes are allowed in the control files. For example, even though 'foo\bar' is a valid escaped identifier in Verilog, it is not valid in the control file. The limited version of the escaped identifiers are added as a convenience to the user and the recommendation is to always use VHDL style extended identifiers.

Note Names with [] need to be replaced with wildcards instead of the []. For example:

```
attribute placement_force of \Sync:genblk1[0]:INST\ : label is "U(0,0)2"
```

becomes:

```
attribute placement_force of \Sync:genblk1?0?:INST\ : label is "U(0,0)2"
```

Optional Keywords:

The keywords "of" and "is" are optional and are simply ignored.

Object-Class:

Object-class refers to the type of HDL object. If the object-class is not specified, a signal is assumed. For VHDL, valid classes include entity, architecture, Component, and label. The label class can be used to specify a directive intended for a Component instantiation. For Verilog, valid classes include module, label, and signal.

Directive Terminator:

A directive is terminated either with a new line, a semi-colon, or a comment.

Value:

Value is the value of the directive.

Control File Pattern Matching

The object name portion of an attribute specification in a control file can be more than just a name -- it is actually a pattern; the attribute will be applied to any object (of the right type) whose name is matched by the pattern. The pattern is composed of normal characters, which must match exactly in a case sensitive or insensitive manner depending on the attribute type. These are the additional wildcard constructs, shown in the following table:

Characters/Constructs	Definition
*	represents a match with zero or more characters
?	represents a match with exactly one character
[c-c]	represents a single character in the given ASCII character range that has to be matched.
[0-9]	represents a single character in the given ASCII character range that has to be matched.
[ccc]	represents a list of characters to match.

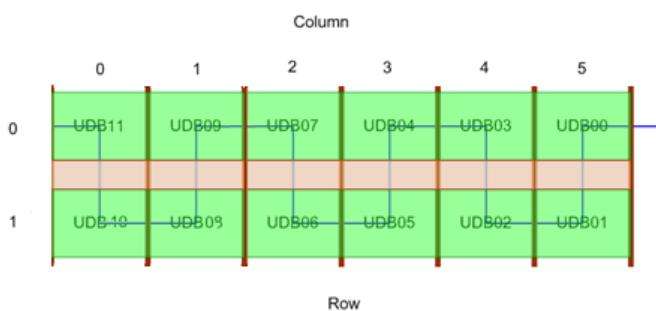
While constructing the pattern, the characters are treated as case sensitive. The matching still depends on the attribute directive type (Attribute, CSAttribute). The pattern matching characters and wildcard constructs are treated as character literals when enclosed in []. The pattern matching is always a complete match.

PSoC UDBs in PSoC Creator

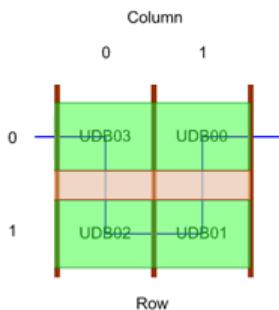
PSoC Creator refers to the universal digital blocks (UDBs) by their location in the UDB grid (Row, Column format [UDB=(0,2)]) in the report file (see [Generated Files](#)) and in the [Directives Editor](#). This is different than how the device Technical Reference Manuals (TRMs) reference UDBs (Bank Number, Udb in Bank, for example, Bank 0, Udb 11).

In the PSoC Creator report files, the UDB elements are referred to with a naming convention different than given in the TRM. The notation used is U (row, column) where the row and column are based on a grid with 0,0 in the upper left corner of the UDB array. The mapping from the grid-based to TRM-based naming convention is shown in the following examples:

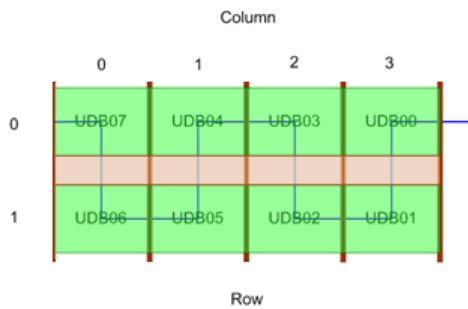
PSoC 6



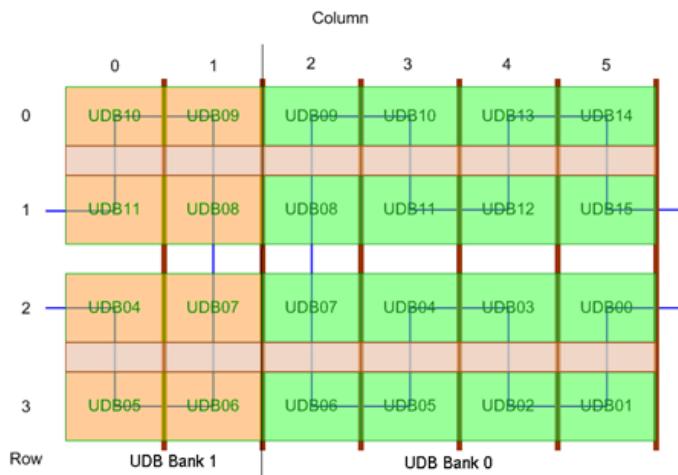
PSoC 4



PSoC 4AL



PSoC 3/PSoC 5LP



Directives

Directives may be used to influence the implementation of a design. They are used in an iterative fashion to refine, improve, or constrain the results of synthesis. Directives may be applied to Components that have been either instantiated in a schematic or inferred by the synthesizer from Verilog HDL code.

Directive Format Summary:

The following table summarizes the directives used in PSoC Creator.

Directive	Target	Format	Location values
Placement Directives			
placement_force	Arbitrary logic	attribute placement_force of signal_name : signal is "string";	U(1,2)A or U(1,2,A)3
placement_force	Fixed function	attribute placement_force of Component_name : label is "string";	F(Timer,3) or F(DFB,0)
placement_force	UDB Component	attribute placement_force of Component_name : label is "string";	U(3,2)
port_location	IO Port	attribute port_location of bit_name : label is "string";	PORT(2,3)
placement_group	Arbitrary logic	attribute placement_group of signal-name : signal is "string" ;	group_1
synchronization_needed	IO Port	attribute synchronization_needed of signal_name : signal is "string";	AUTO or SYNC or NOSYNC
Synthesis Directives			
no_factor	Arbitrary logic	attribute no_factor of signal_name : signal is value;	
opt_level	Arbitrary logic	attribute opt_level of signal_name : signal is integer;	2 or 1 or 0

Directive	Target	Format	Location values
synthesis_off	Arbitrary logic	attribute synthesis_off of signal_name : signal is value;	false or true

Directive Implementation on Different Devices

In cases where there are multiple directives of the same type in a design, and if the directives force the same block to different locations on the device, there is a difference in how the directives are applied between FM0+ devices and PSoC devices. For example:

```
attribute port_location of pin_1(0) : label is "PORT(2,3)";
attribute port_location of pin_1(0) : label is "PORT(2,5);
```

For FM0+ devices, the first directive is selected and applied; the pin instance will be forced to pin P2[3]. This behavior is true for placement_force and port_location directives on FM0+ devices.

For PSoC devices, the last directive is selected and applied; the pin instance will be forced to port P2[5]. This behavior is true for all directives on PSoC devices.

placement_force:

The placement_force directive aids in locking down signals and associated Components to particular locations in the fitter. Typically the fitter positions Components in an optimal location on the device; however, you might want to constrain the fitter to position Components in specific locations.

```
attribute placement_force of signal_name : signal is "string";
attribute placement_force of Component_name : label is "string";
```

Note The Component whose placement is to be forced may be embedded within the hierarchical implementation of a Component on the top level schematic (TopDesign). If so, it is necessary to use the full hierarchical Component name. The easiest way to find this name is to build the project and search in the report file, under "Final Placement Summary," for the name of the instance on the top-level schematic. This will locate all child instances contained within the top-level instance.

The string is a location descriptor of either a programmable logic block, macrocell, UDB element, or fixed function block. Each has its own format and meaning specific to the object being addressed.

- For an element of a UDB (Datapath, status register, etc.), it can be constrained to a specific UDB. The format to specify this is `U(x,y)` where `x,y` is the UDB row and column. For the attribute to be recognized, the attribute must be assigned to the Component using the version of the attribute with the label keyword for it to have an effect. An example of this would be:

```
attribute placement_force of \test_control:ctrl_reg : label is "U(3,5);
```

Note When specifying the location of a datapath chain, the attribute must be applied to the first datapath in the chain for it to have an effect.

- In the case of a set of random logic, it can be constrained to a specific PLD within a specific UDB. The format to specify this is `U(x,y)l` where `x,y` is the UDB row and column and `l` is the PLD descriptor ("A" or "B" to denote one of the two PLDs within a UDB). This will still allow the fitter to choose a macrocell column that best fits the design layout. For the attribute to be recognized, the attribute must be assigned to the output signal from the macrocell for it to have an effect. An example of this would be:

```
attribute placement_force of out_1 : signal is "U(2,4)A";
```

- It is also possible to constrain a set of random logic to a particular macrocell column within a specific PLD on a specific UDB. The format to specify this is: `U(x,y,l)` where `x,y` is the UDB row and column, `l` is the PLD descriptor ("A" or "B" to denote one of the two PLDs within a UDB) and `l` is the macrocell index within the PLD. For the attribute to be recognized, the attribute must be assigned to the output signal from the set of random logic for it to have an effect. An example of this would be:

```
attribute placement_force of out_2 : signal is "U(1,3,A)2";
```

- The attribute should be applied directly to any fixed function Components by referencing them by name (instead of referencing a signal) by using the label keyword. The format for the "label" is `F(block_type,index)`, where the `block_type` is the keyword descriptor for the block, and `index` is the instance of that block to use. The valid set of keywords are: CAN, Comparator, DFB, DSM, Decimator, EMIF, I2C, SC, Timer, USB, VIDAC, Abuf, Lpf:

```
attribute placement_force of \Timer_1:TimerHW\ : label is "F(Timer,1)";
```

- This attribute can be applied to a DMA Component to force the use of a particular channel. The format for the location string is:

```
DMA(0, channel_id)
```

On PSoC 6 devices, channels are spread across multiple controllers; however, this is still specified in the directive as a single channel id. To force placement to controller C, channel N, use the following:

```
channel_id = C * channels_per_controller + N
```

To find `channels_per_controller`, look at the value of `CPUSS_DW_CH_NR` in the PDL `series_config.h` file (e.g., `psoc63_config.h`) for your device.

The [Directives Editor](#) does not support forcing DMA placement; you must use a [control file](#) instead.

port_location:

The `port_location` directive maps the external signals of the design to pins on the target device based on the location of that pin within the device, not on the package. The syntax for the directive is:

```
attribute port_location of bit_name : label is "string";
```

The format for the string to describe the location is `PORT(port_index, pin_index)` where `port_index` is the number of the port on the device and `pin_index` is the pin within the port.

An example for this directive:

```
attribute port_location of pin_1(0) : label is "PORT(2,3)";
```

Note When using this directive to fix the location of a contiguous pin group, the attribute should be applied to the first pin.

placement_group:

The `placement_group` directive can be used to group a set of signals (that relate to arbitrary logic, not fixed function Components) and to ask the place and route tool to place these signals together. The place and route tool will attempt to place all signals that have the same group identifier together in a minimal number of PLDs. The format of this directive is:

```
attribute placement_group of signal-name : signal is "string" ;
```

The string used can be any user defined string that is used to uniquely identify a group (see examples below). The group identifier is case insensitive. The following example will attempt to put the logic driving signal2 and signal3 together:

```
attribute placement_group of signal2: signal is "group1" ;
attribute placement_group of signal3: signal is "group1" ;
```

synchronization_needed:

The synchronization_needed directive can be used to modify the default synchronization behavior for the IOs of the part. By default, IOs used as inputs are synchronized and IOs used for outputs are unsynchronized.

```
attribute synchronization_needed of signal-name : signal is "string" ;
attribute synchronization_needed of logical-port-name(pin-index) : label is "string" ;
```

The synchronization_needed attribute can take the value of a string. This string value can be one of AUTO, SYNC, or NOSYNC. Examples of the use of the attribute would be:

```
attribute synchronization_needed of Terminal_1 : signal is "SYNC" ;
attribute synchronization_needed of dport_1(2) : label is "NOSYNC" ;
```

no_factor:

The no_factor directive prevents logic factoring within the Warp synthesis engine to prevent splitting said node.

```
attribute no_factor of signal_name : signal is value;
```

During the optimization phase, the Warp synthesis engine aliases signals which have identical drivers (equations). Using this directive causes equations to bypass these two actions. This feature can be useful if the design constraints cause certain identical logic to be duplicated or if the logic factoring algorithm is being overaggressive.

Examples:

This example prevents the signal my_signal from being aliased or from being factored.

```
attribute no_factor of my_signal : signal is true;
```

In Verilog designs, attributes can be placed on all signals in a module as follows:

```
attribute no_factor of my_module : module is true;
```

This example prevents all signals in my_module from being aliased or factored.

opt_level:

The opt_level directive instructs Warp on the amount of effort that should be spent optimizing certain signals.

```
attribute opt_level of signal_name : signal is integer;
```

The integer represents the amount of effort. Currently, there are three levels of effort (0, 1 and 2). An opt_level of 0 instructs Warp to turn off all optimization on said signal. This directive is also passed along to the PLD/CPLD fitters which do the same thing. An opt_level of 1 causes Warp to perform a simple and quick optimization of equations. An opt_level of 2 causes Warp to perform the highest level of optimization available. An opt_level of 2 is recommended for all designs.

Example:

This directive disables all optimization on the signal my_signal.

```
attribute opt_level of my_signal : signal is 0;
```

synthesis_off:

The synthesis_off directive controls the flattening and factoring of expressions feeding signals for which the directive is set to true. This directive causes a signal to be made into a factoring point for logic equations, which keeps the signal from being substituted out during optimization.

```
attribute synthesis_off of signal_name : signal is value;
```

The synthesis_off directive can only be applied to signals. The default value of the synthesis_off directive for a given signal is false. This directive gives the user control over which equations or sub-expressions need to be factored into a node (i.e., assigned to a physical routing path).

- When set to true for a given signal, synthesis_off causes that signal to be made into a node (i.e., a factoring point for logic equations) for the target technology. This keeps the signal from being substituted out during the optimization process. This can be helpful in cases where performing the substitution causes the optimization phase to take an unacceptably long time (due to exponentially increasing CPU and memory requirements) or uses too many resources.
- Making equations into nodes forces signals to take an extra pass through the array, thereby decreasing performance, but may allow designs to fit better.
- The synthesis_off directive should only be used on combinational equations. Registered equations are natural factoring points; the use of synthesis_off on such equations may result in redundant factoring.

Example:

This example sets the synthesis_off directive to true for a signal named sig1.

```
attribute synthesis_off of sig1:signal is true;
```

See Also:

- [Directives Editor](#)
- [Control File](#)

Generated Files (PSoC 3, PSoC 4, PSoC 5LP)

Upon a successful build, PSoC Creator generates various files that become a part of your design. These files are listed in the [Workspace Explorer](#) under the **Source** tab. These files are specific to the device family (PSoC 3, PSoC 4, or PSoC 5LP) and the selected compiler. The following lists and describes the files generated from a build.

File(s)	Description
cy_boot (Refer also to the System Reference Guide.)	
CyBootAsmKeil.a51 (PSoC 3) CyBootAsmlar.s (PSoC 4/5LP IAR) CyBootAsmGnu.s (PSoC 4/5LP GCC) CyBootAsmRv.s (PSoC 4/5LP MDK)	Provides device- and toolchain-specific assembly implementations for startup and time critical routines.
CyDmac.c /h	The software API for using the DMA Controller.
CyFlash.c /h	The software API for writing to flash.
CyLib.c /h	The software APIs for power management, string/character routines, memory manipulation, as well as enabling/disabling selected portions of the PSoC device.
cypins.h	Contains the function prototypes and constants used for port/pin access and control.
cyPm.c./h	Provides the function definitions for the power management API.
CySpc.c ./h	The software API for writing to the System Performance Controller.
cytypes.h	Provides macros and defines to allow code to be written tool chain and processor agnostic.
cyutils.c	Implements low-level utility functions used to provide tool chain/processor agnostic functions. Exposed in cytypes.h.
cymem.a51 (PSoC 3)	Specialized memory routines for Keil boot-up.
KeilStart.a51 (PSoC 3)	Bootup code for PSoC 3 chips using Keil tools.
Cm3Start.c/Cm0Start.c/Cm0pStart.c	Startup code for the Arm CM3/CM0/CM0+.
PSoC3_8051.h ./inc (PSoC 3)	8051 register definitions for the PSoC 3 architecture.
cm3gcc.ld/cm0gcc.ld	Linker script for the GCC toolchain
Cm3RealView.scat/Cm0RealView.scat	Scatter file for the RealView & MDK toolchains
core_cm0.h or core_cm3.h core_cmFunc.h and core_cmlInstr.h	CMSIS standard libraries for Cortex-M series of processors: core_cm0.h for PSoC 4/core_cm3.h for PSoC 5LP. Both files included for PSoC 4/PRoC BLE and PSoC 5LP.
core_cm0_psoc4.h or core_cm3_psoc5.h	PSoC 4/PRoC BLE or PSoC 5LP specific interrupt information for CMSIS libraries.
General	
cydevice_trm.h	Defines all of the addresses in the configuration space of the device. These addresses do not contain any context information related to instances drawn in your design(s). You should not need to use any of these addresses directly.
cydevice.h (PSoC 3/PSoC 5LP)	Deprecated version of cydevice_trm.h.
cydevicekeil_trm.inc (PSoC 3) cydevicegnu_trm.inc (PSoC 5LP / PSoC 4 GCC) cydevicerv_trm.inc (PSoC 5LP / PSoC 4 MDK) cydeviciar_trm.inc (PSoC 5LP / PSoC 4 IAR)	Defines all of the addresses in the configuration space of the device for the specific toolchain. These addresses do not contain any context information related to instances drawn in your design(s). You should not need to use any of these addresses directly.

File(s)	Description
cydevicekeil.inc (PSoC 3) cydevicegnu.inc (PSoC 5LP GCC) cydevicerv.inc (PSoC 5LP Real View)	Deprecated version of cydevicekeil_trm.inc, cydevicegnu_trm.inc, or cydevicerv_trm.inc.
cyfitter.h	Defines all of the instance specific addresses calculated by the Code Generation step. This file is mainly intended for use by instance API implementations, although advanced users may find interesting information in this file.
cyfitter_cfg.c	Implements the methods and logic necessary to configure the device before main. You should not need to use anything implemented in this file.
cyfitter_cfg.h	Contains definitions used by the boot firmware to configure the device before main. You should not need to use anything defined in this file.
cyfitterkeil.inc (PSoC 3) cyfittergnu.inc (PSoC 5LP / PSoC 4 GCC) cyfitterrv.inc (PSoC 5LP / PSoC 4 MDK) cyfitteriar.inc (PSoC 5LP / PSoC 4 IAR)	Assembly equivalent of cyfitter.h.
project.h	This file includes all of the other header files found in this directory and its sub-directories. It exists for convenience sake, allowing you to include all of the generated headers with just one #include statement.

Boot Component:

All design projects include a "boot" Component to provide version control for all the boot firmware files, such as CyDma*, CyFlash*, etc. The Component is hidden in the [Component Catalog](#) by default, and it cannot be placed onto a design. The files from the boot Component get deposited into a folder named "Generated_Source/<Architecture Name>/cy_boot" in the Workspace Explorer after a successful build.

Designs created with earlier releases of PSoC Creator will include earlier versions of the boot Component, which contains all the API files as they were. If updated API files are needed, use the [Component Update Tool](#) to upgrade the boot Component to the latest version.

Refer also to the System Reference Guide.

Component APIs:

The generated source will also include APIs for instantiated Components (e.g., counter_1.c, counter_1INT.c, counter_1.h), which will be listed in device-specific folders. If you do not want APIs generated for a specific instance, use the built-in parameter CY_SUPPRESS_API_GEN. See [Configure Component Parameters](#) for more information.

Results Files:

The Workspace Explorer also contains the **Results** tab, which contains the following files:

- <project>.cycdx – This contains XML information specific to Component debug windows. This is used by the debugger to determine what to display for the design. There is no reason for you to open or modify this file. For more information about this file, refer to the [Component Author Guide](#).

- <project>.rpt – This is the project report file. It contains information for how the device was programmed, including a section on how the target device's resources were utilized. Advanced users can review the information in this file to determine if there might be better ways to configure the design.
- <project>_timing.html – This is the Static Timing Analysis report. See [Static Timing Analysis](#) for more information.

The **Results** tab may also contain various other files, including

- <project>.cyfit – This file is an internal database for PSoC Creator to hold data on the results of Code Generation. A user will not interact with this file directly. It is regenerated each time the project is built.
- <project>.elf – This file contains debugging information for the GCC tool chains. A user will not interact with this file directly.
- <project>.ihx (PSoC 3 only) – Intel HEX file produced by building the project, containing only the compiled design. A user will not interact with this file directly.
- <project>.hex – Intel HEX file produced by combining <project>.ihx file with selected protection, configuration, and initialization settings. A user will not interact with this file directly.
- <project>.map – This file is produced by the linker. It contains details on how the device's memory was used, where functions and variables were placed and other details depending on the tool chain. A user will not interact with this file directly.
- <file>.lst – Code listing file showing initial c code and generated assembly. A user will not interact with this file directly.
- <project>.omf (PSoC 3 only) – This file contains debugging information output by the Keil toolchain. A user will not interact with this file directly.

See Also:

- [Workspace Explorer](#)
- [Component Catalog](#)
- [Component Update](#)
- [Static Timing Analysis](#)

Generated Files (PSoC 6)

Upon a successful build, PSoC Creator generates various files that become a part of your design. These files are listed in the [Workspace Explorer](#) under the **Source** tab. These files are specific to the device family and the selected compiler. The following lists and describes the files generated from a build.

File(s)	Description
Header Files	
cy_ipc_config.h	Defines a device-specific configuration for the IPC channels and pipes.
cyapicallbacks.h	Used to specify macro callbacks here. For more information, refer to the Writing Code topic in the PSoC Creator Help.
Source Files	
cy_ipc_config.c	Code required to configure the device-specific IPC channels for locks and pipes.
main.c or main_cm0.c, and main_cm4.c	Empty shell file to write your application code. This file is not overwritten.
pdl	
All files in this folder are copied from the installed peripheral driver library (PDL). See Options Dialog .	
Pins and Interrupts	
All files in this folder contain pin and interrupt defines and structures for each of the Components in the design.	
General	
cycodeshareexport.ld	GCC Linker script for exporting symbols from one application to be used in a second application.
cycodeshareimport.ld	GCC Linker script for importing symbols from a different application.
cycodeshareimport.scat	MDK Scatter file for importing symbols from a different application.
cydevice_trm.h	Defines all the addresses in the configuration space of the device. These addresses do not contain any context information related to instances drawn in your design(s). You should not need to use any of these addresses directly.
cydevicegnu_trm.inc	Defines all the addresses in the configuration space of the device for the GNU assembler (gas). These addresses do not contain any context information related to instances drawn in your design(s). You should not need to use any of these addresses directly.
cydeviceiar_trm.inc	Defines all the addresses in the configuration space of the device for the IAR assembler. These addresses do not contain any context information related to instances drawn in your design(s). You should not need to use any of these addresses directly.
cydevicerv_trm.inc	Defines all the addresses in the configuration space of the device for the Real View assembler. These addresses do not contain any context information related to instances drawn in your design(s). You should not need to use any of these addresses directly.
cydisabledsheets.h	File of defines for disabled schematic pages.
cyfitter.h	Defines all the instance specific addresses calculated by the Code Generation step. This file is mainly intended for use by instance API implementations, although advanced users may find interesting information in this file.
cyfitter_cfg.c	Implements the methods and logic necessary to configure the device before main. You should not need to use anything implemented in this file.
cyfitter_cfg.h	Contains definitions used by the boot firmware to configure the device before main. You should not need to use anything defined in this file.
cymetadata.c	This file defines all extra memory spaces that need to be included. This file is automatically generated by PSoC Creator.
project.h	This file includes all the other header files found in this directory and its sub-directories. It exists for convenience sake, allowing you to include all the generated headers with just one #include statement.

Component APIs:

The generated source will also include APIs for instantiated Components (e.g., counter_1.c, counter_1INT.c, counter_1.h), which will be listed in device-specific folders. If you do not want APIs generated for a specific instance, use the built-in parameter CY_SUPPRESS_API_GEN. See [Configure Component Parameters](#) for more information.

Results Files:

The Workspace Explorer also contains the **Results** tab, which contains the following files:

- <project>.cycdx – This contains XML information specific to Component debug windows. This is used by the debugger to determine what to display for the design. There is no reason for you to open or modify this file. For more information about this file, refer to the [Component Author Guide](#).
- <project>.rpt – This is the project report file. It contains information for how the device was programmed, including a section on how the target device's resources were utilized. Advanced users can review the information in this file to determine if there might be better ways to configure the design.

The **Results** tab may also contain various other files, including

- <project>.cyfit – This file is an internal database for PSoC Creator to hold data on the results of Code Generation. A user will not interact with this file directly. It is regenerated each time the project is built.
- <project>.elf – This file contains debugging information for the tool chains. A user will not interact with this file directly.
- <project>.hex – Intel HEX file produced by post-processing the <project>.elf file. This file contains all the information necessary to program your device, without any of the debug information found in the elf file. A user will not interact with this file directly.
- <project>.map – This file is produced by the linker. It contains details on how the device's memory was used, where functions and variables were placed and other details depending on the tool chain.
- <file>.lst – Code listing file showing initial c code and generated assembly.

Generate Source Files MISRA Compliance

There is no System Reference Guide for PSoC 6 devices, so there is no general section for MISRA (Motor Industry Software Reliability Association) violation information. The MISRA specification covers a set of 122 mandatory rules and 20 advisory rules that apply to firmware design and has been put together by the Automotive Industry to enhance the quality and robustness of the firmware code embedded in automotive devices.

For PSoC 6 devices, most MISRA documentation is located in the PDL API Documentation. However, there are several generated files from PSoC Creator that are not contained in that guide. The following sections contain MISRA information for those generated files.

There are two types of deviations defined:

- project deviations - deviations that are applicable for all PSoC Creator components
- specific deviations - deviations that are applicable for the specific component

Verification Environment

This section provides MISRA compliance analysis environment descriptions.

Component	Name	Version
Test Specification	MISRA-C:2004 Guidelines for the use of the C language in critical systems.	October 2004
Target Device	PSoC 6	Production
Target Compiler	GCC	5.4
Generation Tool	MDK	4.72a
	PSoC Creator	4.2
	Programming Research QA C source code analyzer for Windows	8.2-R
MISRA Checking Tool	Programming Research QA C MISRA-C:2004 Compliance Module (M2CM)	3.2

Project Deviations

Project Deviations are defined as permitted relaxations of the MISRA rules requirements that apply to source code that is shipped with PSoC Creator. The list of deviated rules is provided in the table below.

MISRA-C: 2004 Rule	Rule Description	Description of Deviation(s)
1.1	This rule states that code shall conform to C ISO/IEC 9899:1990 standard.	Some C language extensions (like interrupt keyword) relate to device hardware functionality and cannot be practically avoided. In the main.c file that is generated by PSoC Creator the non-standard main() declaration is used: "void main()". The standard declaration is "int main()" The number of macro definitions exceeds 1024 - program does not conform strictly to ISO:C90. Structures in the generated files are using designators in the initialization. This is allowed by ISO/IEC 9899:1999.
5.1	This rule says that both internal and external identifiers shall not rely on the significance of more than 31 characters.	The length of names based on user-defined names depends on the length of the user-defined names.
8.7	Objects shall be defined at block scope if they are only accessed from within a single function.	The object 'InstanceName_initVar' is only referenced by function 'InstanceName_Start', in the translation unit where it is defined. The intention of this publicly available global variable is to be used by user application.
11.3	This rule states that cast should not be performed between a pointer type and an integral type.	The cast from unsigned int to pointer does not have any unintended effect, as it is a consequence of the definition of a structure based on hardware registers.
14.1	There shall be no unreachable code.	Some functions that are part of the component API are not used within component API. Components API are designed to be used in user application and might not be used in component API.
21.1	Minimization of run-time failures shall be ensured by the use of at least one of: a) static analysis tools/techniques; b) dynamic analysis tools/techniques; c) explicit coding of checks to handle run-time faults.	Some components in some specific configurations can contain redundant operations introduced because of generalized implementation approach.

Documentation Related Rules

This section provides information on implementation-defined behavior of the toolchains supported by PSoC Creator. The list of deviated rules is provided in the table below.

MISRA-C: 2004 Rule	Rule Description	Description
1.3	Multiple compilers and/or languages shall only be used if there is a common defined interface standard for object code to which the languages/compilers/assemblers conform.	No multiple compilers and languages can be used at a time for PSoC Creator projects. The PK51 linker produces OMF-51 object module format. The GCC linker produces EABI format files. The MDK linker produces files of Arm ELF format.
1.4	The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers.	PK51 and GCC treat more than 31 characters of internal and external identifier length, and are case sensitive (e.g., Id and ID are not equal).
1.5	Rule states that floating-point implementation should comply with a defined floating-point standard.	Floating-point arithmetic implementation conforms to IEEE-754 standard.
3.1	All usage of implementation-defined behavior shall be documented.	For the documentation on PK51 and GCC compilers, refer to the Help menu, Documentation sub-menu, Keil and GCC commands respectively.
3.2	The character set and the corresponding encoding shall be documented.	The Windows-1252 (CP-1252) character set encoding is used. Some characters that are used for source code generation in PSoC Creator are not included in character set, defined by ISO-IEC 9899-1900 "Programming languages — C".
3.3	This rule states that implementation of integer division should be documented.	When dividing two signed integers, one of which is positive and one negative compiler rounds up with a negative remainder.
3.5	This rules requires implementation defined behavior and packing of bit fields be documented.	The use of bit-fields is avoided.
3.6	All libraries used in production code shall be written to comply with the provisions of this document, and shall have been subject to appropriate validation.	The C standard libraries provided with C51, GCC, and RVCT have not been reviewed for compliance. Some code uses memset and memcpy. The compiler may also insert calls to its vendor-specific compiler support library.

PSoC Creator Generated Sources Deviations

This section provides the list of deviations that are applicable for the code that is generated by PSoC Creator. The list of deviated rules is provided in the table below.

MISRA-C: 2004 Rule	Rule Description	Description of Deviation(s)
8.8	An external object or function shall be declared in one and only one file	For the PSoC 6, some objects are declared with external linkage, and their declarations are not in header files.
10.1	An integer constant of 'essentially signed' type is being converted to unsigned type on assignment.	An integer constant of 'essentially unsigned' type is converted to signed type on assignment in cyfitter_cfg.c.
11.4	Cast from a pointer to void to a pointer to object type.	The Amux API uses casts between a pointer to an object type and a different pointer to an object type.
14.1	Rule requires that there shall be no unreachable code.	The CYMEMZERO(), CYCONFIGCPYCODE () and CYCONFIGCPY() are not always used in the design.

MISRA-C: 2004 Rule	Rule Description	Description of Deviation(s)
15.2	An unconditional break statement shall terminate every non-empty switch clause.	This code structure is required to ensure that the GCC compiler produces efficient code for generated functions related to the AMuxSeq component.
15.3	The final clause of a switch statement shall be the default clause.	The code structure is required to ensure that the GCC compiler produces efficient code for generated functions related to the AMuxSeq component.

See Also:

- [Workspace Explorer](#)
- [Component Catalog](#)
- [Component Update](#)
- PDL API Documentation (located on the [Workspace Explorer Documentation tab](#))

Generated Files (FM0+)

Upon a successful build, PSoC Creator generates various files that become a part of your design. These files are listed in the [Workspace Explorer](#) under the **Source** tab. These files are specific to the device series (FM0+) and the selected compiler. The following lists and describes the files generated from a build.

File(s)	Description
pdl	
All files in this folder are copied from the installed peripheral driver library (PDL). See Options Dialog .	
General Files	
cydisabledsheets.h	File of defines for disabled schematic pages.
cyfitter.h	Defines all of the instance specific addresses calculated by the Code Generation step. This file is mainly intended for use by instance API implementations, although advanced users may find interesting information in this file.
cymetadata.c	This file defines all extra memory spaces that need to be included. This file is automatically generated by PSoC Creator.
pdl_user.h	User settings header file for Peripheral Driver Library.
project.h	This file includes all of the other header files found in this directory and its sub-directories. It exists for convenience sake, allowing you to include all of the generated headers with just one #include statement.
<device>.rom.icf	Generated IAR linker file for the device. Do not touch.
<device>.rom.id	Generated GCC linker file for the device.
<device>.rom.sct	Generated MDK scatter file for the device.

Component APIs:

The generated source will also include APIs for instantiated Components (e.g., counter_1.c, counter_1INT.c, counter_1.h), which will be listed in device-specific folders. If you do not want APIs generated for a specific instance, use the built-in parameter CY_SUPPRESS_API_GEN. See [Configure Component Parameters](#) for more information.

Results Files:

The Workspace Explorer also contains the **Results** tab, which contains the following files:

- <project>.cycdx – This contains XML information specific to Component debug windows. This is used by the debugger to determine what to display for the design. There is no reason for you to open or modify this file. For more information about this file, refer to the [Component Author Guide](#).
- <project>.rpt – This is the project report file. It contains information for how the device was programmed, including a section on how the target device's resources were utilized. Advanced users can review the information in this file to determine if there might be better ways to configure the design.

The **Results** tab may also contain various other files, including

- <project>.cyfit – This file is an internal database for PSoC Creator to hold data on the results of Code Generation. A user will not interact with this file directly. It is regenerated each time the project is built.
- <project>.elf – This file contains debugging information for the GCC tool chains. A user will not interact with this file directly.
- <project>.hex – Intel HEX file produced by combining <project>.ihx file with selected protection, configuration, and initialization settings. A user will not interact with this file directly.
- <project>.map – This file is produced by the linker. It contains details on how the device's memory was used, where functions and variables were placed and other details depending on the tool chain. A user will not interact with this file directly.
- <file>.lst – Code listing file showing initial c code and generated assembly. A user will not interact with this file directly.

See Also:

- [Workspace Explorer](#)
- [Component Catalog](#)
- [Component Update](#)

Source Code Control

PSoC Creator does not include integration with any source code control or revision control system. However you can use any source code control system to check PSoC Creator files in and out as you work.

This topic describes the various files and folders that should be committed to a source code control system. You should always include any file you have created or edited. If one or more of these files is not available in future check outs, the project may not load correctly, PSoC Creator may generate errors, or you may lose user-specific code.

See also Cypress Knowledge Base Article: *Revision Control for PSoC® Creator™ Projects - KBA86358*
[\(<http://www.cypress.com/?id=4&rID=76644>\)](http://www.cypress.com/?id=4&rID=76644)

Note The more files you include in source control, the more that may need to be checked out to be updated. Files should not be left in a "read-only" state, because this will limit PSoC Creator's functionality.

Project Files/Folders

The following are the project files and folders to include in source code control. There are [two types of projects](#): design and library. Design project files are contained in a folder named `<project>.cydsn`. Library project files are contained in a folder named `<project>.cylib`.

As indicated in the following list, some files are common to both types of projects, while others are specific to design projects only. As a general rule, include all of these files and subfolders in source code control.

- `<project>.cyprj` – This is the project file and is common to both types of projects. It is a container file for all the files in your project.
- `<project>.cydwr` – This is the design-wide resources file for your design project. See [Design-Wide Resources](#).
- `<project>.cyre` – This is the Keil reentrancy file, used for design projects only.
- `main.c` – This is the main application file, which contains all the user-specific application code for your design project.
- Added source files – If you added any C source, header, or assembly files to any project, they will be contained in the project folder or a subfolder.
- `/TopDesign (folder) /TopDesign.cysch (binary)` – This is the main schematic file for your design project.
- `/<added_project>.cydsn (folder)` – If you have additional design projects in your workspace, there will be one or more folders for those project files.
- `/<added_project>.cylib (folder)` – If you have additional library projects in your workspace, there will be one or more folders for those project files.
- `/<Component_name> (folder)` – If you have created a Component within your project, there will be a folder with that Component's name. That folder contains all the files you added and edited as part of creating the Component.
- `<project>.cyfit` – This file is an internal database for PSoC Creator to hold data on the results of Code Generation. It is regenerated each time the project is built. A user will not interact with this file directly. However, this file should be kept with the rest of a 'completed' project as it contains information needed when opening/reviewing a project.

XML-Based Description Files

PSoC Creator provides an optional feature to generate two XML-based files that describe the contents of various design entry files (schematic, schematic macro, symbol, UDB, and sheet template). See [Generating Description Files](#) for more information.

- `<project>.cysem` – This file contains all the semantically meaningful data from the source.
- `<project>.cyvis` – This file includes the cosmetic information from the source.

There can be several sets of these .cysem and .cyvis files in a project, depending on the types of design entry files you create in a project.

Optional Files

These files control the user experience (look and feel, open files in the workspace, and so on) but do not impact the application:

- `<project>.cyprj.username`
- `..><workspace>.cywrk`
- `..><workspace>.cywrk.username`

Generated Source and Output Files

If you have built a project, there will be a `Generated_Source` folder with various Component and compiler files. See [Generated Files](#) for descriptions of these files. Under usual circumstances, you do not need to include most of these files and folders in source code control, because they will be regenerated by PSoC Creator during a build. However, there are some cases where you may want to include them:

- If a generated file is located on disk and has not changed from a previous build, PSoC Creator will not regenerate that file. So you may wish to include some of these files in source code control to speed up future builds.
- If you have included any user-specific code in merge regions of any of these files, then you should include those files in source code control.
- If you have set the [Build Settings Skip Code Generation](#) option to "True," then PSoC Creator will not regenerate any of these files. Therefore they must be available or PSoC Creator will not build.
- You have modified the default Arm linker files (`cm3gcc.ld` or `cm3RealView.scat`), you should add them to source control.

External Source Files

If your project includes source files located outside the directories specified here, add them to your revision control system.

Temporary Folders

As part of a build, PSoC Creator generates temporary folders, such as `codegentemp`, `DP8051`, `Arm_GCC`, etc. None of these folders or files in them should be included in source code control.

Archiving Tool

As an alternate to manually selecting files for source code control, you can use the [Archiving Tool](#). This tool allows you to periodically save all the necessary project/workspace files to a location you specify. You can save the files in one zip file or save the entire directory structure unzipped. You can also choose different levels of files to save.

See Also:

- [Project Types](#)
- [Design-Wide Resources](#)
- [Reentrant Code in PSoC 3](#)
- [Generated Files](#)
- [Generating Description Files](#)
- [Build Settings](#)
- [Archiving a Workspace/Project](#)

Static Timing Analysis

As part of a successful build, PSoC Creator performs static timing analysis (STA) on your design automatically to determine various timing aspects, such as:

- Propagation Delay: Purely combinational delay from an input to an output.
- Clock-to-output Delay: Delay from a clock source through a register to an output.
- Setup Time: The minimum length of time that data must arrive at the input pin before the register's clock signal is asserted at the clock pin.
- Register-to-register Delay: Delay from the output of a register to the input of a register. The static timing analyzer will compute the maximum frequency if both registers have the same clock.

For more information on design practices and strategies about how to best use the STA report, refer to Application Note: [AN81623](#).

Static timing analysis identifies delays in a design's digital logic and computes the maximum frequency for each clock. The static timing analysis report shows the critical paths in the design that limit the clock frequency. If the actual clock frequency exceeds the calculated maximum frequency, the report indicates a timing violation in the design.

Static timing analysis only has access to the design during the build process, so it does not have knowledge of how the elements of the design will be used or of any changes made dynamically (such as firmware that changes a clock frequency). Because of these limitations, static timing analysis may issue warnings about paths that are not actually problematic, because of the way the design is used. If you have verified that the path in question is not used, you can safely ignore these warnings. For example, if a pin Component is configured as "Digital Input & Digital Output," static timing analysis may issue a warning about a path going to the output and then back in on the input of the same pin Component. If no configuration would ever result in this path being used, this warning can be safely ignored.

Note It may not be safe to ignore these warnings. Please review the warnings in the [Notice List Window](#) and address as necessary.

Static timing analysis does not have knowledge of how signals are generated or used outside the PSoC device. It can display delays related to such signals, but cannot automatically find timing violations.

To Open the STA Report:

After a successful build of your design, the STA report is provided as a standalone HTML report in the **Results** tab of the [Workspace Explorer](#).

Double-click the `<project_name>.html` file to open the report in your system's default web browser.

Report Layout:

The STA report contains a title, project information, various sections described under "Report Sections," and expanding/collapsing links to information in those sections.

Project Information

Every STA report contains the following project information:

- Project name and path
- Build time for this report
- Device used
- Device revision
- Temperature used for analysis
- Voltage used for analysis
- Voltage of each of the four I/O domains

Expanding/Collapsing Links

The links include:

- Expand All – This link expands all sections in the report, making the information visible.
- Collapse All – This link collapses all sections in the report, making only the section header visible.
- Show All Paths – This link expands the applicable section tables to show a multiline view of the full timing path.
- Hide All Paths This link collapses the applicable section tables to show only a single line for the timing path.

Report Sections:

The following sections may be included in the report. Except for "Timing Violations," any section that is empty will not be included.

- [Timing Violation](#)
- [Clock Summary](#)

- [Register to Register](#)
- [Asynchronous Clock Crossings](#)
- [Input to Output](#)
- [Input to Clock](#)
- [Clock to Output](#)
- [Input to Output Enable](#)
- [Clock to Output Enable](#)

Timing Violation Section

If there are no timing violations, this section just displays the text "No Timing Violations."

If there are violations and if the [DWR Operating Range](#) is set to Full Range, a note displays. The note indicates that changing the operating range may reduce the number of timing warnings. After the note, there is a table of one line entries for each combination of Violation, Source Clock, and Destination Clock.

The table is separated into three violation types: Setup, Hold, and Asynchronous, as shown in the following example.

Violation	Source Clock	Destination Clock	Slack (ns)
Setup:			
	Clock_1	Clock_2	-2.501
	Clock_1	Clock_3	-3.458
	Clock_2	Clock_3	-2.344
Hold:			
	Clock_1	Clock_2	-0.127
Asynchronous:			
	Pin_3	Clock_1	
	Pin_3	Clock_2	

Only one entry is included for each violating source / destination clock pair. The detail for each failing path is shown in later sections. On the STA report, you can click on an entry in the table to jump to the specific details.

If a particular violation type is not present in the design, that header will not be present in the table. The one line entry for each violating clock pair includes following fields:

- Source Clock: The source clock of the violating path
- Destination Clock: The destination clock of the violating path
- Slack: The slack time of the failure. For a Setup or Hold violation, this is always a negative number (indicating a violation). For an Asynchronous clock crossing violation, this field is left blank.

Clock Summary Section

This section is a short overview that represents the clocking frequency requirements and the achievable frequency with the current implementation of this design. The following is an example:

Clock	Domain	Nominal Frequency	Required Frequency	Maximum Frequency	Violation
BUS_CLK	CyBUS_CLK	48.000 MHz	48.000 MHz	Unrestricted	
Clock_1	CyBUS_CLK	24.000 MHz	24.000 MHz	21.845 MHz	Frequency
Clock_2	CyBUS_CLK	12.000 kHz	12.000 kHz	22.387 MHz	
Pin_3	Pin_3	18.000 MHz	18.000 MHz	45.239 MHz	
Pin_5	Pin_5	Unknown	Unknown	21.764 MHz	Unknown

The one line entry for each clock in the system has the following fields:

- Clock: The name of this clock. The first entry is always BUS_CLK. The remaining entries are shown in alphabetical order.
- Domain: This is the Clock Domain to which this clock belongs. Clocks in the same domain are synchronous to each other.
- Nominal Frequency: This is the frequency that this “solved” by the tool from the desired frequency in the design. It is the direct value of the source clock divided by the divider setting. There is no accommodation for accuracy or jitter due to synchronization with MASTER_CLK. In the case of a clock where the frequency cannot be determined (i.e. clock coming from a pin), the frequency is displayed as “Unknown”.
- Required Frequency: This is the frequency at which paths using this clock must be able to meet timing. This clock is the Nominal clock with the addition of worst case synchronization jitter. Required Frequency (MHz): This is the clock frequency specified in the design. If this is an Asynchronous clock that doesn’t have a clock frequency property, it is displayed as “Unknown”.
- Maximum Frequency: This is the frequency at which this clock can safely run. It is calculated based on the slowest path in the design that impacts this clock. If a clock is not restricted “N/A” will appear in this column.
- Violation: There are two possible violations: "Frequency" or "Unknown." Frequency is shown when the Max Frequency is less than the Required Frequency. Unknown is shown when the Required Frequency is unknown. These violations are shown in red. If there is no violation, this field is blank.

Register to Register Section

This section is for register to register timing paths where the source and destination clocks are either the same, or they are synchronous to each other. Any asynchronous clock crossing is described in the "Asynchronous Clock Crossing" section. However the paths between an asynchronous clock and itself are included here.

There are two major subsections within this section: [Setup](#) and [Hold](#).

Setup Subsection

This subsection is further divided into Source clock and then Destination clock. Each of these clocks is listed in alphabetic order. If the negative edge of the clock is used that is considered a distinct clock and the negative edge is denoted along with the clock name.

The Source clock heading lists the Source clock name and required frequency. For example:

Source clock: Clock_1 (Required Freq. 24.000 MHz)

The Destination clock heading lists the Destination clock name and required frequency. It also includes the requirement for the path delay. This is dependent on the combination of the source and destination clocks. It can be impacted by the use of opposite clock edges or different clocks that are synchronous to BUS_CLK. For example the following destination clock is different from the source clock, but both are synchronous to BUS_CLK, which in this example is running at 48 MHz.

Destination clock: Clock_2 (Required Freq. 12.000 MHz)
 Path Delay Requirement: 20.833ns (48 MHz)

The following is an example of three timing paths with the second entry expanded to show the complete path.

Source	Destination	FMax (MHz)	Delay (ns)	Slack (ns)	Violation
dff_reg1:macrocell.mc_q	Net_5:macrocell.mc_d	8.000	12.500	-2.500	SETUP
dff_reg1:macrocell.mc_q	Net_6:macrocell.mc_d	11.000	9.091	0.909	

Type	Location	Fanout	Instance/Net	Source	Dest	Delay (ns)
macrocell	U(3,1)	1	dff_reg1	.cr_clk	.q	1.250
Route		3	dff_reg1	.q	.main_0	3.608
macrocell	U(3,2)	1	Net_5	.main_0	.mc_d	2.810
macrocell	U(3,2)	1	Net_5		Setup	0.875
Clock					Skew	0.548

dff_reg1:macrocell.mc_q	Net_7:macrocell.mc_d	11.500	8.696	1.304	
-------------------------	----------------------	--------	-------	-------	--

The one line entry for each path has the following fields:

- Source: The register start of the path
- Destination: The register destination of the path
- FMax (MHz): The maximum frequency based on this specific path in MHz (this is 1/Delay).
- Delay (ns): Delay along the path in ns including the setup time and any clock skew.
- Slack (ns): Slack is the (Path Delay Requirement - Actual Delay). Display any slack less than 0 in red.
- Violation: The only violation type is "SETUP". Display "SETUP" in red for negative slack and nothing in the field otherwise.

Entries are listed from the least slack to most slack. All paths that violate setup time are included in the report. After that, up to another 10 entries are shown for each clock pair.

Each single line entry can be expanded to show a detailed complete path. Each entry includes the following fields:

- Type: This is the type of function involved:
- Route: Used for all routes
- Macrocell
- Datapath
- Control: Indicates Control register
- Status: Indicates Status register
- Clock: Used for clock skew entries
- IO: Indicates a pin on the device
- Location: This is the location of the cell indicated in the type field. It is present for all types except Route and Clock.

- U(x,y): Format for all cells in the UDB array (x,y are the coordinates of the UDB)
- Pi[j]: Format for a pin (i is the port number and j is the pin number within the port)
- Fanout: This indicates the fanout of the signal. This is expected to be 1 except for Routes where it should indicate the number of destinations driven by this same signal.
- Instance/Net: This is the Instance or Net name associated with this piece of the route.

Note Macrocell names might not match the original Component name. The fitter may combine macrocells with other nets and macrocells. This process can cause some name information to be lost. Macrocells that have been combined with nets may inherit the net name, such as "Net_73".

- Source and Dest: These are the source and destination pins on the cells at both ends of this portion of the route. The destination field is also used by itself (source empty) for some special cases:
- Setup: Special case included at the end of each setup path to indicate the setup required to the register.
- Skew: Special case included only for routed clocks (Global clocks do not get a skew entry).
- Delay: Incremental delay in ns for this portion of the overall path. The sum of all the incremental delay entries must equal the Delay in the one line summary.

Hold Subsection

This subsection is further divided into Source clock and then Destination clock. Only source / destination pairs are present when at least one of those clocks is a routed clock. The naming and ordering of the subsections is the same as those in the "[Setup](#)" subsection, except that clock frequency is not included for the Source and Destination headings. For example:

```
Source clock: Clock_1
Destination clock: Clock_2
```

The following is an example of three timing paths with the second entry expanded to show the complete path.

Source	Destination		Slack (ns)	Violation
dff_reg1:macrocell.mc_q	Net_5:macrocell.mc_d		-0.541	HOLD
dff_reg1:macrocell.mc_q	Net_6:macrocell.mc_d		2.965	

Type	Location	Fanout	Instance/Net	Source	Dest	Delay (ns)
macrocell	U(3,1)	1	dff_reg1	.cr_clk	.q	1.000
Route		3	dff_reg1	.q	.main_0	3.122
macrocell	U(3,2)	1	Net_5	.main_0	.mc_d	2.523
macrocell	U(3,2)	1	Net_5		Hold	-0.120
Clock					Skew	-3.560

dff_reg1:macrocell.mc_q	Net_7:macrocell.mc_d	5.234	
-------------------------	----------------------	-------	--

The one line entry for each path is the same format as the "Setup" subsection entries, with the following changes:

- FMax and Delay are not present for Holds
- Slack is calculated in the same way that Delay is calculated for "Setup." It is the sum of all the Delay entries present in the detailed path.

- Violation type is "HOLD" for a hold violation

Entries are listed from the least slack to most slack. All paths that violate hold time are included in the subsection. After that, up to another 10 entries are shown for each clock pair.

Each single line entry can be expanded to show a detailed complete path. Each entry is similar to those in the "[Setup](#)" section, with the following changes:

- Delays are calculated based on a best case path instead of a worst case path
- The special case entries for Dest are:
- Hold: Special case included at the end of each hold path to indicate the hold required to the register. Positive hold requirements are indicated with a negative number such that the sum of the incremental delays totals to be the slack time.
- Skew: Always present since without clock skew there can't be a hold time violation in this architecture.
- Delay: Incremental delay sums to the slack time. The sign of the Hold and Clock skew entries needs to be such that these entries sum properly for that calculation.

Asynchronous Clock Crossing Section

This section shows all paths between clock domains where the source and destination clocks are not synchronous to each other.

This section is further divided into subsections for the Source clock and the Destination clock. Only source / destination pairs are present when these clocks are asynchronous to each other and they are not both synchronous to BUS_CLK. The naming and ordering of the subsections is the same as for the "[Setup](#)" subsection, except that clock frequency is not included for the Source and Destination headings. For example:

```
Source clock: Clock_1
Destination clock: Clock_2
```

The following is an example of three timing paths with the second entry expanded to show the complete path.

Source	Destination		Delay (ns)		
dff_reg1:macrocell.mc_q	Net_5:macrocell.mc_d		12.500		
dff_reg1:macrocell.mc_q	Net_6:macrocell.mc_d		9.091		
Path Details:					
Type	Location	Fanout	Instance/Net	Source	Dest
macrocell	U(3,1)	1	dff_reg1	.cr_clk	.q
Route		3	dff_reg1	.q	.main_0
macrocell	U(3,2)	1	Net_5	.main_0	.mc_d
macrocell	U(3,2)	1	Net_5		Setup
Clock					Skew
dff_reg1:macrocell.mc_q		Net_7:macrocell.mc_d		8.696	

The one line entry for each path is the same format as the Setup entries with the following changes:

- FMax and Slack are not present for Clock Crossings
- Delay is calculated to provide information to the user, but it is not used to compute whether a timing requirement is met.

- The Violation field is not present.

Entries are listed from the most delay to least delay. Up to 10 entries are shown for each clock pair.

Each single line entry can be expanded to show a detailed complete path. The detailed entries for the path are identical to the entries that are present for [Setup](#).

Input to Output Section

This section shows combinatorial paths through the device. It contains one single line entry for the longest combinatorial path for each source / destination pair. These entries are ordered from the longest delay to the shortest delay.

The following is an example of three timing paths with the second entry expanded to show the complete path.

Source	Destination		Delay (ns)			
Pin_3(0):iocell._fb	Pin_4(0):iocell.pad_out		56.823			
Pin_3(0):iocell._fb	Pin_6(0):iocell.pad_out		54.113			
 						
Type	Location	Fanout	Instance/Net	Source	Dest	Delay (ns)
Pin	P5[6]	1	Pin_3(0)	.pad_in	.fb	15.258
Route		2	Net_3	.fb	.main_0	5.714
macrocell	U(3,2)	1	Net_4	.main_0	.q	3.350
Route		1	Net_4	.q	.input	6.297
Pin	P5[2]	1	Pin_6(0)	.input	.pad_out	23.495
 						
Pin_5(0):iocell._fb		Pin_6(0):iocell.pad_out		42.696		

The one line entry for each path has the following fields:

- Source: The beginning of this combinatorial path
- Destination: The final destination of this combinatorial path
- Delay (ns): Delay along the path in ns.

Each single line entry can be expanded to show a detailed complete path. The detailed entries for the path are identical to the [Setup](#) entries, except that the special case entries for clock skew and setup are never present.

Input to Clock Section

This section shows the path into the device that terminates at a clocked element.

This section is further divided into Destination clock subsections based. These subsections are ordered alphabetically and labeled with the name of the clock. For example:

Destination clock: Clock_1

The following is an example of three timing paths with the second entry expanded to show the complete path.

Source		Destination			Delay (ns)	
Pin_3(0):iocell._fb		Net_7:macrocell.mc_d			31.763	
Pin_3(0):iocell._fb		Net_6:macrocell.mc_d			24.657	
Type	Location	Fanout	Instance/Net	Source	Dest	Delay (ns)
Pin	P5[6]	1	Pin_3(0)	.pad_in	.fb	15.258
Route		2	Net_5	.fb	.main_0	5.714
macrocell	U(3,2)	1	Net_6	.main_0	.mc_d	2.810
macrocell	U(3,2)	1	Net_6		Setup	0.875
Pin_5(0):iocell._fb		Net_6:macrocell.mc_d			22.376	

The longest path from each input to the clock is shown with a single line entry. These entries are ordered from longest to shortest. These entries are similar to the "[Input to Output](#)" section, except there is a setup entry for each path.

Clock to Output Section

This subsection is present to show the path from a clocked element out of the device.

This subsection is further divided into subsections based on the Source clock. These subsections are ordered alphabetically and labeled with the name of the clock. For example:

Source clock: Clock_1

The following is an example of three timing paths with the second entry expanded to show the complete path.

Source		Destination			Delay (ns)	
dff_reg1:macrocell.mc_q		Pin_4(0):iocell.pad_out			43.873	
dff_reg1:macrocell.mc_q		Pin_6(0):iocell.pad_out			30.459	
Type	Location	Fanout	Instance/Net	Source	Dest	Delay (ns)
macrocell	U(3,1)	1	dff_reg1	.cr_clk	.q	1.250
Route		2	dff_reg1	.q	.input	5.714
Pin	P5[2]	1	Pin_6(0)	.input	.pad_out	23.495
dff_reg2:macrocell.mc_q		Pin_6(0):iocell.pad_out			29.745	

The longest path from a clock to each output is shown with a single line entry. These entries are ordered from longest to shortest. These entries have the same expansion capability and all the same fields as the "[Input to Output](#)" section.

Input to Output Enable Section

This section is identical to the "[Input to Output](#)" section except in this case the destination is the output enable of an output port instead of the data for an output port. The reporting is handled identically except each path has two entries, one for turning the output on (TURNON) and one for turning the output off (TURNOFF). These two cases are processed as two different entries even if the delays are the same.

The following is an example of two timing paths with the second entry expanded to show the complete path.

Source	Destination			Type	Delay (ns)	
Pin_9(0):iocell._fb	Pin_6(0):iocell.pad_out			TURNON	49.625	
Pin_9(0):iocell._fb	Pin_6(0):iocell.pad_out			TURNOFF	49.625	
 						
Type	Location	Fanout	Instance/Net	Source	Dest	Delay (ns)
Pin	P3[5]	1	Pin_9(0)	.pad_in	.fb	15.258
Route		1	Net_26	.fb	.main_0	5.714
macrocell	U(2,1)	1	tmpOE__Pin6_net_0	.main_0	.q	3.350
Route		1	tmpOE__Pin6_net_0	.q	.oe	6.331
Pin	P5[2]	1	Pin_6(0)	.oe	.pad_out	18.972

In order to distinguish these two entries an additional field is added to the one line entry:

- Type: Either TURNON or TURNOFF depending on whether the output is enabled or disabled.

Clock to Output Enable Section

This section is identical to the "[Clock to Output](#)" section except in this case the destination is the output enable of an output port instead of the data for an output port. The reporting is handled identically except each path has two entries, one for turning the output on (TURNON) and one for turning the output off (TURNOFF). These two cases are processed as two different entries even if the delays are the same.

The following is an example of two timing paths with the second entry expanded to show the complete path.

Source	Destination			Type	Delay (ns)	
dff_reg3:macrocell.mc_q	Pin_6(0):iocell.pad_out			TURNON	22.869	
dff_reg3:macrocell.mc_q	Pin_6(0):iocell.pad_out			TURNOFF	22.869	
 						
Type	Location	Fanout	Instance/Net	Source	Dest	Delay (ns)
macrocell	U(1,1)	1	dff_reg3	.cr_clk	.q	1.250
Route		1	dff_reg3	.q	.oe	5.714
Pin	P5[2]	1	Pin_6(0)	.oe	.pad_out	15.905

In order to distinguish these two entries an additional field is added to the one line entry:

- Type: Either TURNON or TURNOFF depending on whether the output is enabled or disabled.

See Also:

- [Generated Files](#)
- [Workspace Explorer](#)

CyPrjMgr Command Line Tool

cypjmgr.exe is a command line tool that exposes common project management functionality to be used through the command line. The tool can be invoked from the command line and used to perform various functions on a workspace/project. It also provides the flexibility to set the build configuration from the command line.

Syntax:

```
cyprjmgr
[-h]
[-ver]
[-wrk <workspace_name>]
[-clean]
[-build]
[-rebuild]
[-archive <archive_level> <archive_as_zip>]
[-t <toolchain>]
[-c <config>]
[-p <TopProject>]
[-n <TopDesign>]
[-d <selectedDev>]
[-m <paramsFile>]
[-import <Source_Project> <Source_Component>]
[-rename <Component_name> <new_name>]
[-delete <Component_name>]
[-exclude <Component_name>]
[-l <NewPrjName>]
[-s]
[-v <visibility>]
[-prj <Target_Project>]
[-cmp <Target_Component>]
[-addprj <prj_path>]
[-cp <path>]
[-con <Target_Project>]
[-batch <file_name>]
[-updateComp <source_project> <source_Component>]
[-updatePrj <source_project>]
[-updateInst]
[-updateDWInst]
[-forceWrite]
[-noCustBuild]
[-noRefresh]
[-ol <compiler optimization level>]
[-warn <High|Low|None>]
[-buildPreCompCust <Project>]
[-updateInstIfNeeded]
[-ignoreDepsWarning]
[-allowIllegalUpdates]
[-generateDescFiles]
[-verifyDescFileEnabled]
[-verifyDescFileContents]
[-export <IDE>]
```

`[-pdlPath <path>]`

Tool-Wide Options:

The following options are tool wide:

<code>-h</code>	Displays this help message
<code>-ver</code>	Displays the version and build number of cyprjmgr

Chosen Workspace Options:

The following options apply to the chosen workspace:

<code>-wrk, -w</code>	Specifies the workspace to be used
<code>-prj</code>	Specifies the target project on which all the command line options will be targeted.
	In case target project is not specified, Top Project of the workspace becomes the target project
<code>-p</code>	Sets the Top Project in the workspace
<code>-cmp, -o</code>	Specifies the target Component on which all the library options will be targeted.
	In case target Component is not specified, the Top Block of the Target Project becomes the Target Component
<code>-addprj</code>	Adds an existing project <code><prj_path></code> to the workspace
<code>-l</code>	Adds a new empty library project with name <code><NewPrjName></code> to the workspace
<code>-d</code>	Sets the selected device of projects to be built
<code>-cp</code>	Copies the entire workspace to the location specified by <code><path></code> , all command line options will act on the copy created

Entire Workspace Options

The following four options target the entire workspace unless the target project is set with `-prj`:

<code>-clean</code>	Cleans the workspace/project
<code>-build</code>	Builds the workspace/project
<code>-rebuild</code>	Rebuilds the workspace/project
<code>-archive</code>	Archives the workspace/project with different archiver levels (complete/typical), and as zip or nozip

Target Project Options:

The following options apply to the target project:

<code>-t</code>	Sets the tool chain the action should use (Keil, Arm etc.)
<code>-c</code>	Sets configuration the action should use (Debug, Release)
<code>-ol</code>	Sets compiler optimization level
<code>-warn</code>	Sets compiler warning level
<code>-n</code>	Sets the Top Design of the Top Project
<code>-m</code>	Parameters file that will override the default parameter values of the schematic in the TopDesign of the Top Project
<code>-import</code>	Imports the Source Component from the Source Project into the target project of the workspace

```

-export      Exports the project to the target IDE. Valid targets are EWA, Eclipse and
uvision
-rename     Renames Component_name in target project of the workspace to new_name
-delete     Deletes Component from the disk
-exclude    Excludes Component from the target project of the workspace
-s          Lists the external dependencies of the target project
-v          Sets the visibility of the target Component to true/false
-con       Checks the consistency of the target project of the workspace
-batch     Reads a file containing a series of commands, one on each line. Executes the
commands
           one by one. When batch option is used, all other optional switches are
ignored
-updateComp   Updates a Component in the Target Project from the source
project
-updatePrj    Updates the Target Project from the Source Project
-updateInst  Components
             Updates the instances on the schematic with the latest
Components
-forceWrite   Makes read-only files writable and then makes the change
-noCustBuild  Delay building of customizer DLLs until the end (e.g., during
imports)
-noRefresh    Disable updates from the refresh manager (Use with extreme
care)
-buildPreCompCust Build customizer for a project
-updateDWInst  Updates the design-wide instances with the latest Components
-updateInstIfNeeded Update instances to minimum valid version
-ignoreDepsWarning Suppress SystemDepNotFoundOnDisk warning while building
primitives
-allowIllegalUpdates Allows updating instances to latest version, even if illegal
-generateDescFiles For all specified projects that have 'Generate description
files' enabled,
           generates the description files.
-files        Verifies that all specified projects have 'Generate
description files' enabled.
-verifyDescFileEnabled Verifies that all specified projects (that have 'Generate
description files'
-current version of their
-pdlPath      Sets the path to the PDL file for the project

```

Usage Scenarios and Examples

-clean / -build / -rebuild

```

cyprjmgr.exe -w workspace -clean
cyprjmgr.exe -w workspace -build
cyprjmgr.exe -w workspace -rebuild

```

This option will clean/build/rebuild the workspace respectively. Only one out of clean, build, or rebuild can be used in one run of the tool.

-h

```
cyprjmgr.exe -h
```

This will display help message for the tool

-t

```
cyprjmgr.exe -w workspace -build -t "Arm CM3-GCC 4.9-2015-q1-update"
```

This will build the workspace with the toolchain specified. If no toolchain is specified by the user, the workspace will build with the toolchain, with which it was built the last time.

-c

```
cyprjmgr.exe -w workspace -build -c Release
```

This will build the workspace with the build config specified (Release, in this example)

-p

```
cyprjmgr.exe -w workspace -build -p Design01
```

This will set the project *Design01* in the workspace as the Top Project and build the workspace

-n

```
cyprjmgr.exe -w workspace -build -n Component01
```

This will set the Component *Component01* in the Top Project of the as the Top Component and build the workspace

-d

```
cyprjmgr.exe -w workspace -build -d CY8C3866AXI-040
```

This option will set the device for all Projects in the workspace and build it.

```
cyprjmgr.exe -w workspace -build -d CY8C3866AXI-040 -prj Design01
```

This option will set the device for the target project and build it.

```
cyprjmgr.exe -w workspace -build -d CY8C3866AXI-040 -t "DP8051-Keil Generic" -prj Design01
```

This switch sets the device and toolchain for the target project 'Design01.cyprj' and builds it

-rev

```
cyprjmgr.exe -w workspace -rev ES1
```

This switch sets the revision of the selected device for all the Projects in the workspace.

```
cyprjmgr.exe -w workspace -build -d CY8C3866AXI-040 -rev ES1 -prj Design01
```

This option sets the device revision to 'ES1' for the target project 'Design01.cyprj' and builds it.

-m

```
cyprjmgr.exe -w workspace -build -m params_file
```

This option will read in parameters and their values from a text file, and override those values in the schematic.

Format of the params file :

The params file accepts parameters and their values as name=value pairs. The name of the instance must be given as *inst_name*=value.

For example,

inst_name=and_1

NumTerminals=8

TerminalWidth=4

inst_name=Counter_1

Resolution=16

This will change the values of parameters *NumTerminals* and *TerminalWidth* in the instance *and_1*, and *Resolution* in the instance *Counter_1*.

-prj

```
cyprjmgr.exe -w workspace -prj Project01
```

This option sets Project01 in the Workspace as the Target Project. All library options on the same command line will act on Project01. If no Target Project is selected, the Top Project of the Workspace becomes the Target Project.

```
cyprjmgr.exe -w workspace -build -prj Project01
```

This option overrides build to work only with *Project01* (only if *Project01* is in *workspace*) and its dependencies instead of building the entire workspace. This is equivalent to selectively building a project in the GUI.

-o, -cmp

```
cyprjmgr.exe -w workspace -o Component01
cyprjmgr.exe -w workspace -cmp Component01
```

This option sets Component01 as the Target Component in the Target Project. All library options on the same command line targeting a Component will act on the Target Component. If no Target Component is selected, the Top Component of the Target Project becomes the Target Component.

-import

```
cyprjmgr.exe -w workspace -import SourceProject SourceComponent
```

This option will import the Source Component from the Source Project to the Top Project of the workspace.

```
cyprjmgr.exe -w workspace -import SourceProject SourceComponent -prj TargetProject
```

This option will import the Source Component from the Source Project into the Target Project of the Workspace.

-rename

```
cyprjmgr.exe -w workspace -rename ComponentName NewComponentName
```

This option will rename the Top Component in the Top Project from *ComponentName* to *NewComponentName*.

```
cyprjmgr.exe -w workspace -rename ComponentName NewComponentName -prj Project01
```

This option will rename the Top Component in Project01 of the Workspace

```
cyprjmgr.exe -w workspace -rename ComponentName NewComponentName -prj -Project01 -o  
Component02
```

This option will rename Component02 in Project01 of the Workspace

-delete

```
cyprjmgr.exe -w workspace -delete Component01
```

This option removes Component01 of the Top Project of the Workspace from the disk.

```
cyprjmgr.exe -w workspace -delete Component01 -prj Project01
```

This option removes Component01 of Project01 of the Workspace from the disk.

-exclude

```
cyprjmgr.exe -w workspace -exclude Component01
```

This switch removes the 'Component01' instance from the 'Top Project' of the Workspace.

```
cyprjmgr.exe -w workspace -exclude Component01 -prj Project01
```

This switch removes the 'Component01' instance from 'Project01' of the Workspace.

-l

```
cyprjmgr.exe -w workspace -l LibraryName
```

This option adds a new empty library project to the Workspace.

-s

```
cyprjmgr.exe -w workspace -s
```

This option lists the external dependencies of the Top Project in the Workspace.

```
cyprjmgr.exe -w workspace -s -prj Project01
```

This option lists the external dependencies of Project 01 in the Workspace.

-v

```
cyprjmgr.exe -w workspace -v false -o Component01
```

This option will set the visibility of symbol Component01 in the Top Project of the Workspace to false.

```
cyprjmgr.exe -w workspace -v false -o Component01 -prj Project01
```

This option will set the visibility of the symbol Component01 in Project01 of the Workspace.

-addprj

```
cyprjmgr.exe -w workspace -addprj Project01
```

This option adds an existing CyDesigner project, Project01 to the Workspace.

-con

```
cyprjmgr.exe -w workspace -con
```

This option checks the consistency of the Top Project of the Workspace. i.e.

1. There is no file in the Target Project folder that the project does not know about.
2. All files that the project knows about, are at the location pointed by their Canonical Name property.

```
cyprjmgr.exe -w workspace -con -prj Project01
```

This option checks the consistency of Project01 in the Workspace.

-cp

```
cyprjmgr.exe -w workspace -cp NewLocation
```

This option will copy the entire workspace folder to NewLocation. All other options given on the same command line will now act on the NewLocation Workspace.

-ver

```
cyprjmgr.exe -ver
```

This option displays the version number of the application.

-batch

```
cyprjmgr.exe -w workspace -batch file_name
```

This option reads a text file that has a series of CyPrjMgr commands, one per line. The CyPrjMgr tool reads each line, executes the command, and if successful goes to the next command. If a command fails, the tool exits with failure message.

When -batch is used, all other optional switches on the command line are ignored.

-updateComp

```
cyprjmgr.exe -w workspace -updateComp source_project source_Component
```

This option updates the Component in the Top Project of the workspace.

```
cyprjmgr.exe -w workspace -prj Project01 - updateComp source_project source_Component
```

This option updates the Component in the Project01 of the workspace.

The update of Component is based on the following rules:

- The tool looks for a Component in the target project with the same base name as the source Component. If it does not find, the Component is imported to the target project.
- If it finds the target Component, it compares the files in the source Component to the corresponding file in the target Component in the following way :
 - If there is any file in the source Component that does not exist in the target, UPDATECOMP.
 - If a file with the same name as in source exists in the target Component, the contents of the two files are compared. If their contents are different, UPDATECOMP.
- If all files are same, there is nothing to be done.

UPDATECOMP : Remove the target Component from the target project. Import the source Component into the target project.

-updatePrj

```
cyprjmgr.exe -w workspace -updatePrj source_project
```

This option will update the Top Project of the workspace.

```
cyprjmgr.exe -w workspace -prj Project01 -updatePrj source_project
```

This option will update the project Project01 of the workspace.

The update of the project is based on the following rules:

- The toolchains of the source project and target project are compared. If they are different, UPDATEPRJ.
- From the confirmed identical toolchains, compare the source project toolchain settings with its counterpart in the target project. If they are different, UPDATEPRJ.
- Compare the non-Component files of the source project with the target project (i.e., main.c, .cydwr file, .cyprj file). If any of these is different from its counterpart. UPDATEPRJ.
- If this step is reached, there is nothing to update.

UPDATEPRJ : The target project is removed from the workspace and its directory is deleted. The source project and all its files/folders/Components are copied in place of the target project and the project is added to the workspace.

-archive

```
cyprjmgr.exe -w workspace -archive complete zip
```

This archives the complete workspace into zip file.

```
cyprjmgr.exe -w workspace -archive typical nozip -prj Design01
```

This archives only project Design01 with typical archive level. The archived project will be exact replica of the source project (in the example shown above, Design01 is the source project).

The archive of the workspace/project is based on the following rules:

- It archives the whole workspace if no target project provided, else it archives only the specified project.
- Two different levels of archive available, complete and typical. Using ‘complete’ archive level the whole project/workspace is archived whereas with ‘typical’ archive level only.
- There is an option available to archive as ‘zip’ or ‘nozip’, which allows the user to archive into zip or just the copy of the content.

-updateInst

```
cyprjmgr.exe -w workspace -updateInst
```

This switch updates all the instances of the Components on the schematic including the boot Component for the project, for all the projects in the specified workspace with the latest versions of the Components available.

```
cyprjmgr.exe -w workspace -updateInst -prj Design01
```

This switch updates all the instances of the Components on the schematic including the boot Component for the target project “Design01” with the latest versions of the Components available.

Update instance of the workspace/project is based on the following rules

- It updates all instances on the schematic with the latest Component instances.
- If a target project is provided, it updates the instances of the Components in the specified project only. If the project is not specified,,it updates all the projects in the workspace with instances of the latest versions of all the Components.

-forceWrite

```
cyprjmgr.exe -w workspace -d CY8C3866AXI-040 -forceWrite
```

This switch sets the device for all the Projects in the specified workspace, even if the workspace and projects are read-only.

Force writing of the workspace/project is based on the following rules:

- The workspace and project files are made writable if they are read-only
- Any other switch passed to the ‘cyprjmgr’ tool, which changes files, need to use this option to save the changes to read-only files.
- If this option is not provided changes will not be saved on read-only files.

-noCustBuild

```
cyprjmgr.exe -w workspace -build -noCustBuild
```

This switch delays the building of the customizer DLLs until the end.

-noRefresh

```
cyprjmgr.exe -w workspace -build -noRefresh
```

This switch disables the refresh manager while building the project.

-buildPreCompCust

```
cyprjmgr.exe -w workspace -build -buildPreCompCust project
```

This switch builds customizer for a specific project the building of the customizer DLLs until the end.

-updateDWInst

```
cyprjmgr.exe -w workspace -build -updateDWInst
```

This switch updates the instances on the schematic to the latest version

-ol

```
cyprjmgr.exe -w workspace -build -ol level
```

This switch sets the compiler optimization level. The level is compiler specific.

-warn

```
cyprjmgr.exe -w workspace -build -warn [High|Low|None]
```

This switch sets the compiler warning level.

-updateInstIfNeeded

```
cyprjmgr.exe -w workspace -build -updateInstIfNeeded
```

This switch updates Components to the minimum valid level.

-ignoreDepsWarning

```
cyprjmgr.exe -w workspace -build -ignoreDepsWarning
```

This switch suppresses SystemDepNotFoundOnDisk warning while building primitives.

-allowIllegalUpdates

```
cyprjmgr.exe -w <workspace> -updateInst -allowIllegalUpdates
```

Updates the Component to the latest version, even if the version is not allowed from cystate files.

-export

```
cyprjmgr.exe -w workspace -prj Proj1 -export uVision
```

This switch exports the project to uVision. The -export switch can take one of the following values: eclipse, EWA (refers to IAR), or uVision.

-generateDescFiles

```
cyprjmgr.exe -w workspace -generateDescFiles
```

For all the projects in the workspace that have 'Generate description files' enabled, generates the description files.

```
cyprjmgr.exe -w workspace -prj <project_name> -generateDescFiles
```

For the specified project in the workspace, if 'Generate description files' is enabled, generates the description files.

-verifyDescFileEnabled

```
cyprjmgr.exe -w workspace -verifyDescFileEnabled
```

Verifies that all the projects in the workspace have 'Generate description files' enabled.

```
cyprjmgr.exe -w workspace -prj <project_name> -verifyDescFileEnabled
```

For the specified project in the workspace, verifies that 'Generate description files' is enabled.

-verifyDescFileContents

```
cyprjmgr.exe -w workspace -verifyDescFileContents
```

Verifies that all the projects in the workspace (that have 'Generate description files' enabled) have generated files that are in-sync with the current version of their source files.

```
cyprjmgr.exe -w workspace -prj <project_name> -verifyDescFileContents
```

For the specified project in the workspace (if 'Generate description files' is enabled), verifies that it has generated files that are in-sync with the current version of their source files.

-pd1Path

```
cyprjmgr.exe -w <workspace> -build -pd1Path <path>
```

Sets the PDL path used by the top project to the specified path and then builds the project.

CyHexTool Command Line Tool

The hex file postprocessor (CyHexTool) is a standalone command line tool that combines data from several hex files to produce the programming file. The output file is in Intel hex format.

This tool is applicable to projects targeting PSoC 3 projects and older PSoC 5LP projects. For PSoC 4 projects and newer PSoC 5LP projects, use the [CyElfTool](#) tool instead.

Syntax:

```
cyhextool -o <out.hex> -f <in.hex> -id <XXXXXXXXXX>
[-ecc <ON|OFF|HexFile>] [-cunv <XXXXXXXXXX>] [-wonv <XXXXXXXXXX>] [-ee <eprom.hex>]
[-prot <protect.hex>] [-a <PROGRAM={0},...>]
```

Normal Options:

The following table lists and describes the various arguments for normal projects:

Argument	Description
-o <out.hex>	Specify output file name (required).
-f <in.hex>	Specify input file name (required). This file is normally produced by the linker.
-id XXXXXXXX	Specifies the target device ID (4 bytes).
-ecc <ON OFF HexFile>	Enable ECC (ON), disable ECC (OFF), or specify a user-defined Intel hex file to program the ECC bits. If this option is omitted, ECC will be disabled.
-cunv <XXXXXXXXXX>	Specify hexadecimal customer NV latch data (4 bytes for PSoC 3/PSoC 5LP only). The default is all zeros.
-wonv <XXXXXXXXXX>	Specify hexadecimal write-only NV latch data (4 bytes for PSoC 3/PSoC 5LP only). The default is all zeros.
-ee <eprom.hex>	Specify user-defined Intel hex file to program EEPROM (if needed).
-prot <protect.hex>	Specify user-defined Intel hex file to program Flash protection bits (64 bytes for PSoC 3/PSoC 5LP only). The default is all zeros.
-a <PROGRAM={0},...>	Set section sizes (such as Flash/ECC size).
-meta XXXX	Specifies the metadata (debugging enabled, silicon rev). The meta value is always exactly two bytes.
-rev <XX>	Specifies the target device revision (1 byte).
- endian <b, 1>	Specifies the endianess of the device.

Bootloader/Bootloadable Options:

The following table lists and describes the various additional arguments for bootloader projects:

Argument	Description
-bl <Hex File>	Specifies the bootloader flash image hex file
-acd <Hex File>	Specifies the *.cyacd bootloadable output file
-acdStart <XXXX>	Specifies the starting address of the bootloadable image
-e <XXXX>	Specifies the entry address of the bootloadable
-blkcs <path to file>	Specifies the file containing the address of the bootloader checksum

Argument	Description
-blsize <path to file>	Specifies the file containing the address of the bootloader size
-blChkType <X>	Specifies the bootloader packet checksum type (1=basic summation, 2=CRC)
-blVer <XXXXXXXXXX>	Specifies the bootloader metadata (BtldrVer, LoadableId, LoadableVersion, Cust ID)
-metaRow <0, 1>	Specifies the bootloader metadata row (0 or 1), 0 if not using MultiAppBtldr
-flsLine <XXXX>	Specifies the number of bytes in a row of flash.
-arraySize <XXXXXX>	Specifies the flash array size.

Input:

The input hex files (program, protect, config) should be in Intel hex format. All input files should begin at address 0. The cyhextool program will automatically add an offset to the addresses to match the address map specified in the following table or the address map specified on the command line.

Name	Default Address (hex)	Default Size (hex)
CUNVLAT	000080	4
WONVLAT	0000F8	4
EEPROM	008000	None
CONFIG	080000	None
PROTECT	0C0000	None
PROGRAM	100000	None
CHECKSUM	200000	N/A

The -a option controls the address map. The argument of the -a option consists of a comma separated list of NAME=VALUE pairs. Each name corresponds to a section of the output file. The value is the address of the beginning of the section in hexadecimal. Optionally, VALUE may be ADDRESS:SIZE where ADDRESS is the hexadecimal address and SIZE is the hexadecimal size of the section. If the size is specified, cyhextool will produce an error message if the input data for that section exceeds the section size.

Output Format:

The output format is expected to be compatible with the format specified under [Input](#). The first line of the output file contains a header for the programmer. The PROGRAM, PROTECT, CONFIG (ECC), and EEPROM data will be aligned to 64-byte boundaries with zero-padding so that each line file will correspond to a Flash row. The checksum is calculated using the following steps:

1. Find the mod-65536 (16-bit) sum of all of the data bytes in the PROGRAM section.
2. Find the mod-256 sum the MSB and LSB of the result from step 1 and the constant 2.
3. The checksum is the two's complement of the result from step 2. The checksum will be between 0x00 and 0x100, so it is represented with two bytes.

Toolchain Support:

The makefile runs the cyhextool program after the linker finishes. If the linker does not produce an Intel hex file directly, the makefile generator will add commands to convert the output file to Intel hex format.

Keil

Keil uses the OMF or OMF2 file format. Keil's OH51 or OHX51 tool may be used to convert OMF/OM2 files to Intel hex format:

```
OH51 program.omf
```

CyElfTool Command Line Tool

The CyElfTool is applicable to projects targeting PSoC 4 and PSoC 5LP devices, using cy_boot v3.5 or later. For PSoC 3 projects and older PSoC 5LP projects, use the [CyHexTool](#) instead. For PSoC 6 projects, use the cymcuelftool provided with the PDL instead.

The CyElfTool tool is used to patch *.elf files. This is critical as many/most 3rd party tools use the *.elf file for programming devices and thus it needs to have all important information contained in it.

Command Line Arguments:

The following arguments are all mutually exclusive and can only be used one at a time:

Argument	Description
-h	Display help information
-V	Display version information
-C <file.elf>	Insert standard flash checksum information at 0x90300000
-S <file.elf>	Report the size of the application.
-P <file.elf> --flash_row_size <bytes> --flash_size <bytes> --size_var_name <name> --checksum_var_name <name> [--ignore_offset <bytes>]	Same as -C option, plus insert bootloader size, bootloader, checksum. Optional --ignore_offset option controls whether to ignore the first <bytes> bytes of flash in bootloader checksum.
-E <file.elf> --flash_row_size <bytes> --flash_size <bytes>	Creates a *.c file with the bootloader's Flash, Flash Protection, Customer NVL, Write Once NVL, EEPROM, Chip Protect, Meta, and Bootloader Meta. It also outputs the NVL section on the console out.
-B <file.elf> --flash_row_size <bytes> --flash_size <bytes> --flash_array_size <bytes> [-ee_array <arrayNum>]	Same as -C option, plus patch bootloadable meta-data with Application Checksum, Application Entry Address, Last Bootloader Row, Application Length, and Bootloader build version. Also generates the bootloadable's *.cyacd file.
-M <inputFile1.elf> <inputFile2.elf> <outputFile.elf> --flash_row_size <bytes> --flash_size <bytes>	Same as -C option, plus merge inputFile1 & inputFile2 into a single file, output as outputFile. Excluding the flash data and bootloadable metadata, inputFile1 & inputFile2 must have the same sections with matching content.

The cyelftool inserts or updates the following sections of the post-link .elf file:

- .cychecksum - All devices, contains the checksum of the flash portion of the application
- .cyloadermeta - All devices. The tool will update the section with the bootloader checksum and size information for a bootloader project.
- .cymeta - All devices. The checksum field will already have the silicon ID so that the cyelftool only needs to read the value, update it with the checksum, and write it back to the .elf file
- .cyloadablemeta (.cyloadable1meta/.cyloadable2meta) - All devices. The linker flow will populate all but the Application Checksum, Application Entry Address, and Application Length in these sections. The tool will need to compute these missing items and insert them into the post-link .elf file.

Keil Compiler

PSoC Creator includes the Keil compiler. It is a fully functional C compiler that is limited to level 5 optimization. If you need better optimization, you can upgrade by contacting Keil.

This Keil compiler will work as is for 30 days, at which time it becomes "Code Size Limited." This means it cannot link a program larger than 2k. To resolve the code size limited issue, you must register the compiler. Registration is free. It only requires that you complete an online form.

To Register the Compiler:

1. Click on the PSoC Creator **Help** menu and select **Register > Keil...**

This brings up a GUI that displays information on your various Keil installations.

2. In this GUI, click **Get license online** button.

This will bring you to a web page to register your compiler. This web page consists of a set of fields in which you are expected to enter pertinent data on your Keil installation. All fields with bolded titles are required.

- The first field is automatically filled in by PSoC Creator.
 - The second field should also be automatically completed. It should contain the code: IKA1P-M6Q0E-8W7ST.
3. Enter all other required values as necessary then press the **Submit** button.
 4. You will be sent a value via email to enter in the Keil Registration dialog. Bring the dialog back up and paste the new LIC value into the **New License ID Code (LIC)** field and click the **Add license** button.
 5. Your license will be added and you will have a fully registered version of the Keil compiler.

If you have any problems with the above registration process, try using the Keil µVision application. Open the registration dialog by selecting **License Management** under the **File** menu.

See Also:

- [Reentrant Code in PSoC 3](#)
- [Keil documentation](#)

Reentrant Code in PSoC 3

Due to the limited amount of stack and RAM space available, and for performance reasons, functions compiled with the Keil compiler are not reentrant by default. This means that the same function cannot be called multiple times concurrently in most cases, depending on the number and types of arguments and the usage of local variables. Concurrent function calls typically occur when the same function is called from two different interrupts, or from one interrupt along with being called from the main program execution. While not reentrant by default, functions can be made to support reentrancy.

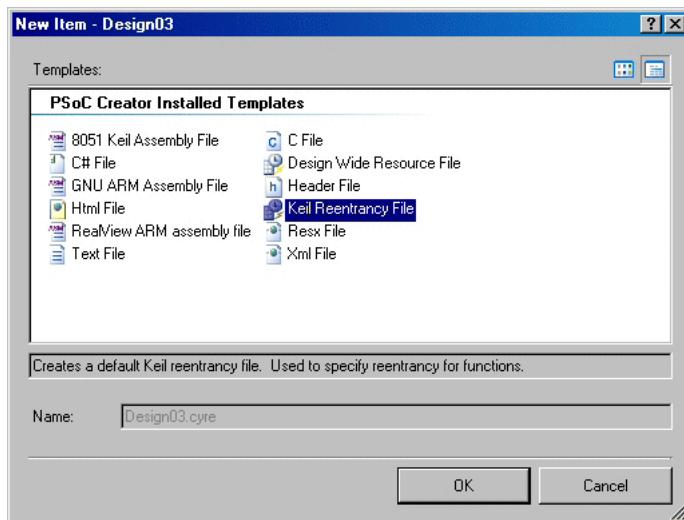
To Make Generated API Functions Reentrant:

PSoC Creator generates functions for the Components in your system. PSoC Creator allows you to specify which functions should be made reentrant on a case-by-case basis via a "reentrancy file" (*.cyre). This file specifies exactly which functions should be made reentrant. Each line of the file must be a single function name. During the build process, any candidate functions contained in the reentrancy file will automatically be marked to support reentrancy.

Most functions in the API files are candidates for reentrancy. Comments in the Component API source files will indicate which functions are not candidates.

*To Add a *.cyre file to a project:*

1. Right-click on a project in the Workspace Explorer, and select **Add > New Item**.
2. On the New Item dialog, select the "Keil Reentrancy File" and click **OK**.



The *.cyre file opens in the code editor. The file is named based on the project and cannot be renamed without renaming the project (similar to the DWR file).

To enter reentrant functions:

1. Type a single function name per line, for example:

```
ADC_Start
PWM_Start
```

2. Save the *.cyre file when complete.

To Make User Application Code Reentrant:

To support reentrancy in your own application code, specify the “CYREENTRANT” #define from *cytypes.h* as part of the function prototype and also in the function definition. For the Keil tool chain, this will evaluate to the “reentrant” keyword, while for all other tool chains it will evaluate to nothing. This allows the code to function properly across multiple tool chains and multiple device architectures.

Original function:

```
void Foo(void);
```

Modified function:

```
void Foo(void) CYREENTRANT;
```

To Make Custom Component APIs Support Reentrancy:

When creating a custom Component that might need to support reentrancy, the function can be declared using the “ReentrantKeil” build expression. Both the function declaration in the .h file and the definition in the .c file must include this expression. This will allow it to behave the same as other Components shipped with PSoC Creator. By default, it will be a standard function; however, if you choose to add that function name to the reentrancy file, it will get marked as reentrant.

Original function:

```
void `$INSTANCE_NAME`_Foo(void);
```

Modified function:

```
void `$INSTANCE_NAME`_Foo(void) `=ReentrantKeil($INSTANCE_NAME . "_Foo")`;
```

To Determine What to Make Reentrant:

The Keil compiler can help determine which functions should be marked reentrant during the build process. When the optimization level is set to 2 or higher and a build is performed, Keil will output a warning for any functions that are called simultaneously that are not marked as reentrant.

You only want to mark a function as reentrant when the Keil compiler allocates RAM space for the function in addition to being called concurrently. This is based on the number and type of arguments to the function, the usage of local variables, and the complexity of the calculations in the function. The Keil compiler warning messages should be used to determine what functions need to be marked as reentrant. The following is example output from the Keil linker:

```
Warning: L15 MULTIPLE CALL TO FUNCTION NAME: _MYFUNC/MAIN CALLER1: ?C_C51STARTUP  
CALLER2: ISR_1_INTERRUPT/ISR_1
```

In this case, the function `MyFunc`, which is in the file `main.c`, is being called from two different concurrent execution flows. The first caller, denoted `?C_C51STARTUP`, is the main flow of execution that originates from the `main()` function. The second caller is the `ISR_1` interrupt that is in the `isr_1.c` file.

See Also:

- [Keil Compiler](#)

6 Integrating into 3rd Party IDEs



PSoC Creator provides different methods for integrating a PSoC Creator design and various firmware files in a 3rd party IDE. These methods vary depending on the selected device in your PSoC Creator design. There are also differences in the process depending on the 3rd party IDE you wish to use. The following table summarizes the methods and options for 3rd party IDE integration.

Cypress Device	3rd Party IDE Available	Integration Method to Use
PSoC 6	CMSIS Pack (Eclipse and Arm MDK) IAR Makefile	Generating PSoC 6 Files for 3rd Party IDEs
PSoC 4 and PSoC 5LP	Eclipse IAR μVision CMSIS Pack Makefile	Select the desired IDE on the IDE Export Wizard .
PSoC 3	μVision Only	Exporting a PSoC 3 Design to Keil μVision IDE .
FM0+	Makefile Only	Exporting a Design to Makefile .

Once you've generated files to use with the desired 3rd Party IDE, refer to the following table for the appropriate path to use for the selected Cypress device and target 3rd party IDE:

Cypress Device	Selected 3rd Party IDE	Integration Path to Follow
PSoC 6	Eclipse	Using PSoC 6 Designs in Eclipse
	IAR	Setting up a PSoC 6 IAR Project
	Arm MDK (μVision)	Creating μVision Projects for PSoC 6
	Makefile	Building PSoC 6 Designs with Make
PSoC 4 and PSoC 5LP	Eclipse	Import into Eclipse
	IAR	Setting up a PSoC 4/PSoC 5LP IAR Project
	μVision	Exporting a PSoC 4/PSoC 5LP Design to Keil μVision IDE
	CMSIS Pack	Opening Generated CMSIS-Pack Projects (μVision 5 IDE)
	Makefile	Opening PSoC Creator Designs in Makefile
PSoC 3	μVision Only	Opening Projects in μVision IDE
FM0+	Makefile Only	Opening FM0+ Designs in Makefile

PSoC 6 Designs

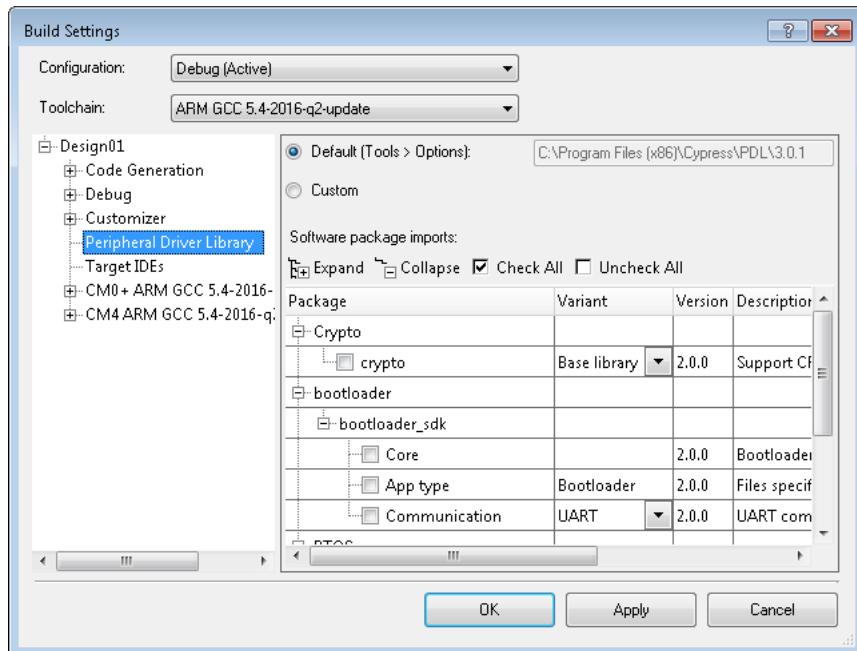
For all PSoC 6 Designs, use the Build Settings > [Target IDEs](#) page to select one or more IDEs for which PSoC Creator will generate files. This section contains the following topics for generating the files and working with the various 3rd Party IDEs:

- [Generating PSoC 6 Files for 3rd Party IDEs](#)
- [Using PSoC 6 Designs in Eclipse](#)
- [Setting up a PSoC 6 IAR Project](#)
- [Creating µVision Projects for PSoC 6](#)
- [Building PSoC 6 Designs with Make](#)

Generating PSoC 6 Files for 3rd Party IDEs

The process to generate PSoC 6 design files for use in 3rd party IDEs is similar for all the IDEs available, as follows:

1. Develop your PSoC 6-based PSoC Creator design in the usual manner as for any other device. That is, add Components to the schematic, write firmware, program and debug as you normally would.
- Note** If you want to program the device using JTAG in your desired 3rd party IDE, you must set the Select Debug option to JTAG in [PSoC Creator System Editor](#).
2. As part of that process, open the [Build Settings dialog](#). Right-click on a project, and select **Build Settings...**
3. Select the "Peripheral Driver Library" page.

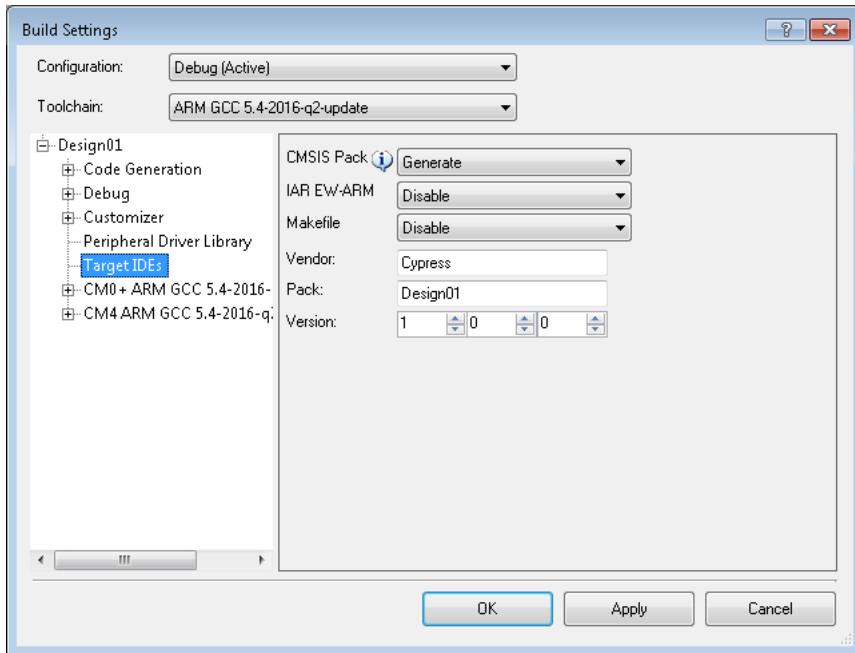


- Verify that PDL 3.0.1 is installed.

- If needed, select **Custom** and navigate to the location of an alternate PDL installation.
- Under **Software package imports**, select software packages (Middleware, RTOSes, etc.) to include in your build.

For more information about this page, refer to the [Build Settings > Peripheral Driver Library](#)

4. Select the "Target IDEs" page.



Select one or more IDEs for which to generate files from the pull-down menus. Options include CMSIS Pack (Eclipse and Arm MDK), IAR, and Makefile.

- For all IDEs, the options include **Generate** or **Disable**.
- For IAR, there is also **Generate without copying PDL files**. This option will generate a project connection file (.ipcf file) that does not include PDL files from the PSoC Creator design. You may use a separate .ipcf file to include compatible PDL files in the IAR project.

For more information about these options and what types of files are generated, see the [Build Settings > Target IDEs](#) page.

Note The Target IDEs page displays as part of [Creating a New Project](#) for PSoC 6 devices only.

5. Click **OK** to close the Build Settings dialog. Then build your PSoC 6-based PSoC Creator design in the usual manner.

Refer to the following sections for instructions about integrating the generated files into the desired 3rd party IDE.

- [Using PSoC 6 Designs in Eclipse](#)
- [Setting up a PSoC 6 IAR Project](#)
- [Creating µVision Projects for PSoC 6 \(CMSIS-Pack\)](#)
- [Opening PSoC 6 Designs in Makefile](#)

Using PSoC 6 Designs in Eclipse

This section covers the process for using generated PSoC 6-based PSoC Creator design files in the Eclipse environment. If not already done, refer to [Generating PSoC 6 Files for 3rd Party IDEs](#) for the process to generate the necessary files.

This process involves several subsections, follows:

- [Eclipse Configuration](#)
- [Creating a New Design Project](#)
- [Initial Project Setup](#)
- [Multi-Project Build](#)
- [PSoC 6 Debug Flow Using Eclipse/J-Link](#)

Eclipse Configuration

You will need to perform the following Eclipse installation and configuration steps. These steps only need to be performed once per Eclipse installation on your machine.

Eclipse CDT

Install Eclipse Luna or later with the CDT (C/C++ Development Tools) features. From www.eclipse.org, the download ZIP archive will be named something like the following:

- eclipse-cpp-luna-SR2-win32.zip
- eclipse-cpp-mars-SR2-win32.zip
- eclipse-cpp-neon-1-win32.zip

Arm CMSIS Pack Management Plugins

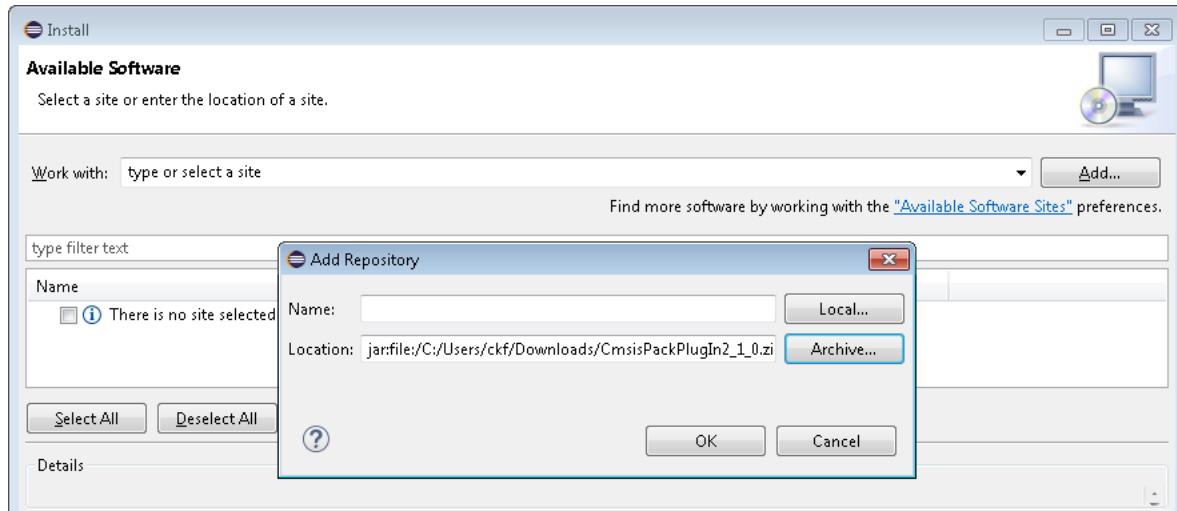
Also install the Arm CMSIS-Pack Management plugins. Version 2.0.1 or higher is required. The plugins can be downloaded as a ZIP archive from:

<https://github.com/arm-software/cmsis-pack-eclipse/releases>

Install the plugins for Eclipse as follows:

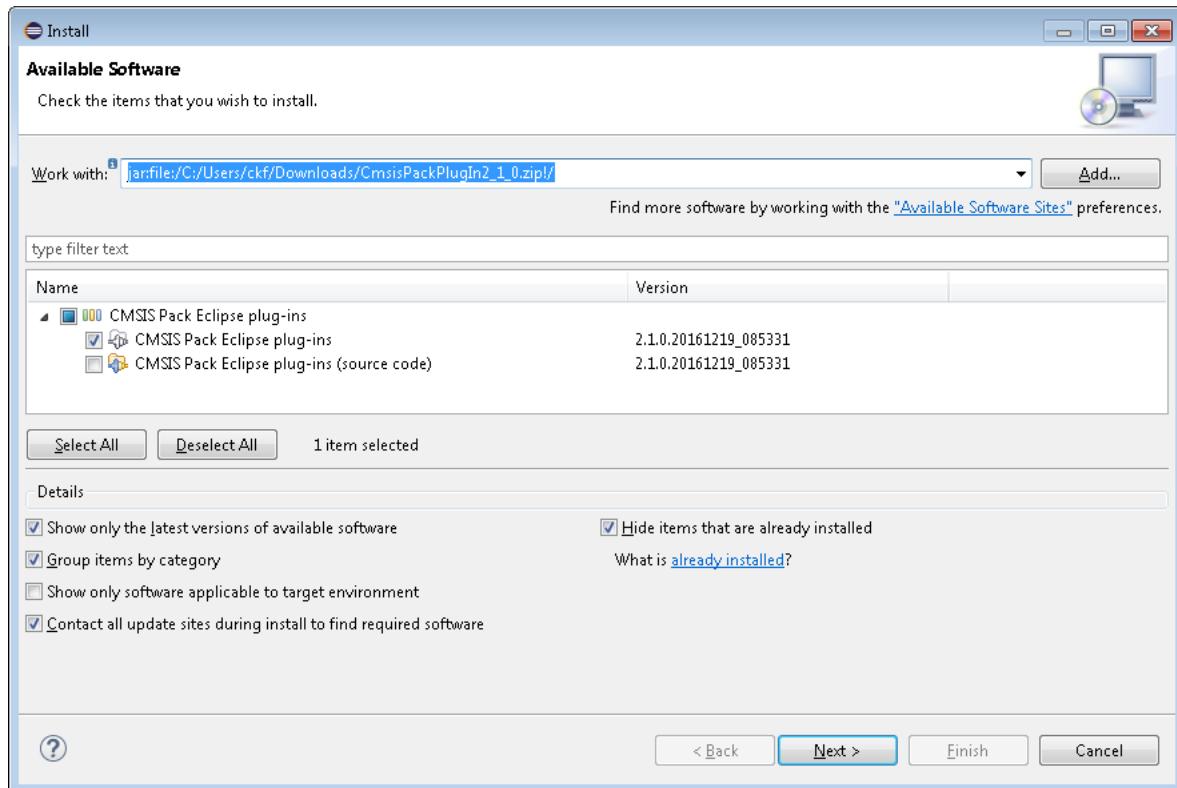
1. In Eclipse, go to the **Help** menu and select **Install New Software...**

2. On the Install dialog, click **Add...** Then on the Add Repository dialog, enter the path to your downloaded ZIP file as the **Location** value and click **OK**.



3. The Install dialog will populate as shown. Select the **CMSIS Pack Eclipse plugin-ins** check box and click **Next >**.

Note If you see a message "There are no categorized items" instead of the expected feature name in the following image, unselect the **Group items by category** check box.

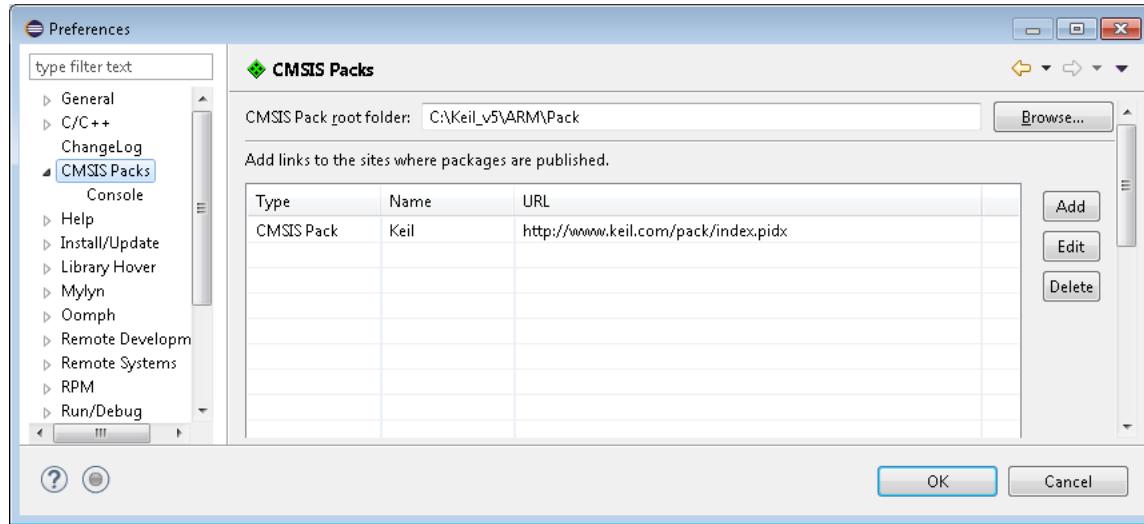


4. On the next page, accept the license agreement and click **Finish**. The plugins will be installed and you must restart Eclipse to complete the installation.

Eclipse CMSIS-Pack Folder

The CMSIS-Pack Management plugins require that a CMSIS-Pack root installation folder be set. Go to the Eclipse Window menu, select Preferences, and enter the path to the CMSIS-Pack root installation folder you wish to use.

For users who have Keil µVision 5 or later installed, the path will be the one shown below.



Cypress Toolchain Adapter Feature

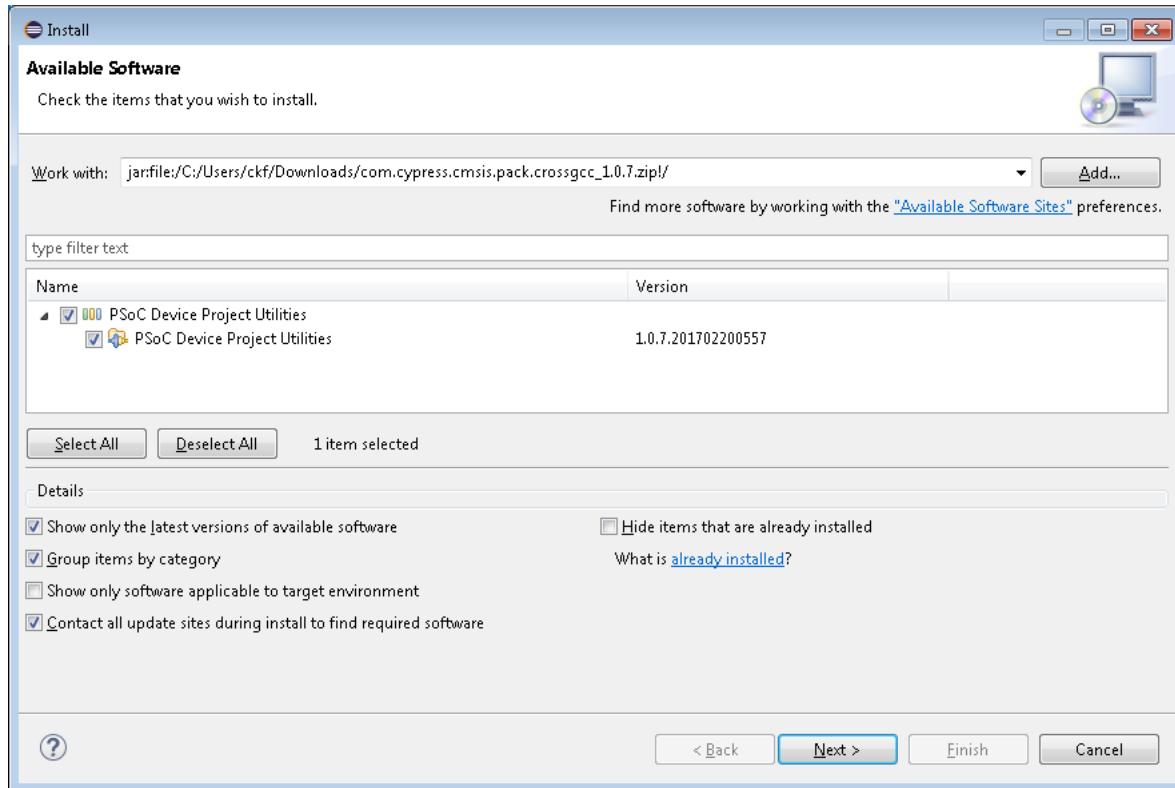
As a convenience to our PSoC Creator/Eclipse users, Cypress provides a small "toolchain adapter" feature, which populates several Eclipse project tool settings for projects imported from PSoC Creator, saving time and reducing the chances of incorrectly typed tool setting values. The feature can be downloaded as a ZIP archive from the PSoC Creator downloads page:

<http://www.cypress.com/products/psoc-creator-integrated-design-environment-ide>

To install this feature:

1. Download the ZIP archive file.
2. In Eclipse, go to the **Help** menu and select **Install New Software...**
3. On the Install dialog, click **Add...** Then on the Add Repository dialog, enter the path to your downloaded ZIP file as the **Location** value and click **OK**.

4. The Install dialog will populate as shown. Select the **PSoC Device Project Utilities** check box and click **Next >**.



5. On the next page, accept the license agreement and click **Finish**. The feature will be installed and you must restart Eclipse to complete the installation.

This feature provides the following entries in a project's toolchain options:

- Postbuild command, including a call to *cypdlefelftool.exe* to finish preparing a flash image file
- Assembler flags: `-mcpu=<core> -mthumb` options
- Compiler flags: same, plus `-I` include paths
- Linker: same, plus `-specs=nano.specs` and `-T <path to PDL GCC linker script>`
- Linker: adds `-Wl,--start-group` and `-Wl,--end-group` options around linker inputs

Creating a New Design Project

Installing the CMSIS Pack

The steps you performed in PSoC Creator created a CMSIS Pack file for your configured PSoC 6 device. You now need to install that CMSIS Pack in your CMSIS Pack root folder, where CMSIS compliant tools will locate it. This needs to be performed only once.

- If you have the Keil µVision 5 or later toolset installed, this is straightforward. In your file system explorer window, navigate to your PSoC Creator design folder, and then into the folder named Export and then into the folder named Pack. You will find a file with an extension of .pack, for example *Cypress.Design01.1.0.0.pack*. Double click on this file to launch the Keil PackUnzip tool to complete the installation step.

- If you do not have Keil µVision 5 or later installed, there are two scripts provided in your PSoC Creator installation that can be used to perform this same task: PackInstall.bat (for Windows only) and PackInstall.bash. These scripts are located in the same folder as your .pack file above. Their command lines are as follows:

```
PackInstall.bat <packFile> [<destFolder>] [/force]
- or -
PackInstall.bash <packFile> [<destFolder>] [-force]
```

- packFile: path to the .pack file generated by PSoC Creator
- destFolder: the folder where the .pack file contents are to be installed. If omitted, the current working directory is used.
- /force / -force: remove and overwrite any existing files in the destFolder [optional]

Note These scripts require that you have 'unzip' available via your Path environment variable. To temporarily change the PATH variable in a command line session, type the following:

```
PATH=%PATH%;<full_path_to_unzip_folder>
```

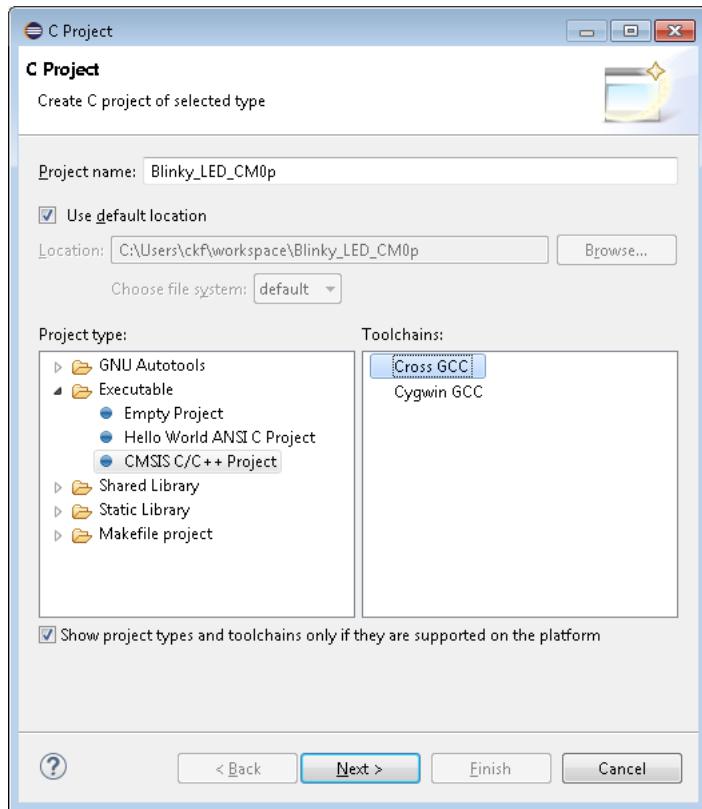
You can also permanently edit your Path environment variable and provide the full path to the "unzip" folder, usually C:\Program Files\unzip.

Creating the Eclipse Project

A new project type exists for CMSIS-Pack-based designs to speed user project creation. This helps to quickly generate new projects based on CMSIS-Pack firmware content. To create a new project:

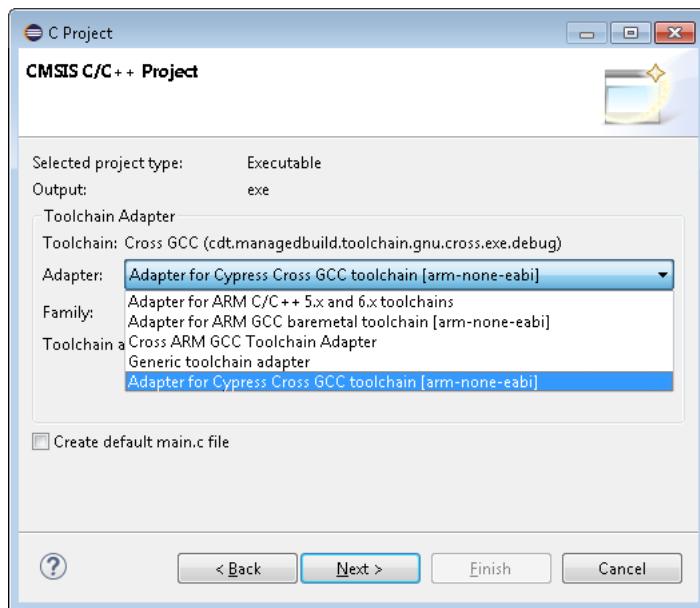
1. Create a project via the **File > New > C Project** menu entry.

2. On the C Project page:



- Enter a project name.
- Under Project type, select **CMSIS C/C++ Project**.
- Under Toolchains, select **Cross GCC**.
- Click **Next >**.

3. On the CMSIS C/C++ Project page, select **Adapter for Cypress Cross GCC toolchain** and click **Next >**.

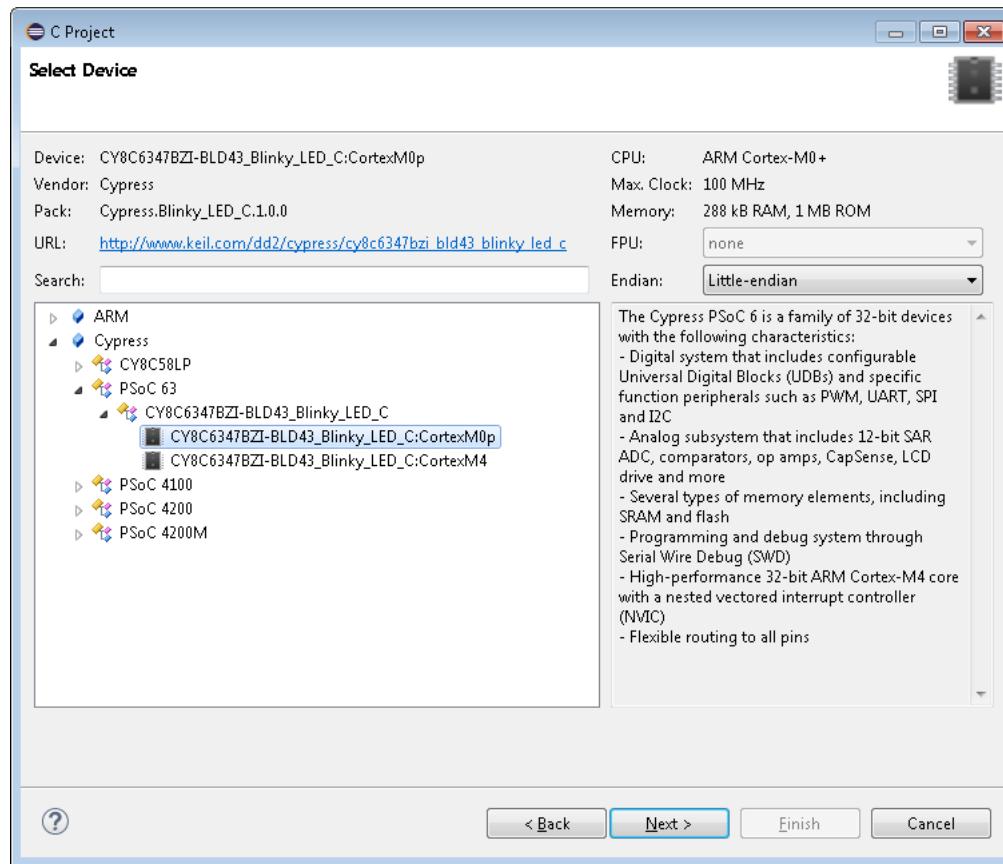


Note Do not select the **Create default main.c file** check box. See [Including Application Files in the Project](#).

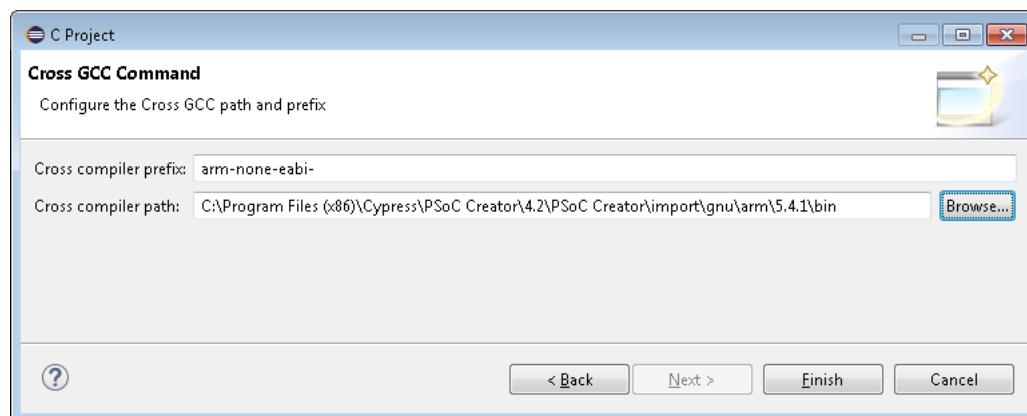
4. On the Select Device page, select the appropriate device and click **Next >**.

Note For PSoC 6 devices, both CM0+ and CM4 core entries will be shown. Eclipse projects can only be built for one processor core, so select the appropriate core for the project you are configuring, either CM0+ or CM4.

Note If you created a CMSIS Pack with a custom vendor name, it will still be listed in the Cypress section of the following dialog, as this list is organized by hardware device vendor.



5. On the next page, accept the configurations (Debug and Release) and click **Next >**.
6. On the Cross GCC Command page, enter the prefix `arm-none-eabi-` and the path to your Arm GCC toolchain. The Arm toolchain shipped with PSoC Creator releases can be used.



Note You will only need to enter these values for the first CMSIS-Pack project you create. These values will auto-populate in the dialog fields for subsequently created projects.

7. Click **Finish** and the project will be created.

Initial Project Setup

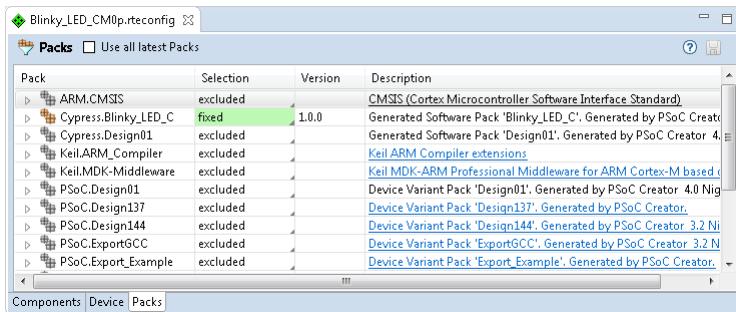
There are several initial steps that must be taken before this newly created project will build. These steps only need to be performed once per newly created project.

Note The New Project wizard toolchain selection page (see [Creating a New Design Project](#)) includes a check box for creating a default main.c file. Unfortunately, this creates a default for Arm tools, which is not appropriate for PSoC device development.

Importing PDL Firmware

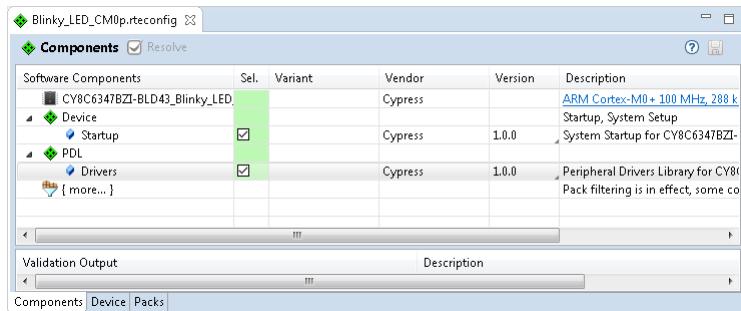
To import the device's firmware into the project, go to the RTE Config file pane, under **Packs**:

- Unselect the **Use all latest Packs** check box.
- Under the **Selection** column, select "fixed" from the drop down menu for the CMSIS Pack that PSoC Creator created for your configured PSoC 6 device. All other unused packs should be excluded.



Then under **Components**, check the following boxes under the **Sel.** column:

- Device > Startup
- PDL > Drivers



This file editor pane was opened when you first created the project. Make sure you save this file; the selected firmware will be included in your Eclipse project source and be listed in the Project Explorer pane.

Including Application Files in the Project

PSoC 6 projects typically require the following files:

- main*.c file
- cyapicallbacks.h file
- cy_ipc_config.c and cy_ipc_config.h files

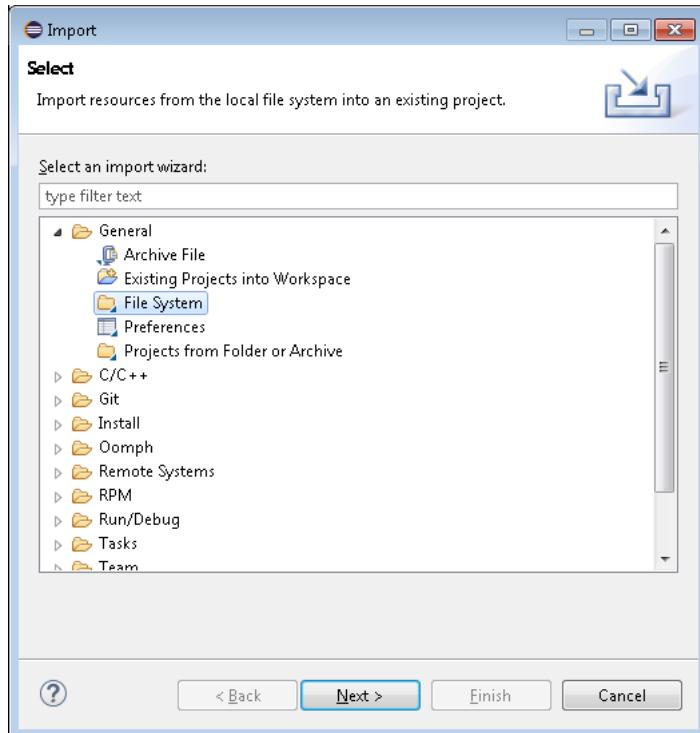
- linker script file

You may need different files depending on your configuration. You can include the original copies of these files from your PSoC Creator project or create new ones from template files.

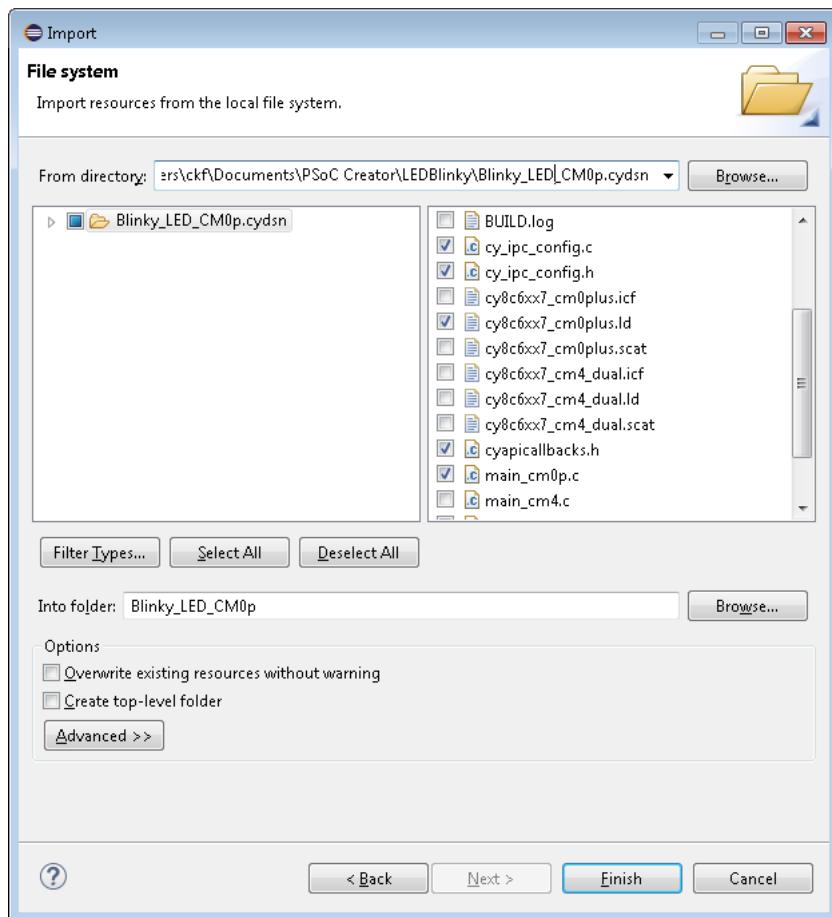
Include Original Files

To include the original copies of these files from your PSoC Creator project:

1. Right-click on the project and select **Import**.
2. On the Import dialog Select step, expand **General** and select **File System**; then click **Next >**.



3. On the Import dialog File system step, navigate to the <project>.cydsn folder, select the files, and click **Finish**.

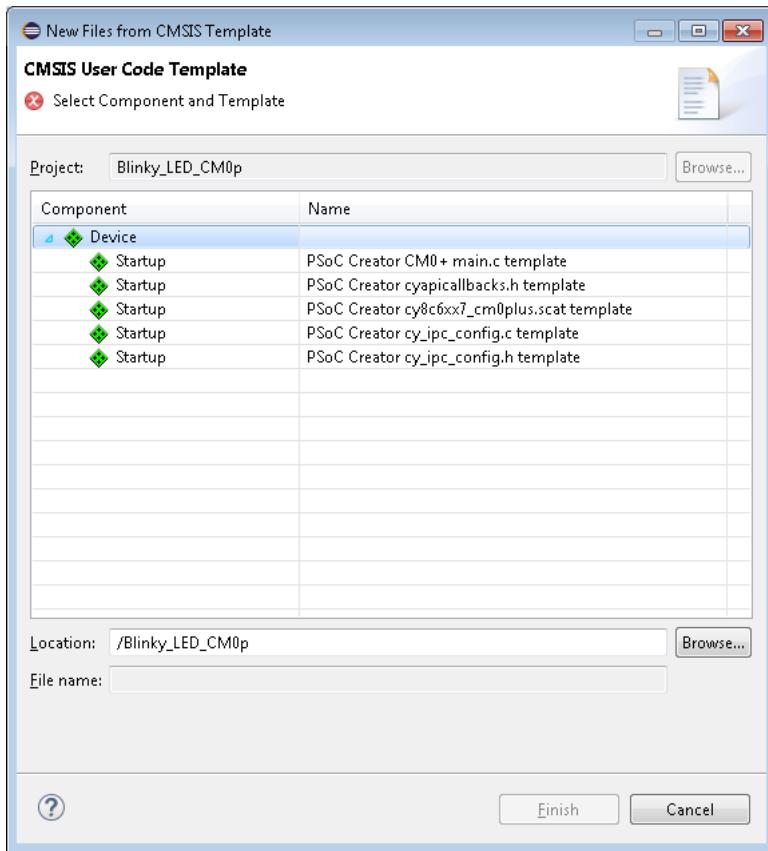


Create new Files

Template files for your project have been added to the toolchain adapter plugin. These files can be added to your project as follows:

1. Select the project in the Project Explorer.

2. Right click and select **New > Files from CMSIS Template** to open the New Files from CMSIS Template dialog.



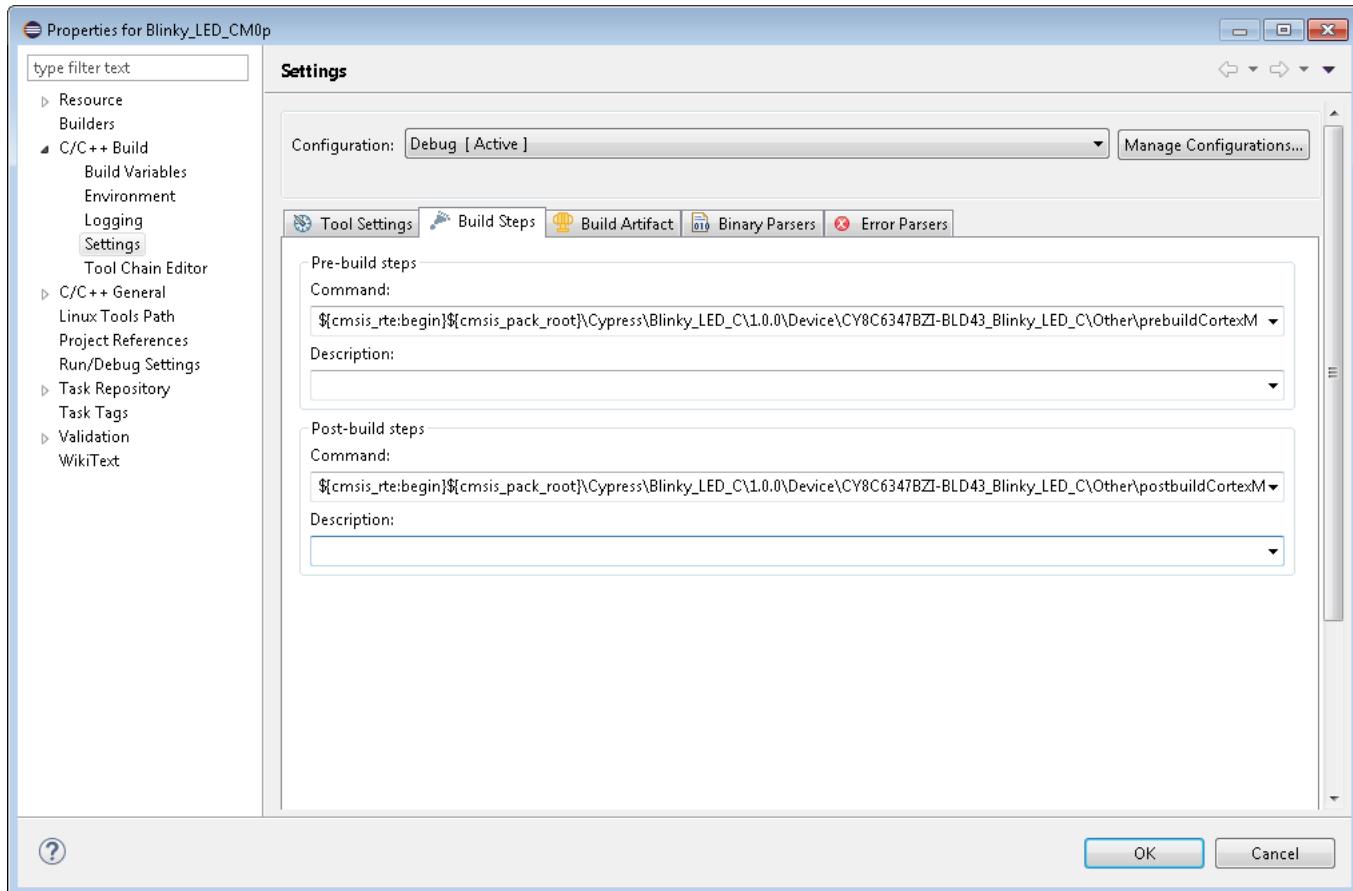
3. Select the file(s) to include in your project and click **Finish**.

In order for your builds to pick up the newly added files, add a new include path entry as follows:

1. Select the project in the Project Explorer.
2. Right-click on the project and select **Properties** to open the Properties dialog.
3. Under C/C++ Build, select Settings.
4. On the Tool Settings tab, under Cross GCC Compiler, select Includes.

User Commands:

PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. For Eclipse, you can edit your project's settings (under the project's **Properties > C/C++ Build > Settings**) on the Build Steps tab. PSoC Creator provides a call to our generated pre-build and post-build scripts where needed; you can add your own commands, separating all commands with semicolons.



Note If you see errors like the following while building a design containing a CapSense or other Components making use of floating point hardware, you must enable the use of floating point hardware.

Error:

```
Caps_s.elf uses VFP register arguments
```

To fix this, add the following to your Assembler/Compiler/Linker toolchain options:

```
-mfloat-abi=softfp
```

Note If you are running Eclipse on linux or macOS, change the extension of the pre-build and post-build scripts from .bat to .sh to use the shell scripts provided by PSoC Creator.

Multi-Project Build

While PSoC 6 projects for devices with two cores are represented as a single project in PSoC Creator 4.2, other third-party IDEs including Eclipse require one project per processor core. For a PSoC 6 device with two cores, Eclipse requires that you create two projects, one for CM4 code and one for CM0+ code.

You must complete a dependency from your CM4 to the CM0+ project as follows:

In your CM4 project, you must right-click on the project name in the Project Explorer pane and select Properties. From there, select C/C++ Build and then Settings. On the Build Steps tab, the postbuild command will have an argument that resembles:

```
"${workspace_loc}\OTHER_PROJ\${ConfigName}\OTHER_PROJ"
```

The values OTHER_PROJ above need to be replaced with the name of your CM0+ project.

In order to build the full design, you should first build your CM0+ project and then your CM4 project. If you have modified the CM0+ project but not the CM4 project, you should perform a clean and then a build on the CM4 project to ensure that the combined image is correctly regenerated.

PSoC 6 Debug Flow Using Eclipse/J-Link

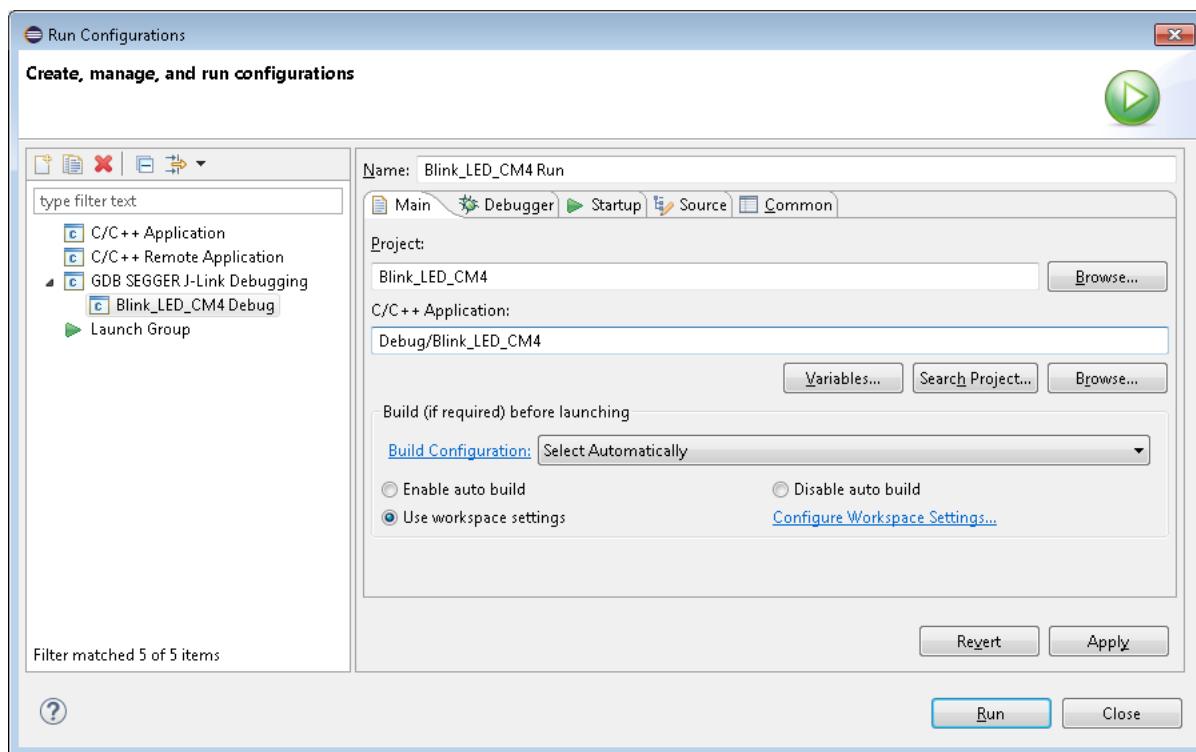
Install Required Software

This process was tested using the Windows OS only. Install the following additional software, as needed:

- J-Link Software and Documentation Pack V6.30 (<https://www.segger.com/downloads/jlink>)
- GNU Arm Eclipse (<https://gnuarmeclipse.github.io/plugins/install/>), in particular the GNU Arm C/C++ J-Link Debugging plugin

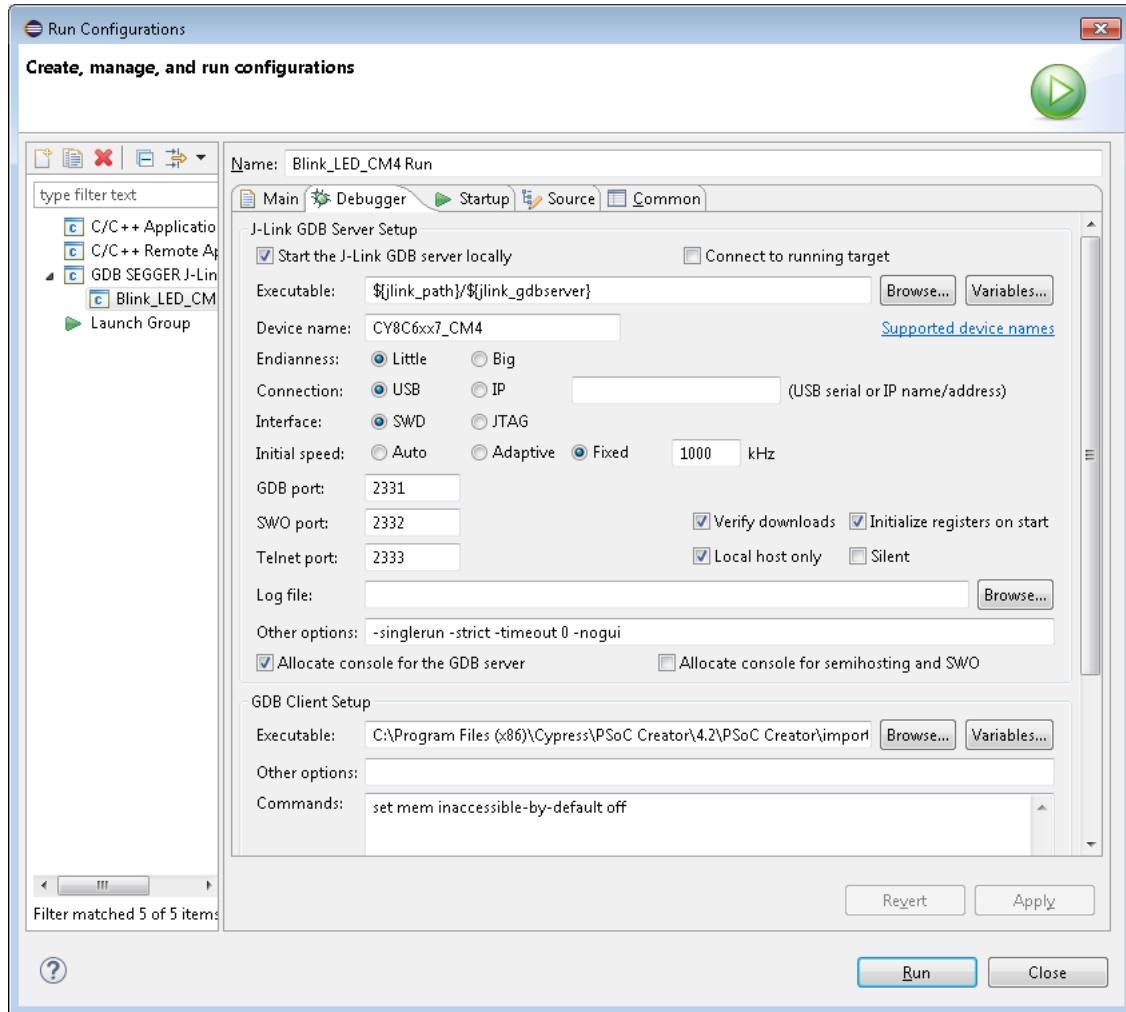
Program Flow

1. Create a new configuration by using 'Create, manage, and run configurations' dialog (**Run > Run Configurations...**). Create a new GDB SEGGER J-Link Debugging configuration.
2. Select the Run Configurations window.
3. Select the **Main** tab:
 - Set a name for the configuration in the **Name** field.
 - In the **C/C++ Application** field, browse to the project file generated by Eclipse in Debug or Release folder under your Eclipse project directory.
 - Do not select the *PROJECTNAME_link.out* file.
 - Click **Apply**.



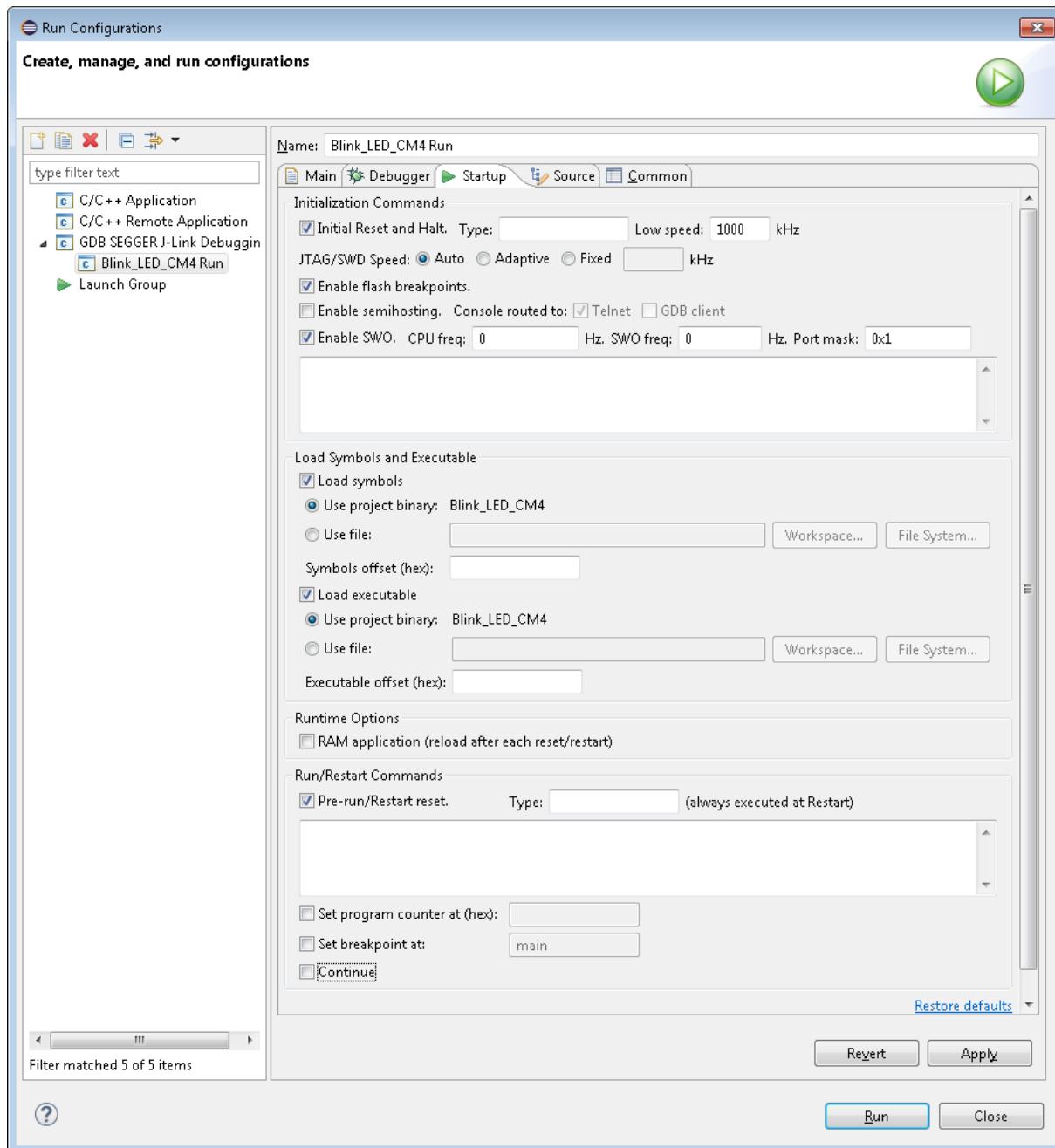
4. Select the **Debugger** tab:

- Unselect the **Connect to Running Target** check box.
- Set the **Device Name** based on the core.
- Unselect the Allocate console for semihosting and SWO check box.
- Under **GDB Client Setup** set the **Executable** path to "<PSoC Creator Install Directory>\import\gnu\arm\5.4.1\bin\arm-none-eabi-gdb.exe".
- Click **Apply**.



5. Select the **Startup** tab:

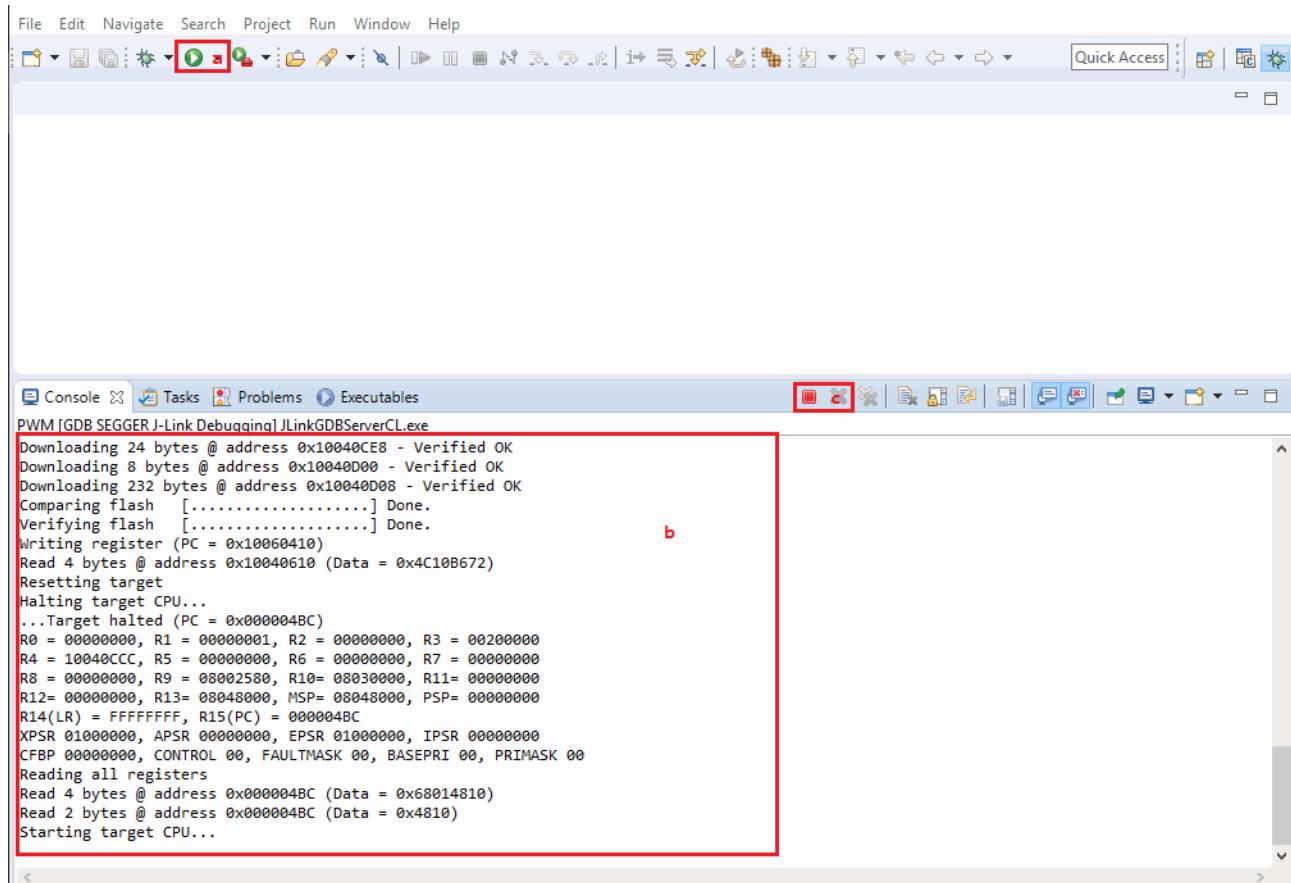
- Unselect the **Enable semihosting** check box.
- Unselect the **Set breakpoint at** check box.
- Unselect the **Continue** check box.
- Click **Apply**.



6. Close the Run Configurations dialog.

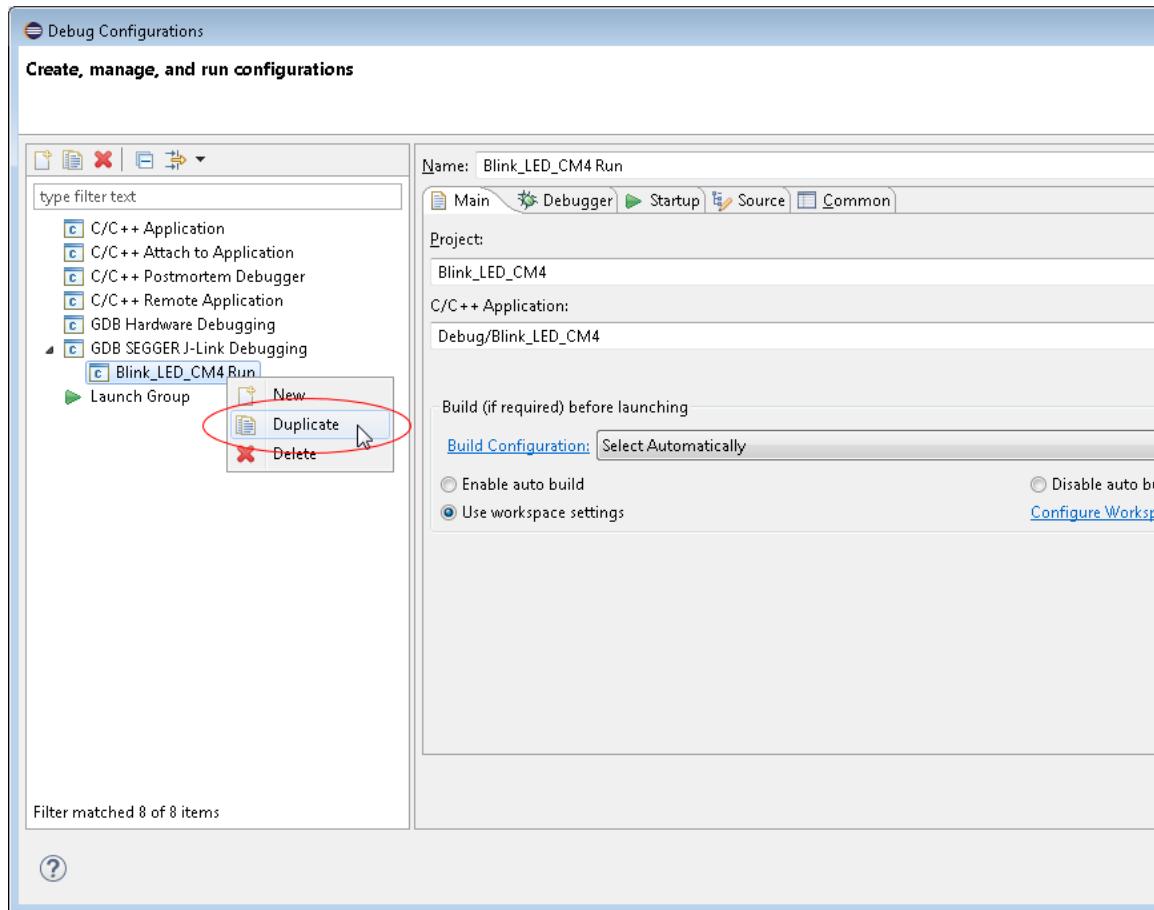
7. On the main Eclipse window:

- Click the **Run** button.
- When run is completed, check if "Downloading" and "Verifying" operations complete with OK status.
- Click the **Stop** button.

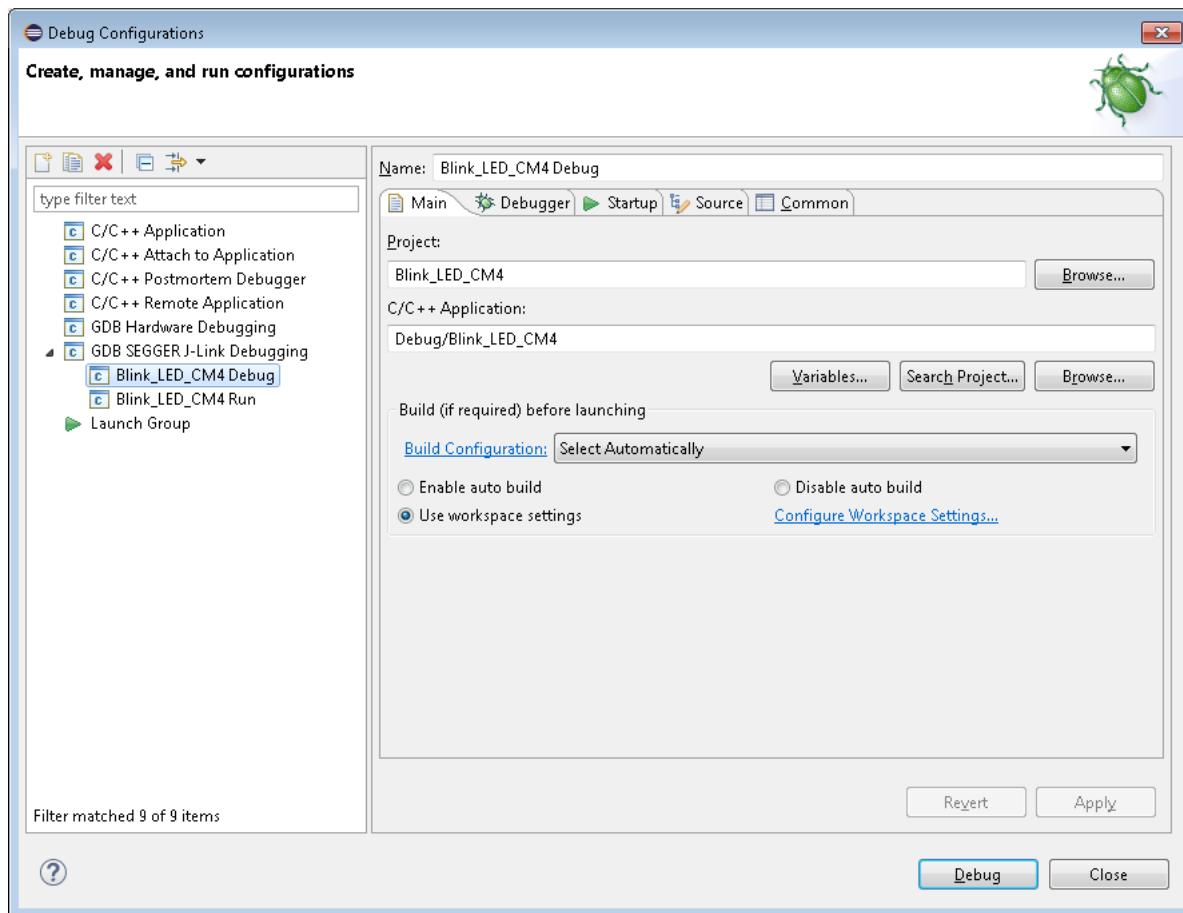


Debug Flow

1. Use a workspace that was created for [Program Flow](#).
2. Create a new configuration by using the Create, manage, and run configurations dialog (**Run > Debug Configurations...**) and **Duplicate** options.

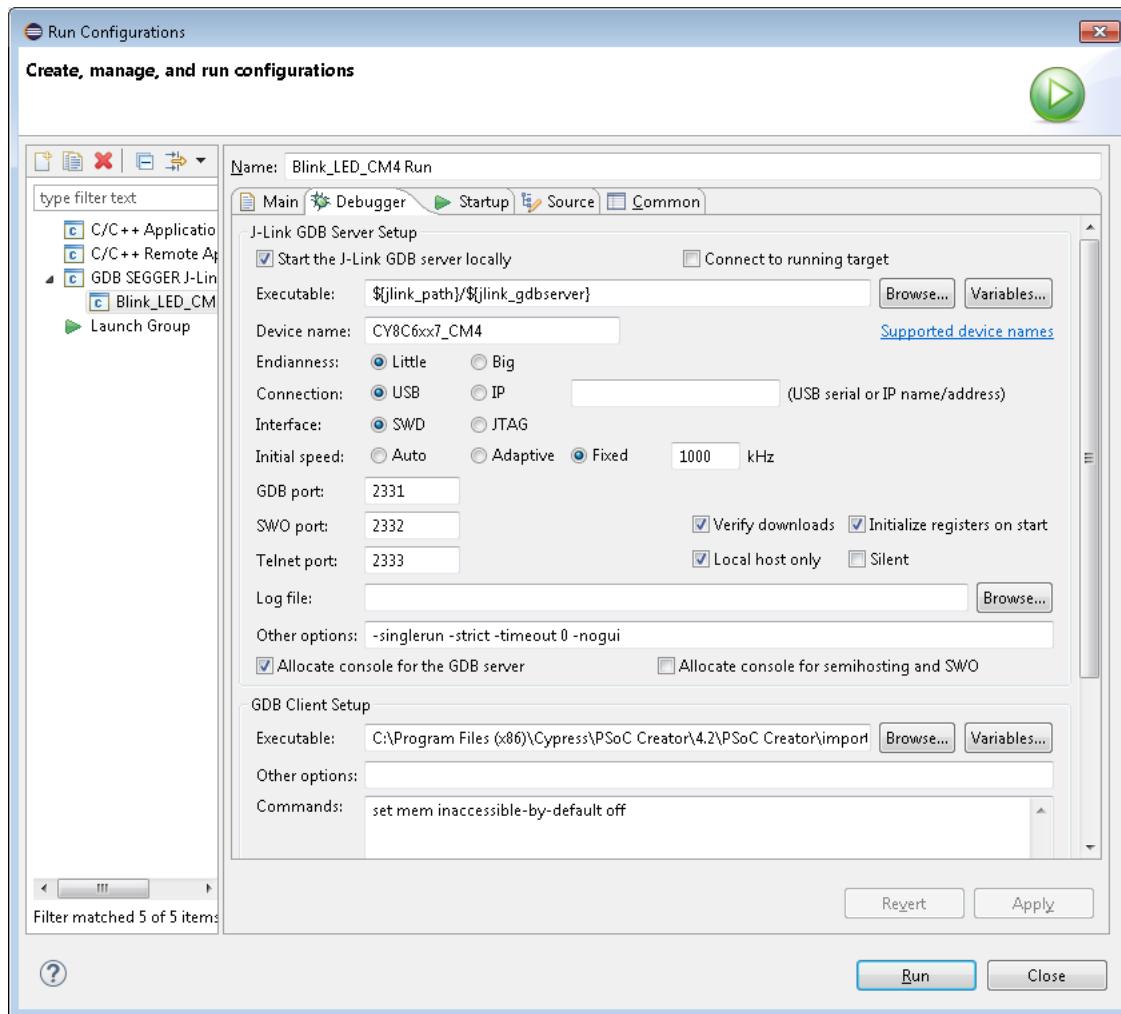


3. Select Debug Configurations window.
4. Select the **Main** tab:
 - Set a name for the configuration in the **Name** field.
 - Click **Apply**.



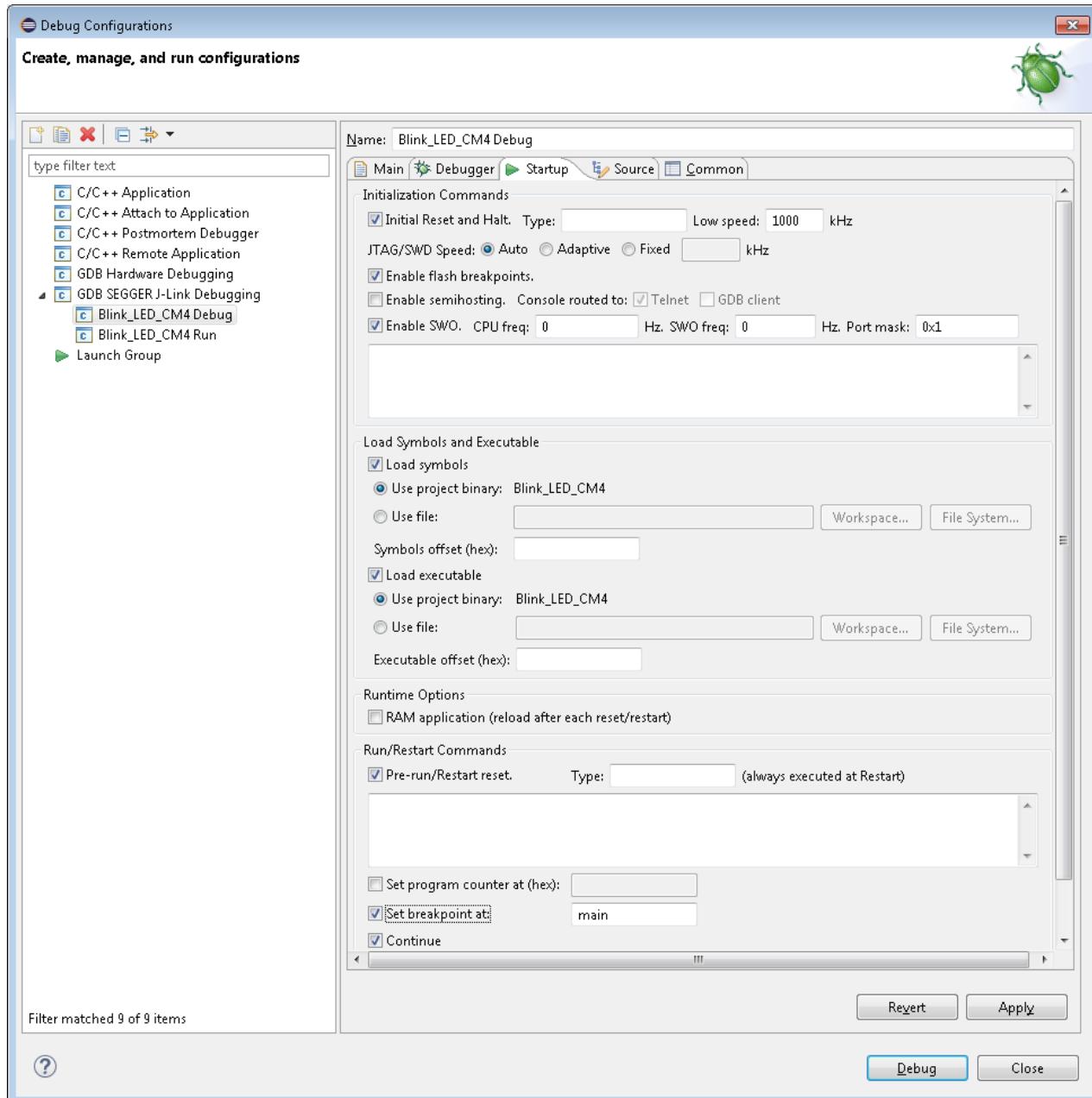
5. Select the **Debugger** tab:

- Unselect the **Connect to Running Target** check box.
- Set the **Device Name** per core.
- Unselect the Allocate console for semihosting and SWO check box.
- Click **Apply**.



6. Select the **Startup** tab:

- Unselect the **Enable semihosting** check box.
- Select the **Set breakpoint at main** check box.
- Select the **Continue** check box.
- Click **Apply**.

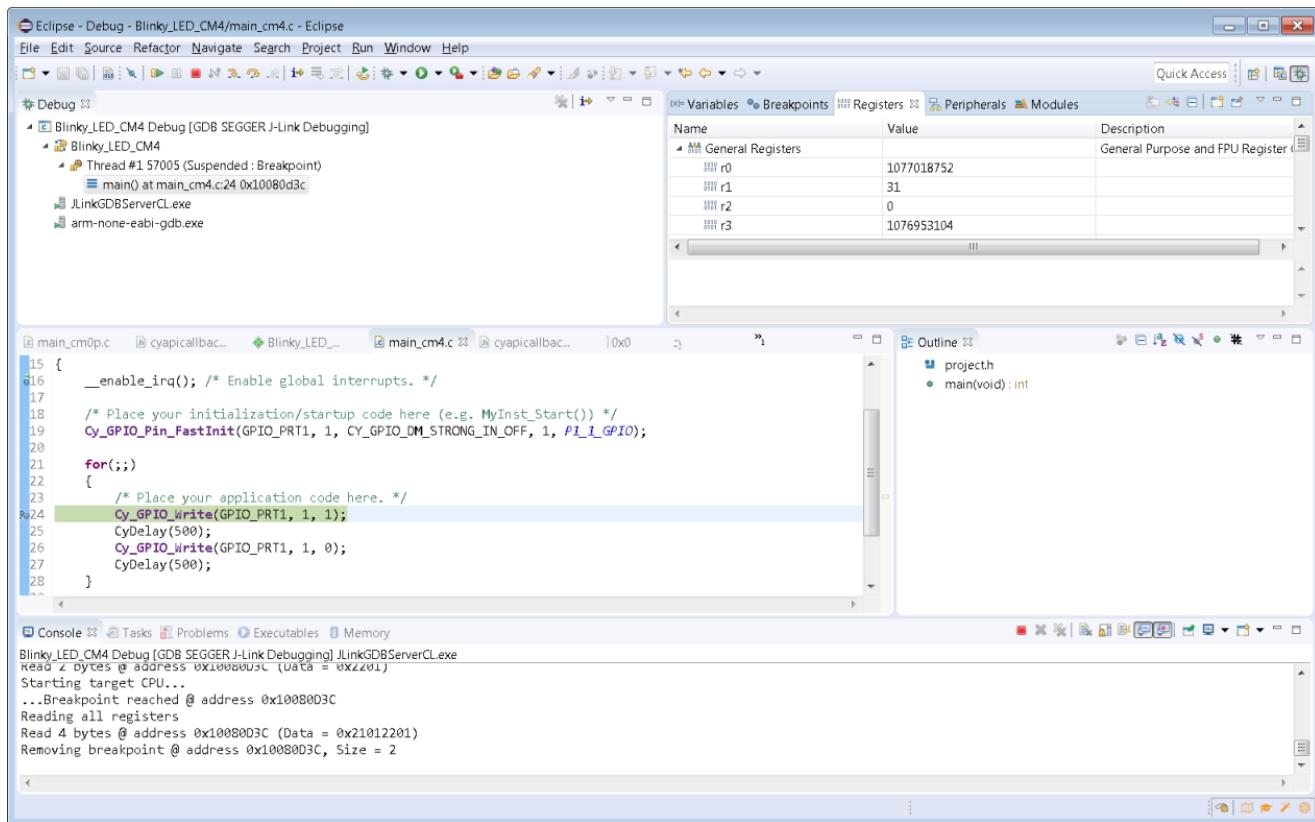


7. Close the Debug Configurations dialog.

8. On the main Eclipse window:

- Click on triangle in the **Debug** button.

□ Select debug configuration.

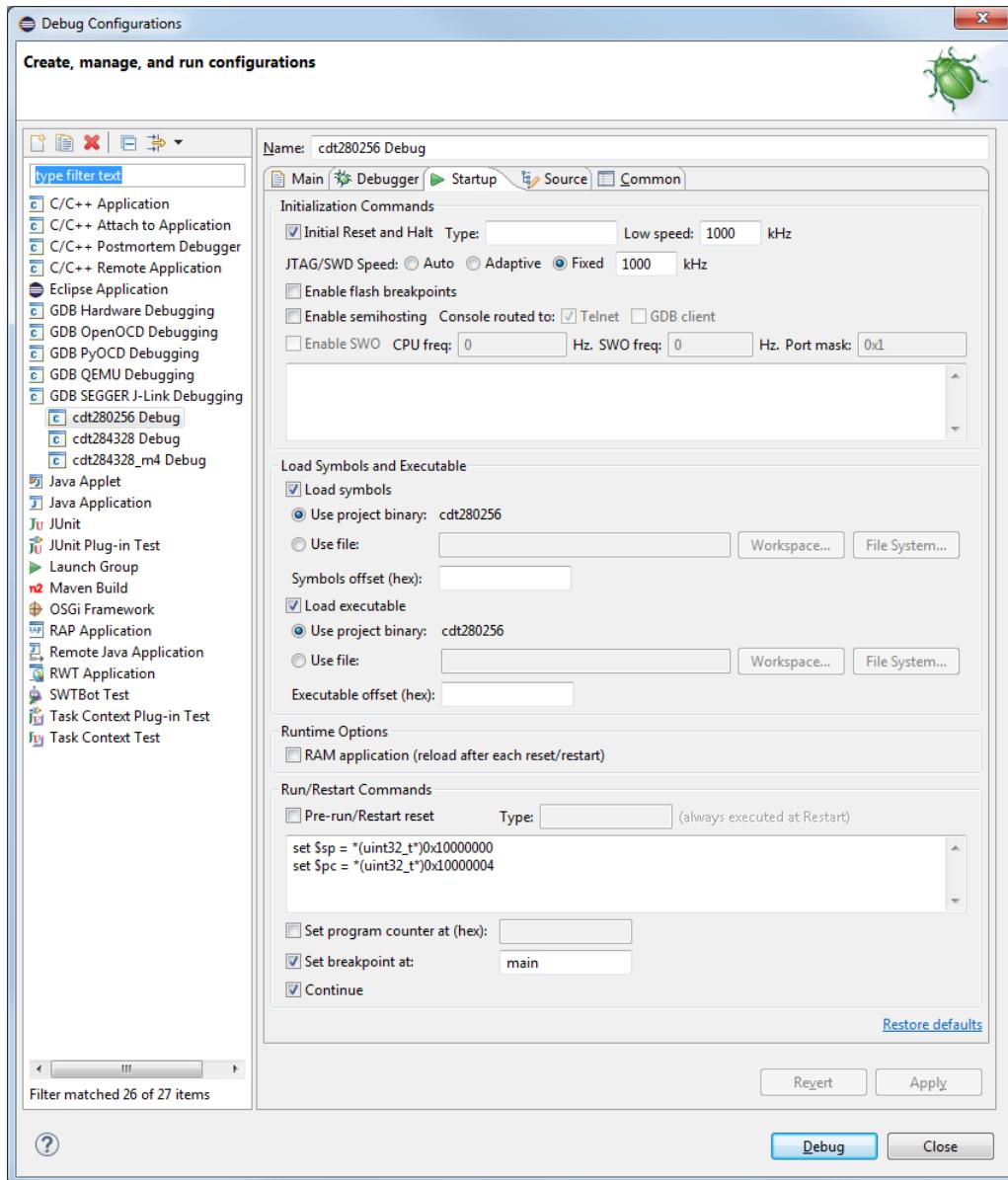


Debugging non-secure M0+ applications with JTAG:

Unselect Pre-run/Restart reset.

Add the following commands to the Run/Restart Commands:

```
set $sp = *(uint32_t*)0x10000000
set $pc = *(uint32_t*)0x10000004
```



Debugging secure M4 applications with JTAG or SWD:

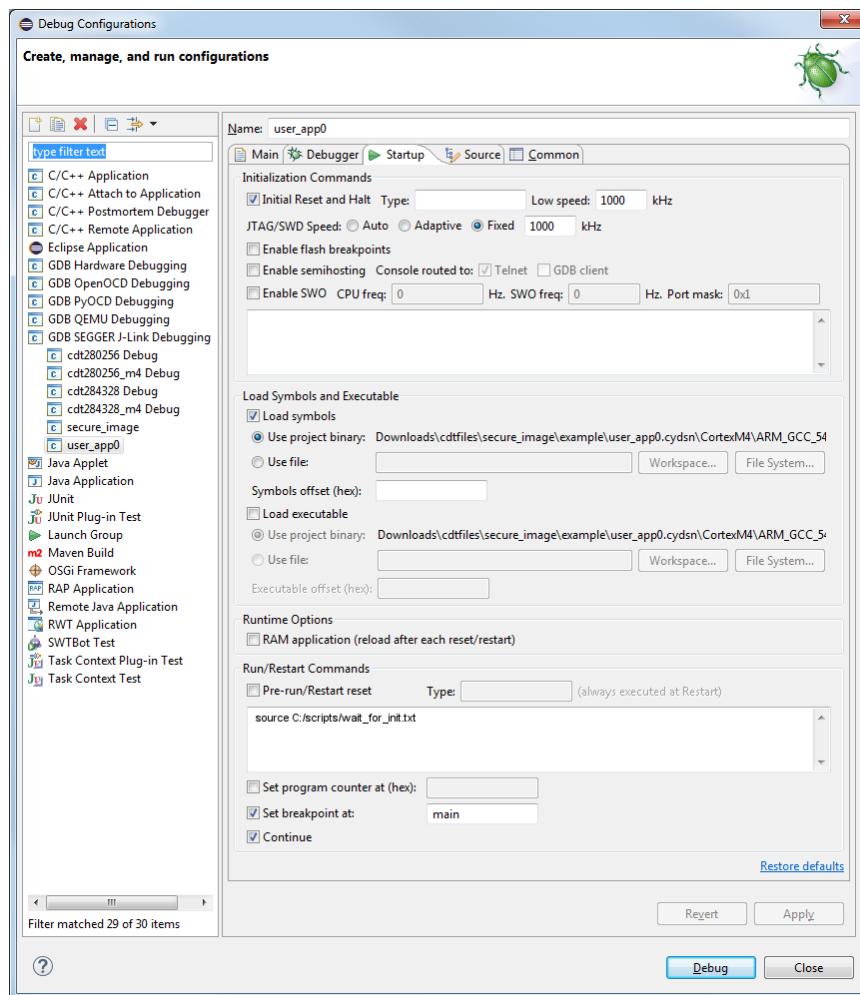
Unselect Pre-run/Restart reset.

Copy the following commands to a text file. For this example, assume that the file is saved as C:\scripts\wait_for_init.txt.

```
set $retry = 10
while (*(uint32_t*)0x402102c0 == 0xFFFFFC00) && $retry > 0
    set $retry = $retry - 1
    # For Mac/Linux
    #shell sleep 1
    # For Windows
    shell timeout /T 1
end
```

Add the following commands to the Run/Restart Commands:

```
source C:/scripts/wait_for_init.txt
```

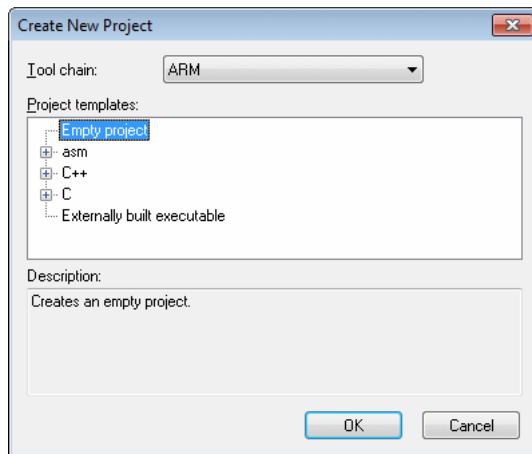


Setting up a PSoC 6 IAR Project

This section covers the process for using generated PSoC 6-based PSoC Creator design files in the IAR environment. If not already done, refer to [Generating PSoC 6 Files for 3rd Party IDEs](#) for the process to generate the necessary IAR files.

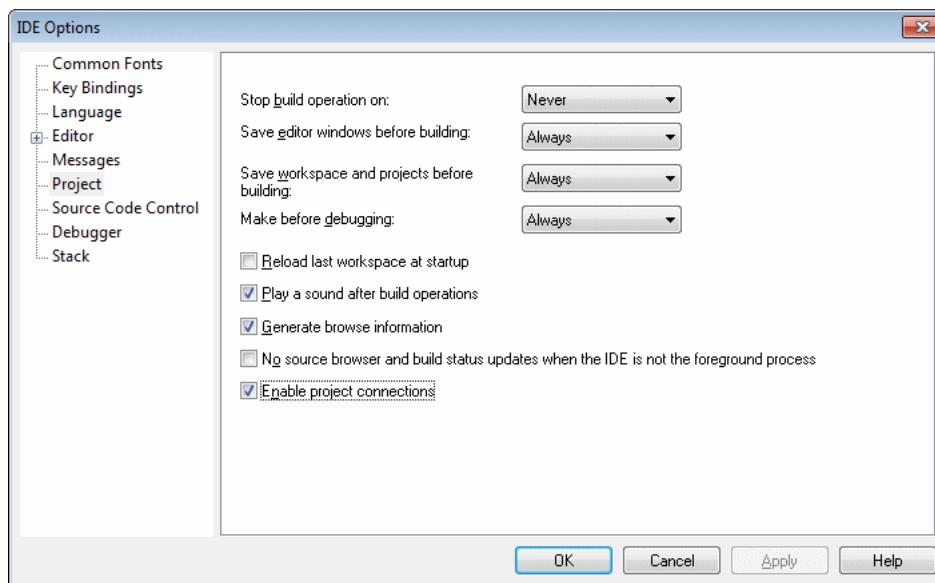
Note If you do not have IAR installed, PSoC Creator will perform the generation with the assumption that IAR version 7.10.3 or higher will be used to create an IAR project. If IAR is installed, PSoC Creator will use that version of IAR. However; when you create your project in IAR, you need an IAR version that includes PSoC 6 devices.

1. Launch IAR Embedded Workbench for Arm (EW-Arm).
2. Create a new Empty project (**Project > Create New Project**), and save the project in the PSoC Creator `<project>.cydsn` directory.



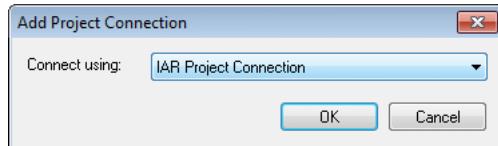
Note For a PSoC 6 multi-core design, two project connection files (DesignName CoreName.ipcf file) are generated by PSoC Creator; therefore, you need to create IAR projects for each core by repeating these steps for each core. Projects need to be created in the same directory as a requirement.

3. Open the IDE Options dialog (**Tools > Options**).

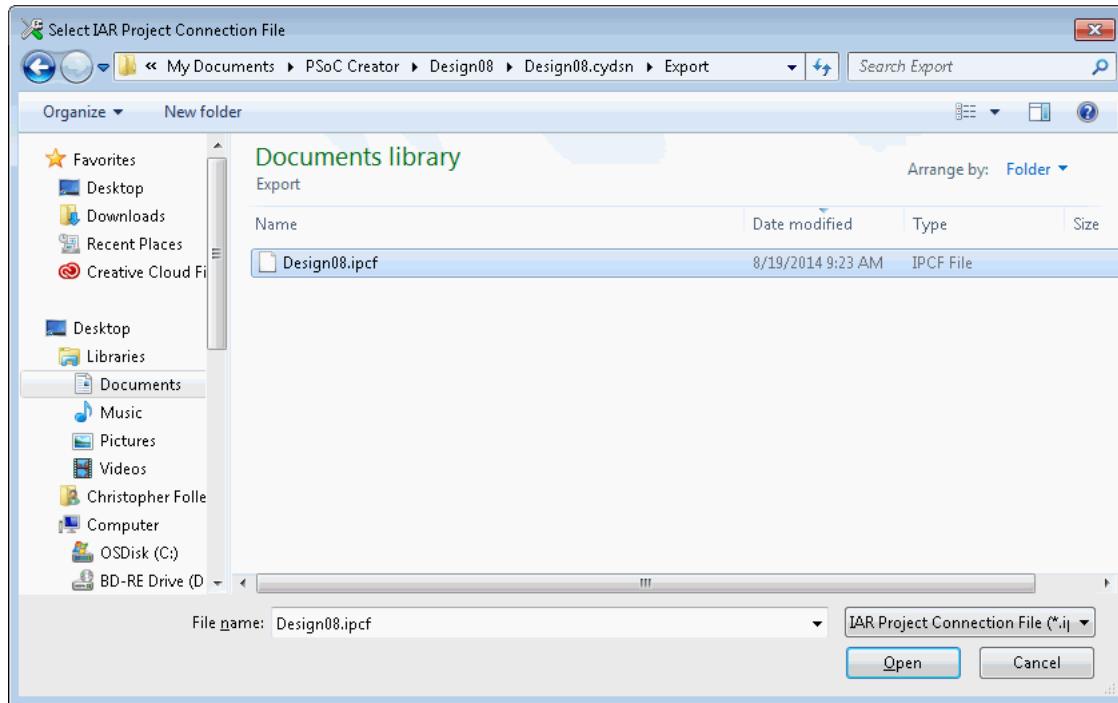


- Select Project in the tree.

- Select the Enable project connections.
 - Click **OK**.
4. Open the Add Project Connection dialog (**Project > Add Project Connection**).

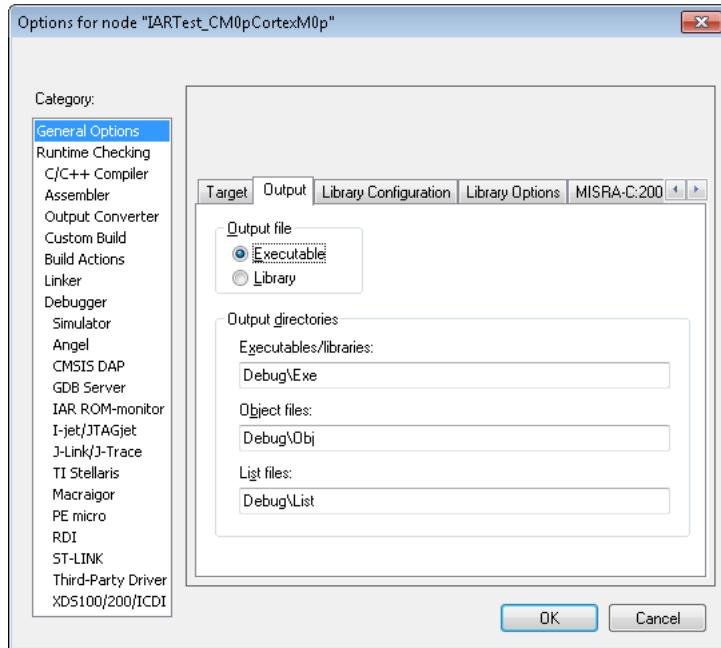


- Select IAR Project Connection.
 - Click **OK**.
5. On the Select IAR Project Connection File dialog, browse to the PSoC Creator Export directory, select the `<project>.ipcf` file, and click **Open**.

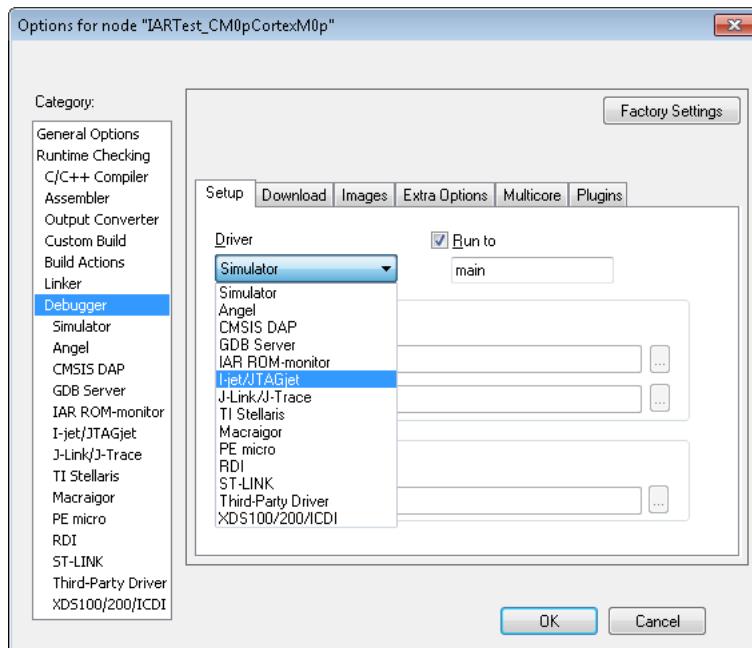


6. In the EW-Arm Workspace window, right-click on the project and select **Options...** to open the Options dialog.
7. For dual core designs, it is important to specify two different folders for the object files and list files of each core-specific project. You can leave "Debug\Obj" and "Debug\List" folders for the CortexM0p-based project and change the name of these folders for CortexM4-specific project. For CortexM4-specific project, on the General Options page, select **Output** tab, and change the Obj and List folder names in Object files and List files text boxes, respectively. Keep the content of Executables/libraries text box as is.

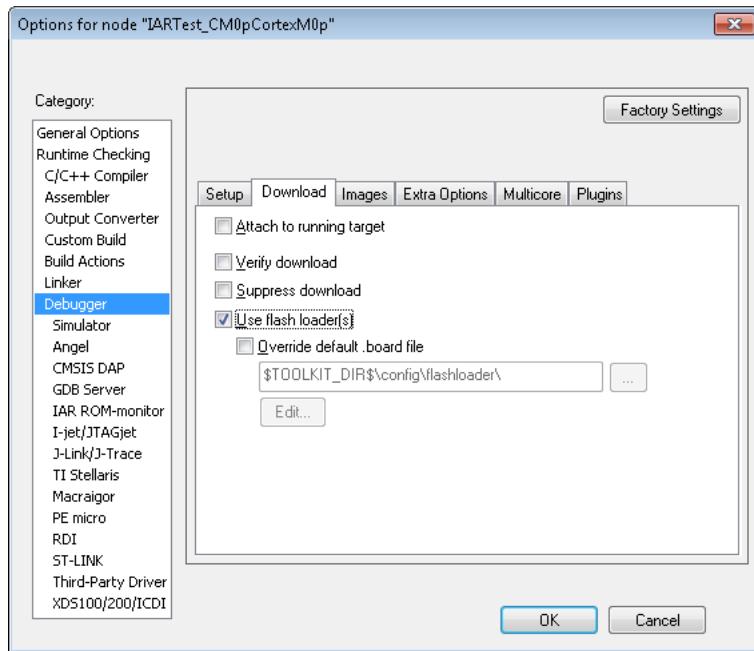
Notice that PSoC Creator post-build scripts for the CortexM4 consider the default executable output directory "Debug\Exe" for the CortexM0p based project. If you are changing the default executable output directory for the CortexM0p-based project, make sure you update the variable "CORTEXM0P_OUTPUT_DIRECTORY" under the IAR section in the *postbuildCortexM4.bat* and *postbuildCortexM4.sh* scripts located in the <project>.cydsn\Export directory.



- On the Debugger page, click the **Driver** drop-down menu and select the appropriate debugger probe: either I-jet/JTAGjet or J-Link/J-Trace.

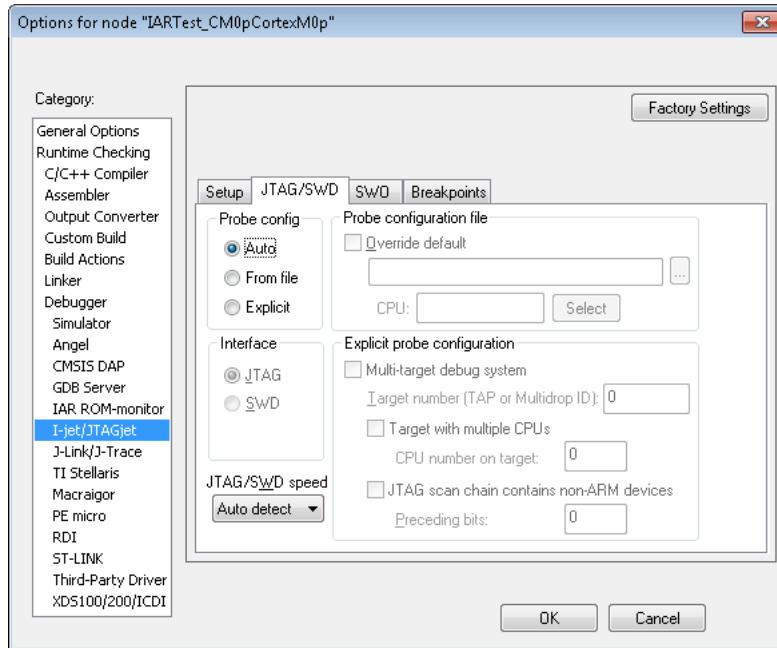


9. On the Debugger page, select the **Download** tab and check the **Use flash loader(s)** option.

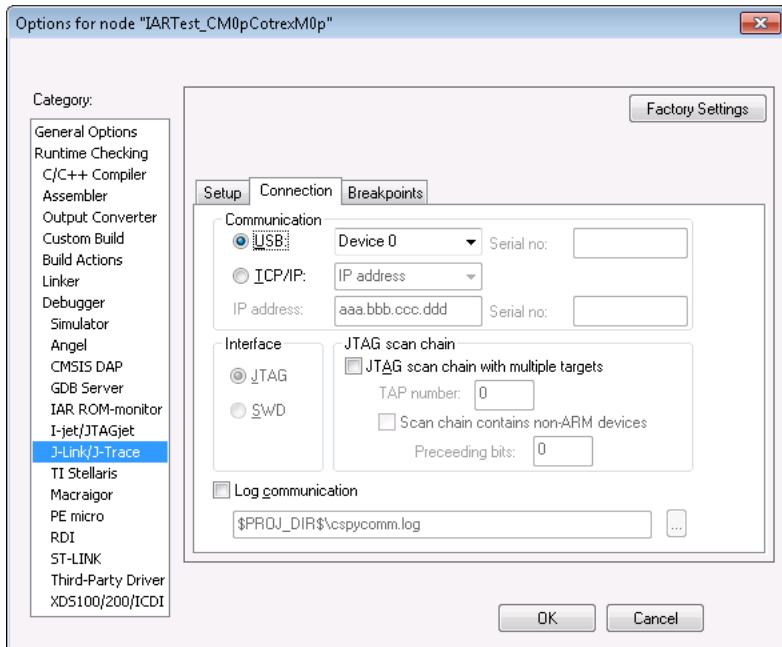


10. Under "Category," select the appropriate debugger option.

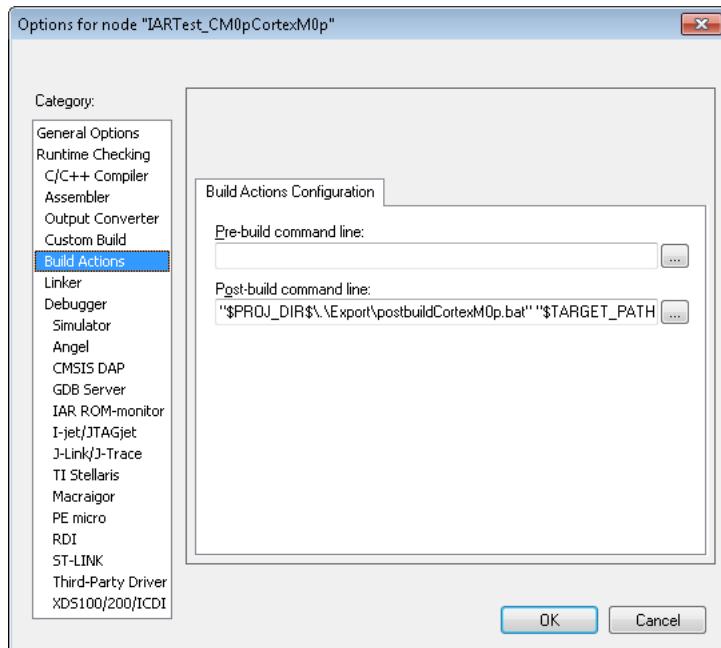
- For I-jet/JTAGjet, select the **JTAG/SWD** tab and set appropriate options.



- For J-Link/J-Trace, select the **Connection** tab and set the appropriate options.



11. On the Build Actions page, enter appropriate pre-build and post-build commands.



PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. For IAR, add these commands to the projects options. PSoC Creator provides a call to our generated pre-build and post-build scripts where needed; you can add your own commands, separating all commands with semicolons.

12. Click **OK** to close the Options dialog.

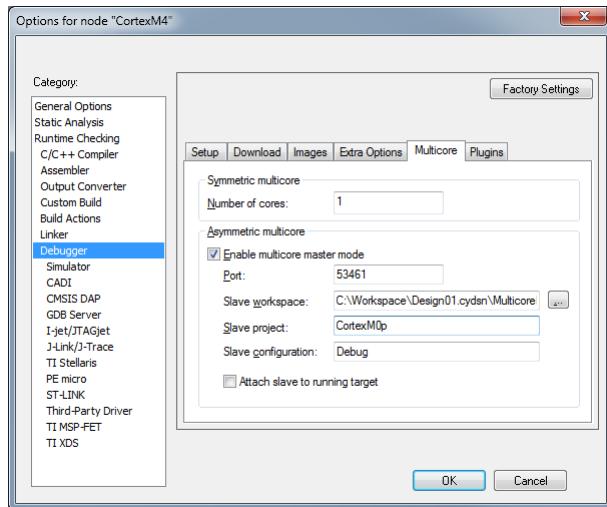
Note For multi-core projects, you need to build the projects for each core in the ascending order from the lower core number to the higher. For example, if you have two projects based on CortexM0p and CortexM4, first build the CortexM0p project then build the CortexM4 project. The output file for CortexM4 project will have the combined

image of both projects. If you have modified the CortexM0p project but not the CortexM4 project, you must perform a "rebuild all" on the CortexM4 project to ensure that the post-build step is executed.

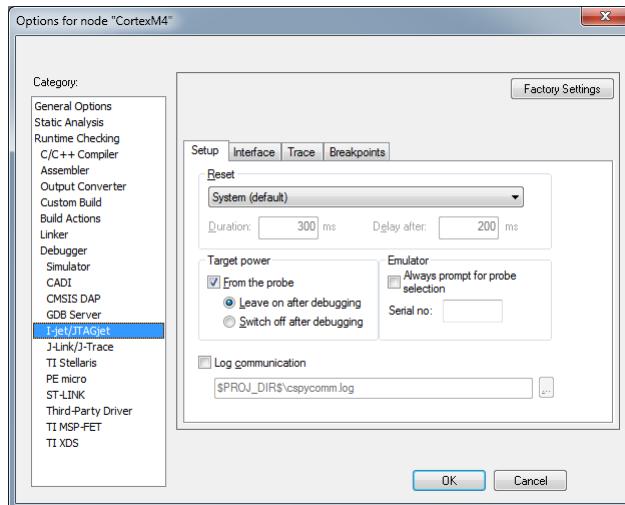
Setting up Multi-core Debugging

If you are using IAR's multi-core debugging feature, you must configure your two projects as follows:

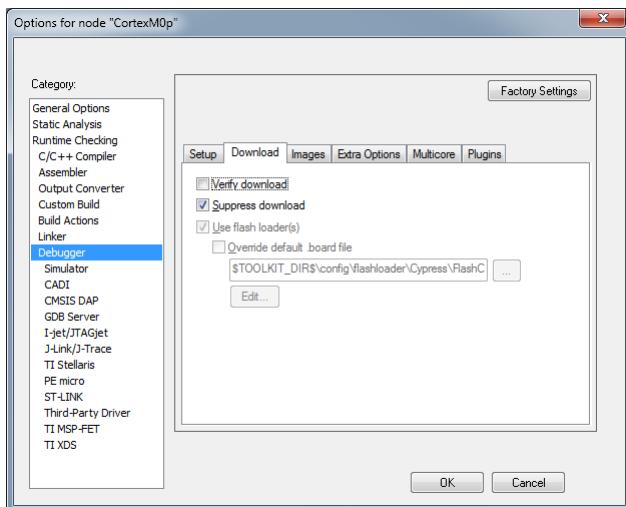
1. Select the **Enable multicore master mode** check box for the CortexM4 project, and set the slave project to the CortexM0p project.



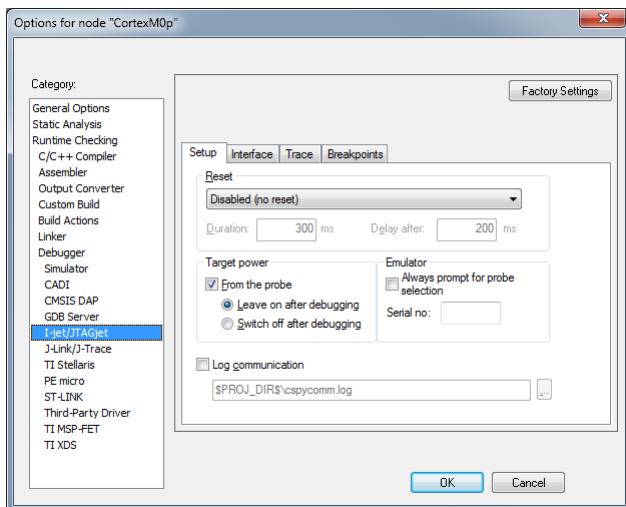
2. In the debug probe configuration, set the **Reset** mode to "System (default)."



3. For the CortexM0p project, select the **Suppress Download** check box.



4. In the probe configuration, set the **Reset** mode to "Disabled (no reset)."



You can now build, program and debug your EW-Arm project. Refer to the IAR documentation as needed.

For information about how to use the IAR I-jet/Debugger system, go to the IAR IDE. In the Help menu, open the documents named *C-SPY Debugging Guide* and *I-jet User Guide*.

For information about J-Link, refer to the Segger J-Link documentation.

Creating µVision Projects for PSoC 6

Note You need µVision 5 or later to import CMSIS Pack in µVision.

For PSoC 6-based designs, PSoC Creator does not generate a µVision project. You must create a µVision project per core and build them in the ascending order based on the core numbers. For example, if your design uses two cores: CortexM0P and CortexM4, first build CortexM0P-based project, then build the CortexM4-based project. The .axf file for CortexM4 project will have the combined image of both projects.

When you build a PSoC 6-based design for use in µVision, you must choose the CMSIS-Pack **Generate** option on the [Target IDEs Build Settings](#) dialog. As part of a build, PSoC Creator generates a CMSIS-Pack for your design. The pack will include Component firmware and peripheral driver library (PDL) files, based on the selected PDL (see [Peripheral Driver Library](#)). You can choose to use a different compatible PDL pack in the µVision Manage Run-Time Environment, instead of including the PDL used to build your design in PSoC Creator.

After building your design in PSoC Creator, the generated CMSIS-Pack is located in `<project_name.cydsn>/Export/Pack` folder (`Cypress.<pack_name>.<version>.pack` for example, `Cypress.MyPack.1.0.0.pack`). Specify the `<pack_name>` and `<version>` on the Target IDEs Build Settings dialog. The following additional steps are required for a PSoC 6-based µVision project. You will need to create a project for one core, and then repeat the steps to create another project for the second core, with the exceptions noted:

1. Double-click the pack file to install it in the default location for packs.
2. Open µVision and create a µVision project.

Note You must create the project for the second core in the same folder as the first project.

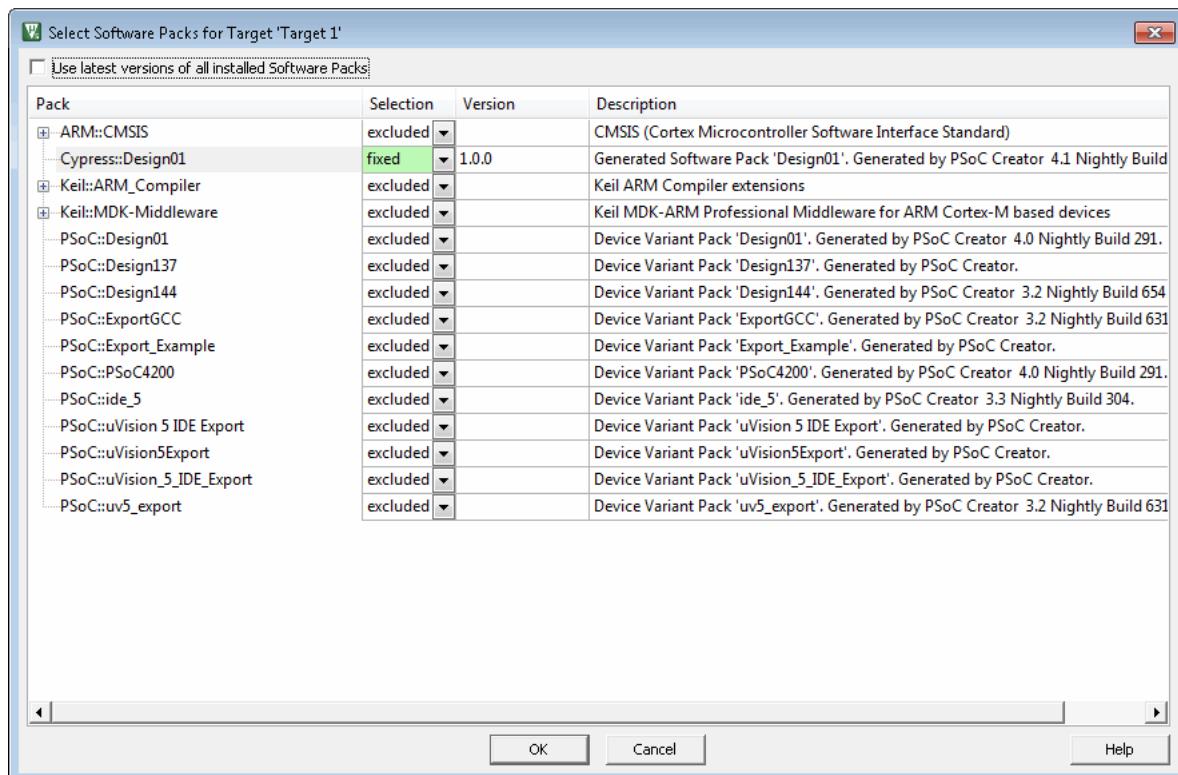
Note It is important to specify two different folders for the object files of each core-specific project. When building a project, µVision automatically generates an output directory named "Objects" in the directory that the µVision project exists, unless you specify another directory through "Options for Target" dialog, **Output** tab. You can leave the "Objects" folder for the CortexM0p-based project and create another output folder in the µVision project directory, for example named "Objects_M4", for the CortexM4-based project. Then, before building the CortexM4-based project, open "Options for Target" dialog, navigate to **Output** tab and from **Select folder for objects**, navigate to the output folder that you created.

The generated pack includes the device information for multiple cores. When you create a project, choose the device per target core from the pack. The device specified in the µVision project **must** match precisely the device in the CMSIS Pack. This is important for [step 6](#).

3. As part of creating a µVision project, the Select Device for Target dialog will display. Select the newly installed pack and appropriate core for the project from the list shown.

Note If you created a CMSIS Pack with a custom vendor name, it will still be listed in the Cypress section of this dialog, as this list is organized by hardware device vendor.

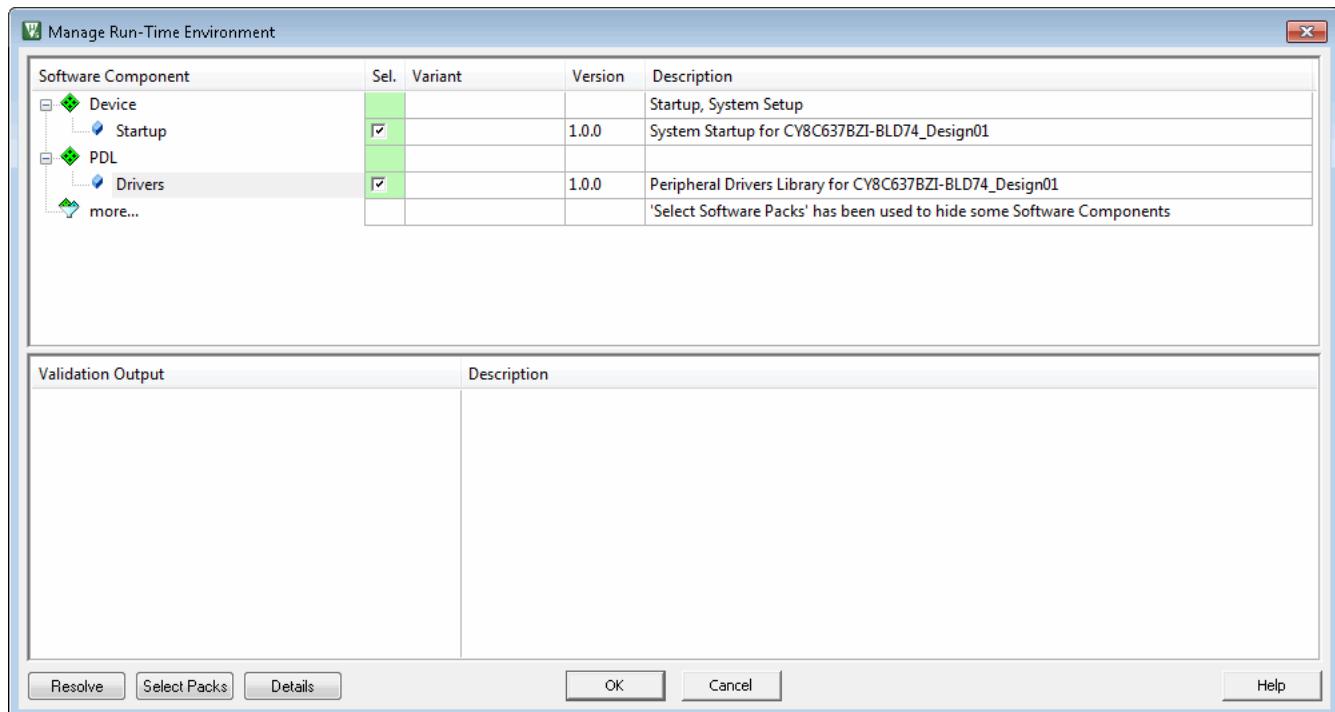
4. The Manage Run-Time Environment dialog will open. Click the **Select Packs** button.
5. On the Select Software Packs for Target dialog:
 - Unselect the Use latest versions of all installed Software Packs check box.
 - Under the **Selection** column, select "fixed" from the drop down menu for the CMSIS Pack that PSoC Creator generated for your configured PSoC 6 device. All other unused packs should be excluded.
 - Click **OK**.



6. On the Manage Run-Time Environment dialog, select **Device/Startup** and **PDL/Drivers**.

Note If at this point you do not see the required software components, check the device used in your µVision project. The device must match the device used in the CMSIS Pack, or the software components will not appear.

Note The **PDL/Drivers** are the PDL files that PSoC Creator used to build your design and included in the generated pack specific to your design. You have the option to not select the PDL files from the PSoC Creator generated pack, and instead use other PDL files compatible with your design. If you install a PDL pack, select it in the Select Software Packs for Target dialog, and it will be available in the Manage Run-Time Environment.

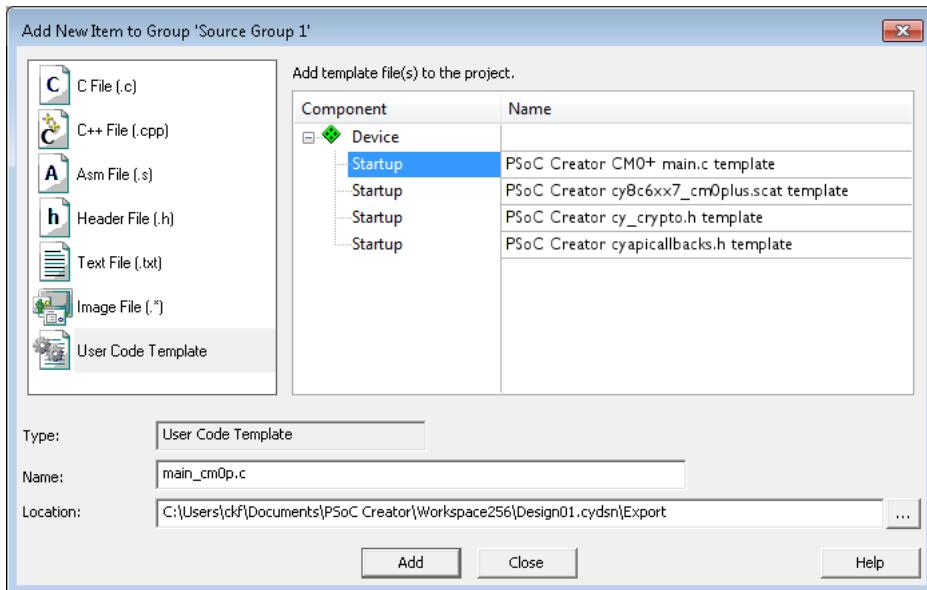


7. Click **OK** to close the Manage Run-Time Environment dialog.

8. Then in the main µVision window, add the application files to your µVision project. Application files include *main_<core>.c* and any other file you might have added to your design. You can add application files to the Source Group.

If you have not added any code to these files, you can create new ones quickly in your µVision project as follows:

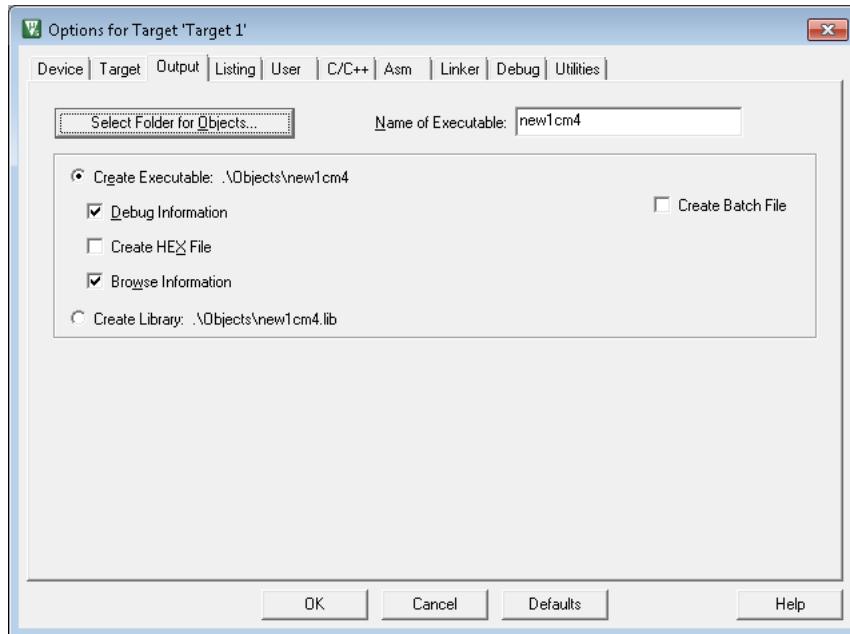
- In the Project window, right-click on the Source Group folder and select **Add New Item to Source Group...**
- On the Add New Item to Group dialog, select the "User Code Template" icon in the left pane, and create the linker scatter file from the template, because later steps in this process will expect it.
- You can also create your *main_<core>.c* and any application files from the templates provided.



Next Steps:

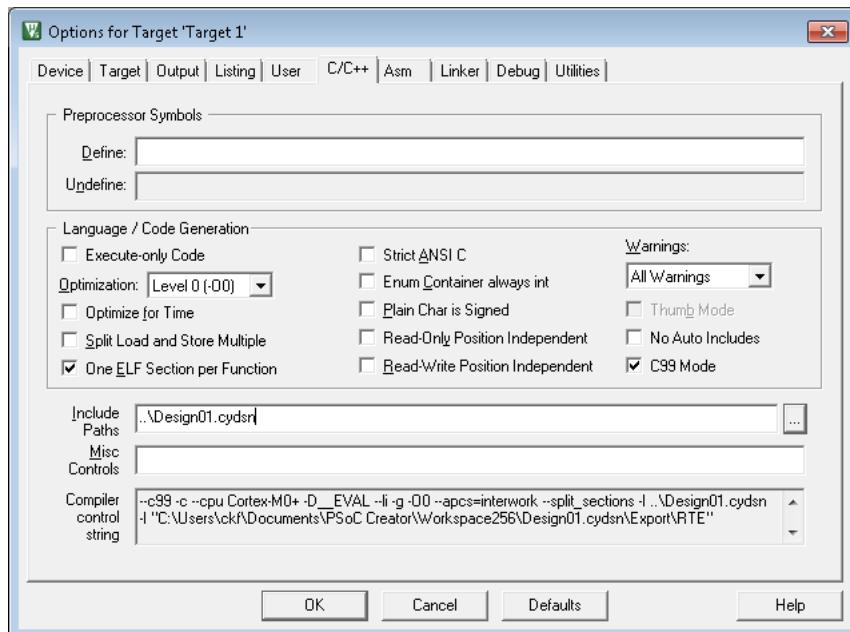
1. Open the "Options for Target" dialog.
2. For dual core designs, when creating a CortexM4-based project, create a new folder for objects in the same directory as the project exists, and name it different than the folder for objects for CortexM0p-based project (default is "Objects"). Then, navigate to the **Output** tab and from **Select folder for objects** navigate to the folder that you created.

Notice that PSoC Creator post-build scripts for CortexM4 consider the default output directory "Objects" for a CortexM0p-based project. If you are changing the default output directory for a CortexM0p-based project, make sure you update the variable "CORTEXM0P_OUTPUT_DIRECTORY" under the CMSIS section in the *postbuildCortexM4.bat* and *postbuildCortexM4.sh* scripts located in the <pack_vendor_name>\<pack_name>\<pack_version>\Device\<deviceName_projectName>\Other directory. Clear the read-only flag of a build script file before modifying it.



3. Select the **C/C++** tab, and then select/add the following:

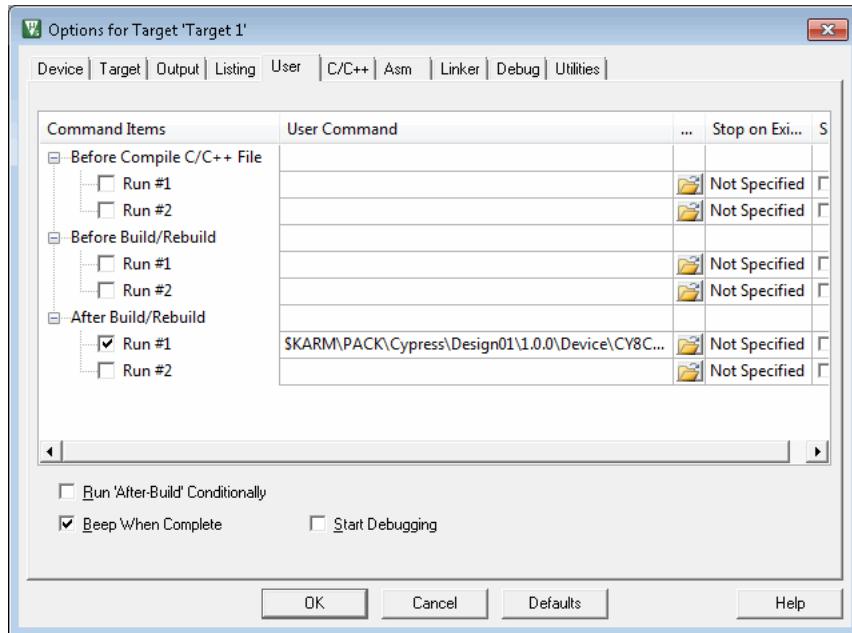
- Select **C99 Mode** option (PDL is developed based on C99).
- Add the design directory or any other directory containing the application files to the **Include Paths**.
- For CortexM4-based projects add **--fpu=fpv4-sp** to **Misc Controls**.



4. Select the **User** tab. The path to pre/post build scripts for PSoC 6 devices is similar to the following.

```
$KArm\PACK\Cypress\Design01\1.0.0\Device\CY8C68237BZ-
BLE_Design01\Other\postbuildCortexM0p.bat "#L" -p "$P" "cmsis"
"$KArm\PACK\Cypress\Design01\1.0.0\Device\CY8C68237BZ-BLE_Design01\Other\win\elf"
```

Since the pack includes the script, Cypress uses the default pack path and the command. It will be filled automatically, but you can double check it. If you are using a custom path for pack installation directory, you need to modify the post-build command to replace **\$KArm\PACK** with your custom pack installation directory.

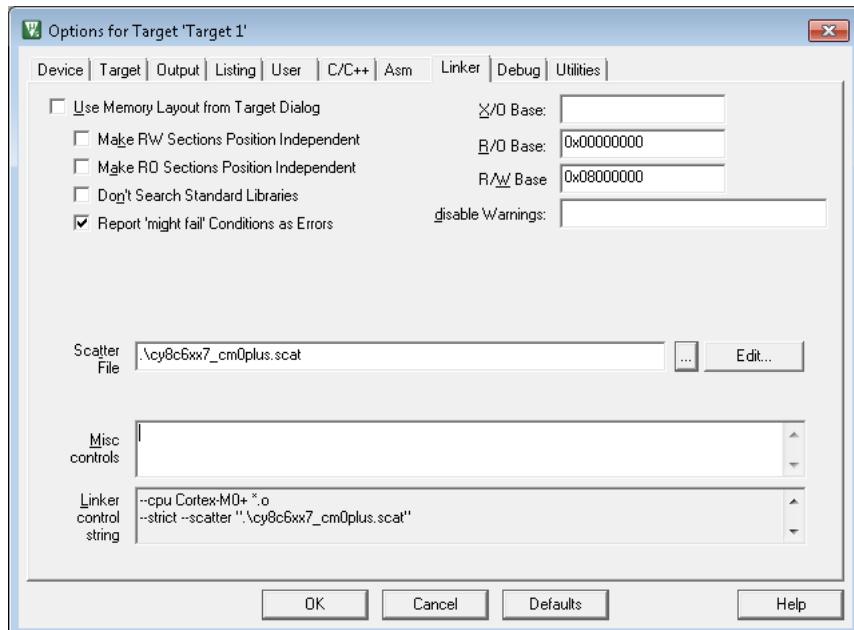


Note PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. You can add these commands on the **User** tab. PSoC Creator provides a call to generated pre-build and post-build scripts where needed. You can add your own commands in the unused entry fields on this tab.

5. Select the **Linker** tab and unselect the **Use Memory Layout from Target Dialog** check box. Provide the path to the linker according to the processor in your project. The path to linker script file will be similar to the following:

```
.\cy8c6xx7_cm0plus.scat
```

For CortexM4-based projects add `--fpu=fpv4-sp` to **Misc Controls**.



6. Click **OK** to accept all the changes, close the settings dialog, and build the code.

Note The linker files used by PSoC Creator are generic to handle all common use cases. Your project may not use every section defined in the linker file. In that case, you may see warnings during the build process. You can ignore or suppress the warning, or modify the linker command file to eliminate the warning.

Setting Up ULink2/ULink Pro and Segger J-Link Debugger Probes for PSoC 6

These steps are for users of the ULink2/ULink Pro and Segger J-Link debugger probes. These instructions apply only to PSoC 6 devices.

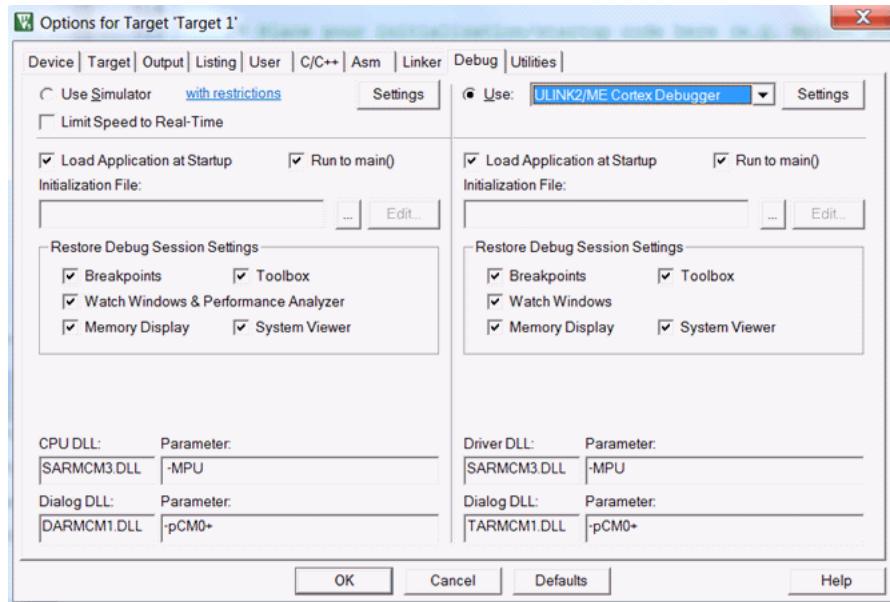
If not already running, launch μ Vision, open your project, and select your build target. Then follow these steps.

Note The probe must be connected to the host computer and to the board for these instructions.

1. Select **Project > Options for Target** to open the dialog, and go to the **Debug** tab.
2. Select the appropriate debugger and click on **Settings**.

Note Do not select the Cypress MiniProg3 option, it does not yet work with PSoC 6. Select the correct ULINK or J-LINK option.

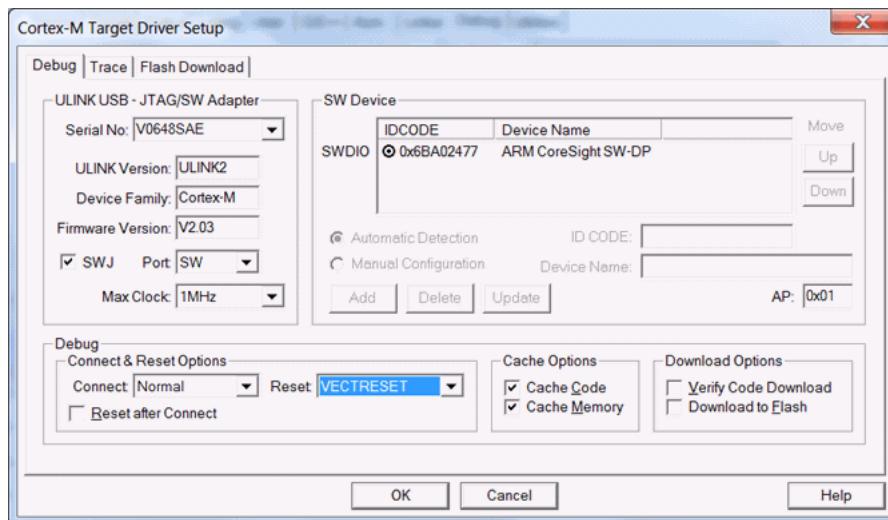
When using the Segger J-Link debugger for dual core devices, if you launch the J-link device detection dialog for the project for the first core, it will generate a *JLinkSettings.ini* file in the project directory. In this case, when you launch the J-Link device detection dialog for the project for the second core, make sure you first delete the *JLinkSettings.ini* file from the project directory.



3. On the Driver Setup dialog, on the **Debug** tab, select the appropriate **Port** (SW or JTAG).

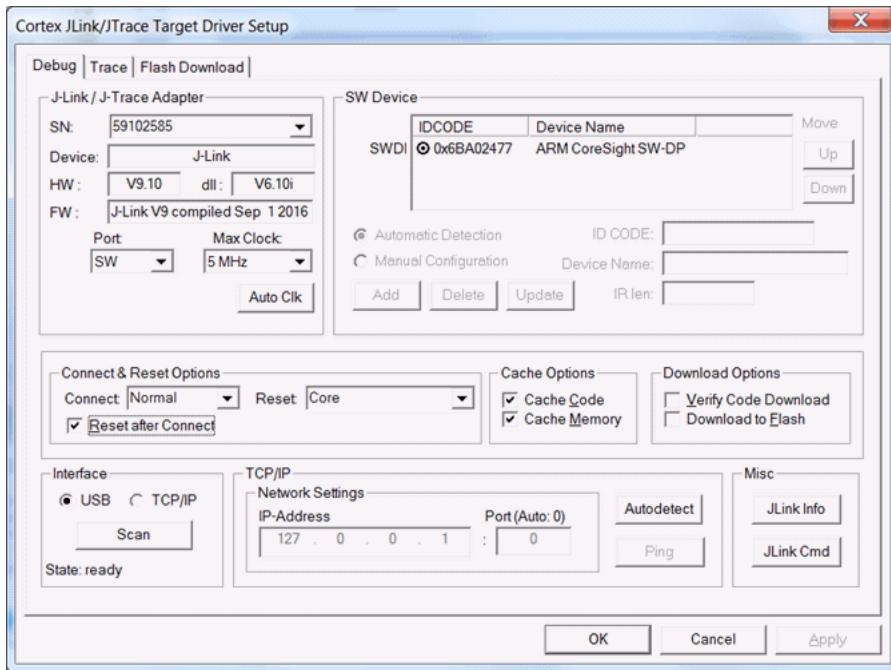
For ULink2/ULink Pro debuggers, do the following:

- Set **AP** (Access Port) settings: For CM0p, use 0x01; for CM4, use 0x02.
- Select "VECTRESET" for **Reset**.
- Unselect the **Reset after Connect** check box.

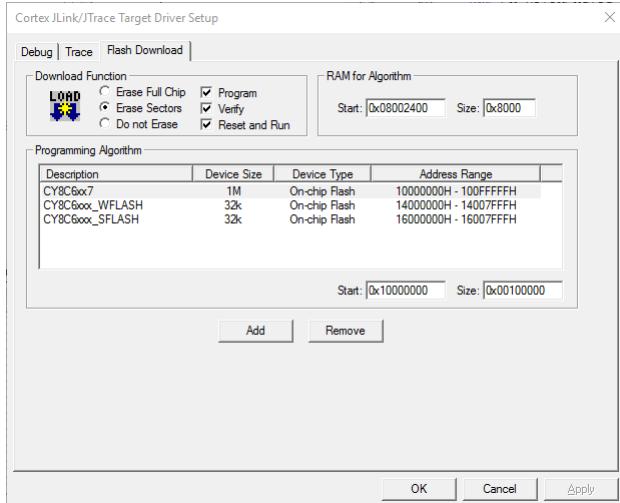


For Segger J-Link Debuggers, select "Core" for **Reset**.

For PSoC 6, select "Core" when debugging the CM0p and "Normal" when debugging the CM4.



4. Click on the Flash Download tab and select Erase Sectors.



The *.FLM file should be present in a path similar to the following, based on the µVision installation directory and your chosen pack name information:

C:\Keil_v5\Arm\PACK\<PackVendor>\<PackName>\<PackVersion>\FLM\<CypressDeviceName>*.FLM

"RAM for Algorithm" values for Keil ULink and Segger J-Link Debuggers

PSoC Device family	RAM for Algorithm		Programming Algorithm *
	Start	Size	
PSoC 6	0x08002400	0x8000	CY8C6xx6 (512kB) Flash CY8C6xx7 (1MB) Flash CY8C6xxx_WFLASH (Work Flash) CY8C6xxx_SFLASH (Supervisory Flash)

Notes:

For dual core projects, when you launch the J-link device detection dialog for the project per first core, it generates a *JLinkSettings.ini* file in the project directory. To launch the J-Link device detection dialog for the project for the second core, the *JLinkSettings.ini* file should be deleted from the project directory.

For more information about how to set up your project; refer to the *Third-Party Tools for Cypress Devices User Guide.pdf* file in the "3rd_Party_Configuration_Files/Documents" folder in the root installation folder of PSoC Programmer.

Add SMIF Flashloaders to PDL Build

In order to add SMIF flashloaders to the build, device programming must be done in two passes:

- Program all internal flash (FLASH, SFLASH and optionally WFLASH). Appropriate loaders must be added in the debugger configuration dialog and the SMIF loader must be removed from the list.
- Program SMIF only (only the SMIF loader must be added to the list in the debugger configuration dialog).

Therefore, SMIF loader must exist along with other loaders, but it should not be added to the list of project loaders by default.

See Also:

- [Target IDEs Build Settings](#)
- [Peripheral Driver Library Build Settings](#)

Building PSoC 6 Designs with Make

Generated Files:

If you have enabled makefile generation in your project's [Target IDEs Build Settings](#) page, the following files will be generated on successful completion of a project build:

File	Notes	Location
<i>makefile</i>	Top-level GNU Make compatible makefile. This file may be updated or altered as desired.	
<i>platform_debug.mk</i> - or - <i>platform_release.mk</i>	Platform and toolchain specific configuration. The Target IDEs feature will generate a <i>platform_debug.mk</i> or <i>platform_release.mk</i> file, depending on whether PSoC Creator is configured to create debug or release builds (see Building a PSoC Creator Project). These files may be updated or altered as desired.	project directory (<project_name>.cydsn)
<i>app_source.mk</i>	Application firmware source. This file may be updated or altered as desired.	
<i>gen_source.mk</i>	PSoC Creator generated source code. This file is generated in the <i>Generated_Source/PSoC6</i> folder. This file should NOT be modified. It is automatically re-written by PSoC Creator as a part of the build process. This file is written out for the selected toolchain. Later, it will be updated during each build process, based on the current toolchain in the Build Settings . That is, if you build and change from GCC to MDK, the makefiles will likely not work. The <i>gen_source.mk</i> file will contain source code that is not supported by the toolchain in <i>platform_debug.mk</i> (or <i>platform_release.mk</i>). Either update the platform configuration file, or use the Target IDEs feature to regenerate the files for the new toolchain.	<project_name>.cydsn/ <i>Generated_Source/PSoC6</i>

Important Notes:

- Since the Make utility does not reliably support files with spaces or \$ in the file name, PSoC Creator avoids using them. Also, the tool avoids using colons and slashes in the names of files and folders because some operating systems and drive formats use these characters as volume and directory separators. Furthermore, non-alphanumeric characters may not be supported by all file systems or operating systems. Punctuation marks, parentheses, quotation marks, brackets, and operators, such as following, are often reserved for special functions in scripting and programming languages:

, [] { } () ! ; " ' * ? < > |

Therefore, PSoC Creator checks the design project name and user source files for white spaces and those special characters in their names. PSoC Creator performs the check as part of the build and the error message specifies the affected files. In this situation you need to address the errors and try to re-build the design.

- When making the makefile for a dual core project, it generates the elf file and hex file per core and locates the files related to each core under a directory with the core name (for example: output/debug/CortexM4). Then it merges them together and locates the final elf and hex files in the make output directory (for example: output/debug).

- The makefile uses BASH scripts (*prebuild.sh* and *postbuild.sh*) by default. If you would like to use BATCH scripts, you need to modify the makefile appropriately.
- If you are using Windows, you need CYGWIN (Make package) or MSYS installed in your machine. Cypress has tested this feature on CYGWIN_NT-6.3-WOW64 1.7.33-2(0.280/5/3) i686 cygwin (Make 4.0), and MinGW32_NT-6.1 1.0.18(0.48/3/2) 2012-11-21 i686 Msys (Msys 2013072300).
- If you would like to use another toolchain rather than the one used during the build, you can modify the *platform_debug.mk*/*platform_release.mk* file to remove the text in front of the TOOLCHAIN_DIR variable and write the path to the target toolchain instead. Be sure to change back slashes to forward slashes in the path.
- You need to install Arm GCC on Linux OS. It can be downloaded from: <https://launchpad.net/gcc-arm-embedded/>
- The additional library files to link, which you added to your design through the Build Settings dialog (that is, by using the linker's -L and -l arguments), will be populated as linker options in the *platform_debug.mk*/*platform_release.mk* file. You may add more library files to the APP_LIBS section in the *app_source.mk* file.
- If you generate a makefile, but then make significant changes to your design (for example, changing the selected toolchain) and regenerate a new makefile, you must run "make clean" before running "make" to ensure your make-based build is properly up to date.
- PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. For generated makefiles, you can add these commands to the top-level makefile, as part of the *prebuild_** and *postbuild_** rules.
- PSoC Creator only propagates project-level Build Settings, such as compiler optimization level. The export process does not support propagation of file-level Build Settings to the makefile.

See Also:

- [Generating PSoC 6 Files for 3rd Party IDEs](#)
- [Build Settings](#)

PSoC 4 and PSoC 5LP Designs

This section contains the following topics:

- [Exporting a Design to 3rd Party IDEs](#)
- [Using PSoC 4/PSoC 5LP Designs with 3rd Party IDEs](#)

Exporting a Design to 3rd Party IDEs

The IDE Export Wizard dialog provides support for developing application firmware in 3rd party development environments. Once the design has been exported, you can write, debug, and test firmware in your preferred environment.

Note This wizard applies to PSoC 3, PSoC 4, PSoC 5LP and FM0+ devices only. **It does not apply to PSoC 6 devices.** For PSoC 6 devices, use the [Target IDEs](#) section of the Build Settings dialog to choose third party IDEs for which to generate files. For PSoC 3 devices, see [Exporting a PSoC 3 Design to Keil µVision IDE](#). For FM0+ devices, see [Exporting a FM0+ Design to Makefile](#).



The export process supports the generation of new projects, as well as updates to existing ones. With the update option, you can make last-minute changes to the hardware and quickly update your project.

To Open this Dialog:

- Develop your PSoC Creator design as usual.

Note If you want to program the device using JTAG in your desired 3rd party IDE, you must set the Select Debug option to JTAG in [PSoC Creator System Editor](#).

- When complete, click the **Project** menu and select **Export to IDE...**

To Perform the Export:

Depending on the selected device, you can export the design to the 3rd Party IDEs shown on the dialog. Select one of the appropriate options, and refer to the related section for more information.

- [Exporting a Design to Eclipse IDE](#)
- [Exporting a Design to IAR IDE](#)
- [Exporting a PSoC 4/PSoC 5LP Design to Keil µVision IDE](#)
- [Exporting a Design to Generated CMSIS-Pack](#)
- [Exporting a Design to Makefile](#)

IMPORTANT All devices using 3rd party programmers except µVision using MP3 require that the project exported to the appropriate IDE have a [System Editor Debug Select](#) value of anything other than GPIO. If a project with **Debug Select** set to GPIO is exported, it will be able to program only one time. Subsequent attempts to program via the 3rd party will fail. This is a limitation of the Arm standard acquire sequence, which is not aware of the special acquire sequence used by Cypress for our devices.

Notes

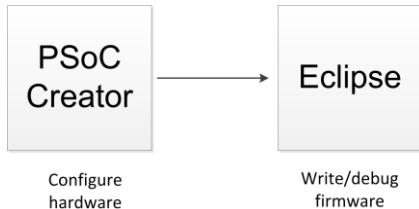
- If the project has not been built, you will be prompted to build it.
- If the project is out of date, you will be prompted to re-build it.

Exporting a Design to Eclipse IDE

Overview

Users can export designs from PSoC Creator and continue firmware development work in Eclipse. After the initial export, users can re-export additional hardware configuration changes from PSoC Creator to Eclipse. Eclipse will detect such changes and update the design accordingly. The reverse is not true. Do not attempt to import firmware changes made in Eclipse back into PSoC Creator.

Note The Export to IDE wizard does not apply to PSoC 6 devices. For PSoC 6 devices, use the [Target IDEs](#) section of the [Build Settings](#) dialog to choose third party IDEs for which to generate files.



Within Eclipse, users can archive their project in order to snapshot a design and supply it to another firmware developer using Eclipse.

Users must initially install and configure Eclipse as outlined in [Eclipse Installation Configuration](#).

Export a Design to Eclipse IDE

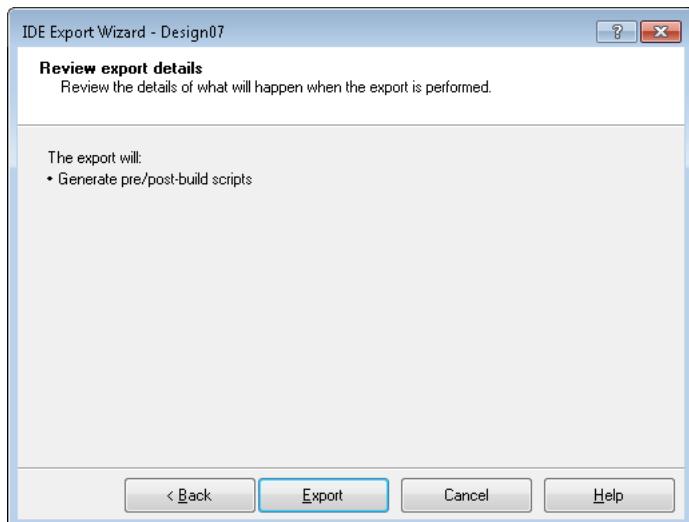
To export your PSoC Creator design for use in the Eclipse IDE, select the "Eclipse" option on the IDE Export Wizard dialog.

Note For PSoC 6 devices, use the [Target IDEs](#) section of the [Build Settings](#) dialog, and enable the **CMSIS Pack** option.

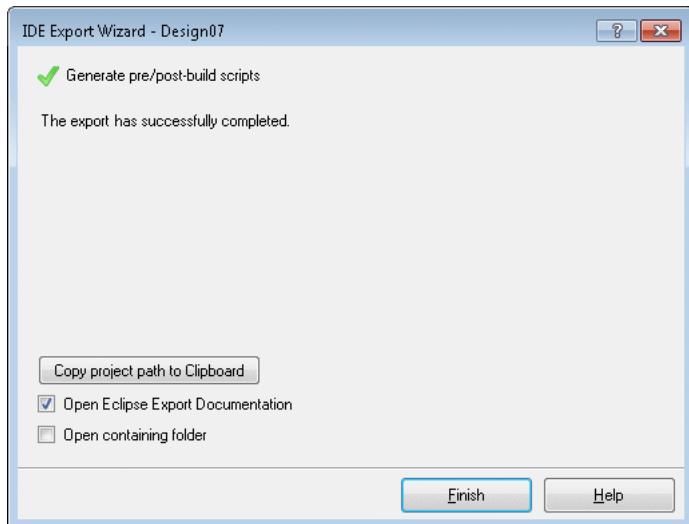


Note The Eclipse IDE option is not available for PSoC 3 projects.

The process is simple. The next step is to confirm that you want to export the design. Click **Export**.



Once the export completes, the final step displays.



- Use the **Copy project path to Clipboard** button to copy the project folder path so you can paste it in the Eclipse New Project dialog as the location path.
- Select the **Open Eclipse Export Documentation** option to display this document.
- Select the **Open containing folder** option to open a folder showing all the files that were exported.

Select the appropriate options and click **Finish** to complete the export process.

Notes:

- The project folder created in Eclipse shares file system folders with the original PSoC Creator project. This is required so that changes to merge regions in any Component files made within Eclipse by a firmware developer can be seen within PSoC Creator during any subsequent updates, and vice versa.
- When deleting a project in Eclipse, do not check the **Delete project contents on disk** check box. This will remove the file system folder contents, making them unavailable in PSoC Creator as well.

Next Steps:

- [Import into Eclipse](#) (see also [Eclipse Installation Configuration](#))
- [Flashing and Debugging in Eclipse](#)
- [Eclipse Bootloader Support](#)

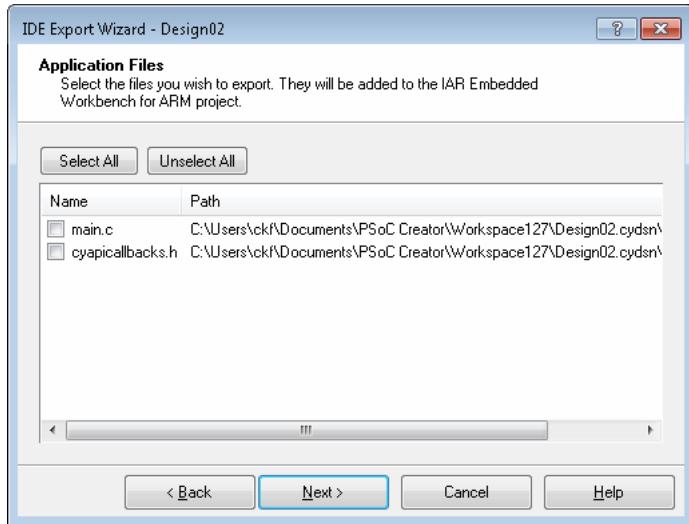
Exporting a Design to IAR IDE

To export your PSoC Creator design for use in the IAR IDE, select the "IAR" option on the IDE Export Wizard dialog.

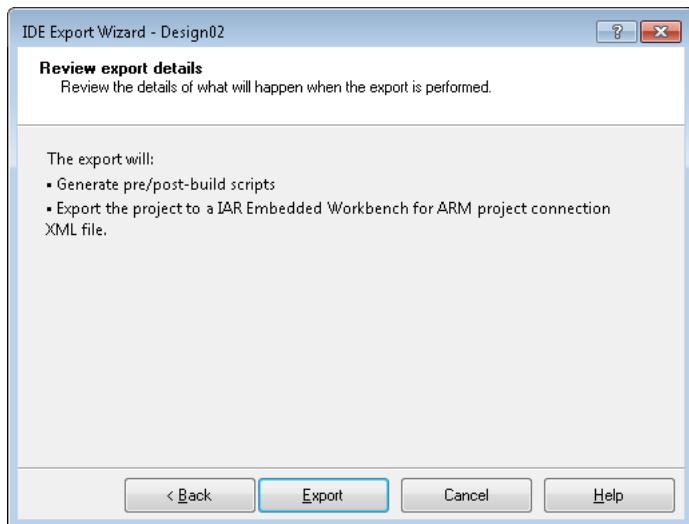
Note The Export to IDE wizard does not apply to PSoC 6 devices. For PSoC 6 devices, use the [Target IDEs](#) section of the [Build Settings](#) dialog, and enable the **IAR** option.



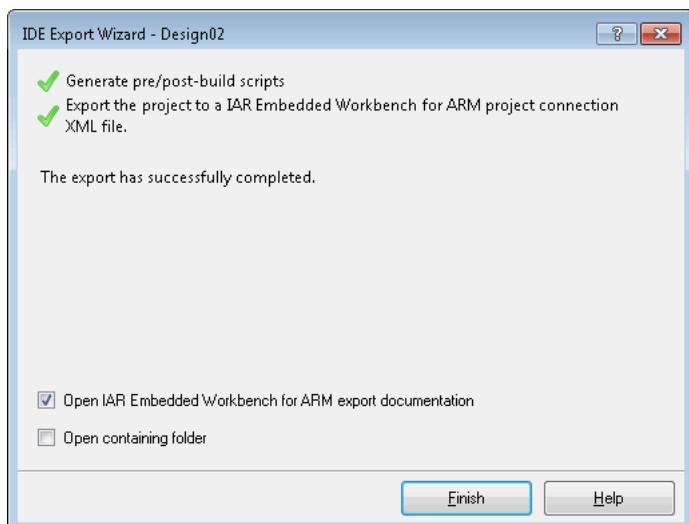
Click **Next >** to go to the Application Files page to select what non-generated code to export to the project:



Click **Next >** to go to the next page of the wizard to show the actions that will occur (such as exporting the selected project XML files).



Click **Export**. The wizard will show the success/failure of the export actions and provide a link to extended documentation on what the user needs to do now.



Optionally select the following action(s) when the export process completes:

- Open the IAR EWSarm Export Documentation. See [Steps for Setting up the IAR Project](#).
- Open the folder containing the project files

Click **Finish** to close the wizard.

See Also:

- [Exporting a Design to 3rd Party IDE](#)
- [Steps for Setting up the IAR Project](#)

Exporting a PSoC 4/PSoC 5LP Design to Keil µVision IDE

PSoC Creator supports multiple PSoC devices as well as multiple versions of the Keil µVision IDE. This topic applies only to PSoC 4/PSoC 5LP devices only.

- PSoC 4/PSoC 5LP device projects can be exported to either µVision 4 or µVision 5 IDEs.
- PSoC 6 devices use a different process. See the [Target IDEs](#) section of the [Build Settings](#) dialog to choose third party IDEs for which to generate files.
- The export to µVision IDE process is slightly different for PSoC 3 devices than it is for PSoC 4 and PSoC 5LP devices. See [Exporting a PSoC 3 Design to Keil µVision IDE](#).
- The **Export to Generated CMSIS-Pack** option applies only to PSoC 4 and PSoC 5LP devices, and it applies only to the µVision 5 IDE. See [Exporting a Design to Generated CMSIS-Pack](#).

Note The Export to IDE wizard does not apply to PSoC 6 devices. For PSoC 6 devices, use the [Target IDEs](#) section of the [Build Settings](#) dialog to choose third party IDEs for which to generate files.

Use PSoC Creator to develop a PSoC hardware design. Then use either PSoC Creator or Keil's µVision IDE for firmware development. To use the µVision IDE, you must [build the design in PSoC Creator](#) and then use the Export to IDE feature.

Note PSoC Creator supports the Advanced System Viewer in Keil µVision. Ensure that you have the appropriate version of µVision to support CMSIS-SVD.

Note When building a project in µVision, all output is written into the UV4Build directory found in your project directory.

For PSoC 4/PRoC BLE/PSoC 5LP, the build and export process is the same for new projects or updated projects.



Changing Device Architectures

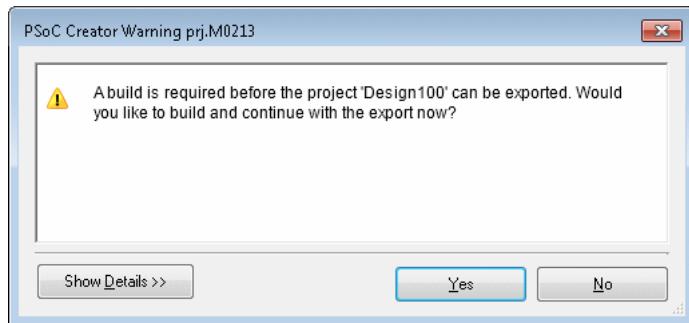
Creating and exporting a project using µVision and then changing the device to a different architecture (for example, PSoC 3, PSoC 4, PSoC 5LP, etc.) and re-exporting is not supported. If you wish to do this, you must manually delete your µVision project (*.uvproj *.uvprojx) before re-exporting. Then, follow the appropriate instructions for the new PSoC device and version of µVision.

Exporting a New PSoC Creator Design:

The initial flow is to create a design in PSoC Creator and export the design to the µVision IDE.

1. Create your design in PSoC Creator in the usual manner.
2. Use the **Project > Export to IDE** menu option to open the IDE Export Wizard dialog.

3. If you have not built the design, you will be prompted to build at this time.



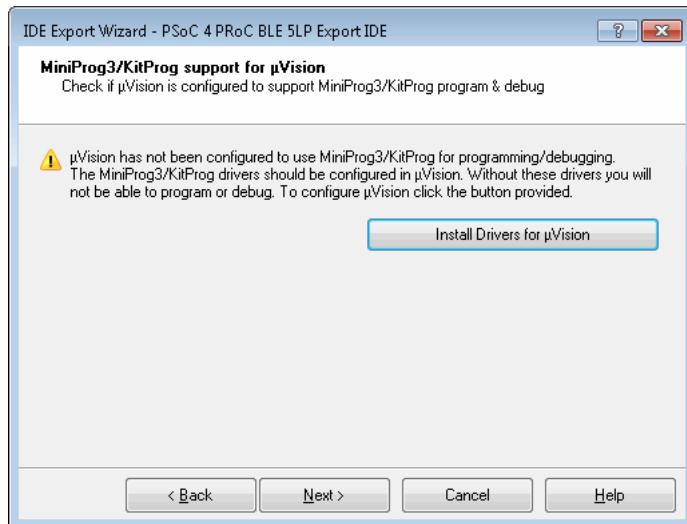
4. Click **Yes** to build. If you click **No**, the export process will be cancelled.

After a successful build, the IDE Export Wizard opens.



- If not already selected, select the µVision option. Click **Next >** to continue.

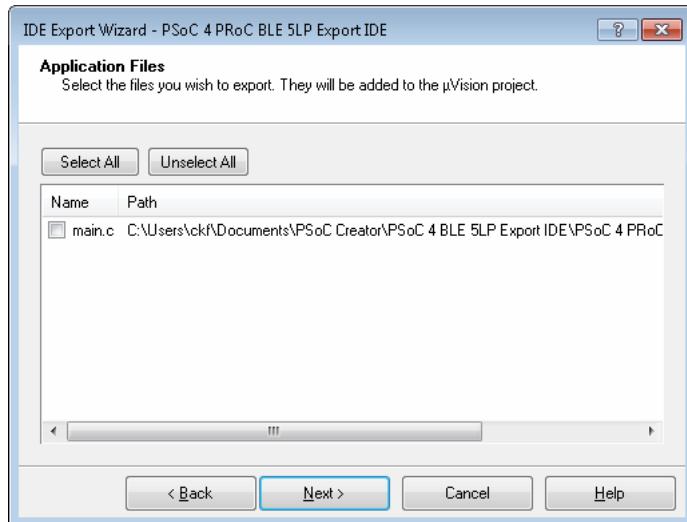
PSoC Creator checks whether MiniProg3/KitProg drivers have been registered with µVision. If they are not registered, the MiniProg3/KitProg support for µVision step opens. However, if PSoC Creator locates a µVision installation with MiniProg3/KitProg support enabled, then it will skip this step.



Note PSoC Creator only examines the first µVision installation found to see if it has MiniProg3/KitProg drivers properly registered. If you have multiple copies of µVision installed on your computer, the auto-detection process may not be accurate. If you are sure you already have drivers registered, you may skip this step of the wizard.

- If you need to register the drivers, click **Install Drivers for µVision** and follow the prompts on the installation wizard. This process is only required once. See [Registering MiniProg3/KitProg Drivers](#) for more information.
- If you need to register drivers with another µVision installation, go to the PSoC Creator **Tools** menu and select **Install drivers for µVision** to launch the installation wizard.
- Click **Next >** to continue.

The Application Files step opens.



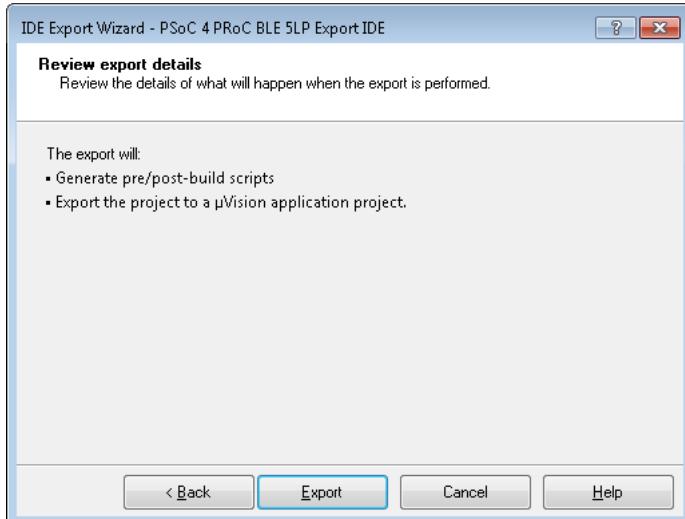
- Select the files you want added to your new µVision application project.

Note The wizard does not copy these files when exporting. It simply adds references to the existing files to the µVision application project.

- To start with an empty µVision application project, click the **Unselect All** button.
- Once the initial export is complete, build settings and file management must be performed within the µVision environment. For example, if you want to add a new source file, select **File > New** in µVision.

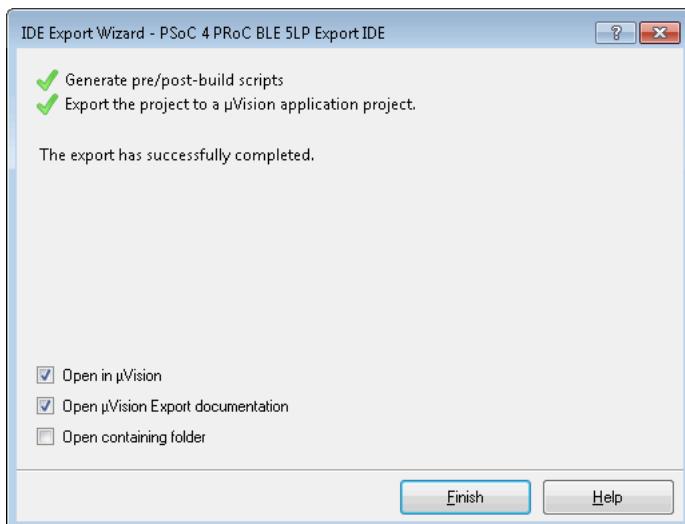
9. Click **Next >** to continue.

The Review export details step opens.



10. Review the export details and click **Export**.

The final step opens showing the completed export.



Optionally select the following action(s) when the export process completes:

- Open the project in µVision
- Open µVision Export documentation

- Open the folder containing the project files

11. Click **Finish** to close the wizard.

Note If you select the **Open the project in µVision** option and if you have multiple versions of the µVision IDE installed (for example version 4 and version 5), this option will launch the version you installed most recently. In this case, if version 4 does not launch, close the µVision 5 IDE, and launch version 4 manually.

See Also:

- [Key IDE Export Files/Projects](#)
- [Opening PSoC 4/PSoC 5LP Projects in µVision IDE](#)
- [GCC Settings in µVision](#)
- [Flash Programming/Debugging using µVision](#)
- [PSoC Creator Toolchain Settings](#)
- [Registering MiniProg3/KitProg Drivers](#)
- [Miscellaneous Export Notes](#)

Exporting a Design to Generated CMSIS-Pack

Use PSoC Creator to develop a PSoC hardware design. Then use either PSoC Creator or Keil's µVision IDE for firmware development. To use the Generated CMSIS Pack, you must [build the design in PSoC Creator](#) and then use the Export to IDE feature.

Note PSoC 3 devices are not supported to be exported using this feature and can be exported through Export to µVision feature. See Exporting a PSoC 3 Design to Keil µVision IDE.

Note The PSoC Creator export process generates a Generated CMSIS Pack specific for the PSoC Creator design being exported. You can define information for this pack during the export process. Also as part of the export, a file with the same name of the project and .gpdsc extension will be created. This is used by µVision to detect the exported software pack for the design.

Note When building a project in µVision 5 using Arm MDK toolchain, all output is written into the UVBuild directory found in your project directory.

The build and export process is the same for new projects or updated projects.

Note The Export to IDE wizard does not apply to PSoC 6 devices. For PSoC 6 devices, use the [Target IDEs](#) section of the [Build Settings](#) dialog to choose third party IDEs for which to generate files.



Changing Devices

Creating and exporting a project using µVision and then changing the device and re-exporting is not supported. If you wish to do this, you must manually delete your µVision project (*.uvproj *.uvprojx) before re-exporting. Then, follow the appropriate instructions for the new PSoC device and version of µVision

Exporting a New PSoC Creator Design:

The initial flow is to create a design in PSoC Creator and export the design to the µVision IDE.

1. Create your design in PSoC Creator in the usual manner.
2. Make sure that µVision 5 is closed when you perform the export to µVision 5 generated software pack. If not you have to re-start µVision after Export to refresh the generated pack. Use the **Project > Export to IDE** menu option to open the IDE Export Wizard dialog.
3. If you have not built the design, you will be prompted to build at this time.



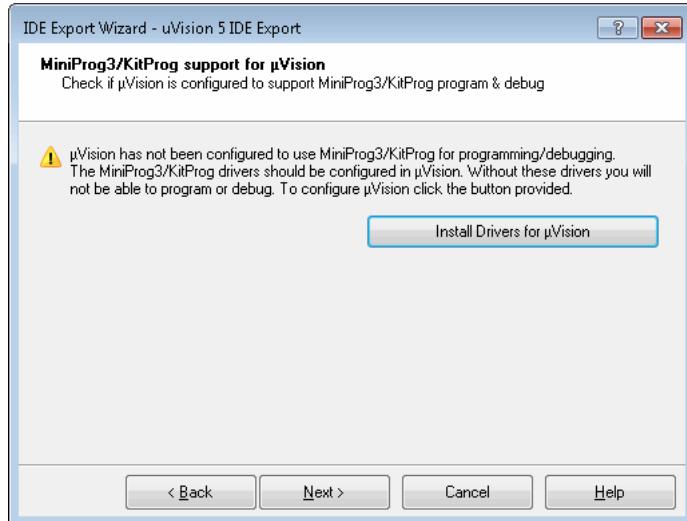
4. Click **Yes** to build. If you click **No**, the export process will be cancelled.

After a successful build, the IDE Export Wizard opens.



5. If not already selected, select the Generated CMSIS-Pack option. Click **Next >** to continue.

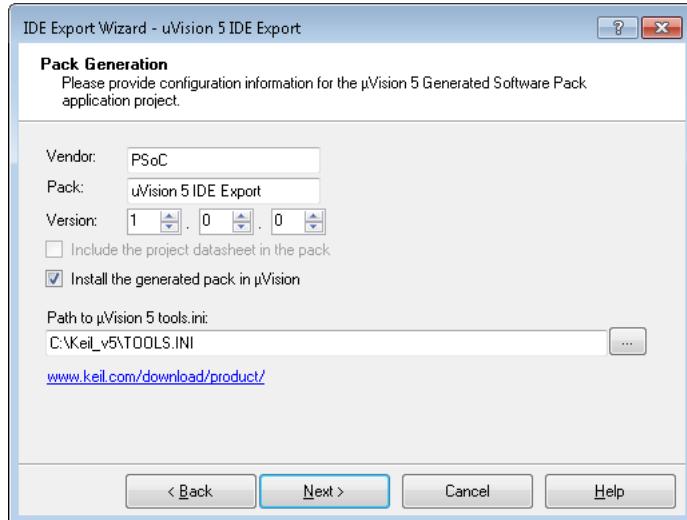
PSoC Creator checks whether MiniProg3/KitProg drivers have been registered with µVision. If they are not registered, the MiniProg3/KitProg support for µVision step opens. However, if PSoC Creator locates a µVision installation with MiniProg3/KitProg support enabled, then it will skip this step.



Note PSoC Creator only examines the first µVision installation found to see if it has MiniProg3/KitProg drivers properly registered. If you have multiple copies of µVision installed on your computer, the auto-detection process may not be accurate. If you are sure you already have drivers registered, you may skip this step of the wizard.

6. If you need to register the drivers, click **Install Drivers for µVision** and follow the prompts on the installation wizard. This process is only required once. See [Registering MiniProg3/KitProg Drivers](#) for more information.
7. If you need to register drivers with another µVision installation, go to the PSoC Creator **Tools** menu and select **Install drivers for µVision** to launch the installation wizard.
8. Click **Next >** to continue.

The Pack Generation step opens.



9. Enter the appropriate information, as follows:

- Vendor:** Enter the Vendor name shown in the µVision 5 software pack.

- Pack:** Enter the Project name of the µVision 5 Pack.

The pack name will be combined with the device part number to present the device name in the µVision environment. µVision restricts the device name to be a maximum of 48 characters. So, the pack name must be short enough to be combined with the other characters to make total number of characters equal to 48 or less.

- Version:** Enter the major version, minor version, and revision number for the software pack being exported.
- Include the project datasheet in the pack:** If applicable, select this check box to include the project datasheet.

If the [project datasheet](#) is generated, this option is available. To add a project datasheet to the pack:

- Select Build > Generate Project Datasheet.
- Then, build the project and perform the export.

- Install the generated pack in µVision:** Select this check box to install the generated pack that will be exported to µVision.

Note If you deselect this option, the pack will not be installed automatically. You will have to install it manually. Refer to µVision documentation.

If you do not have µVision 5 installed, deselect this option to perform the export and generate the pack without installing it.

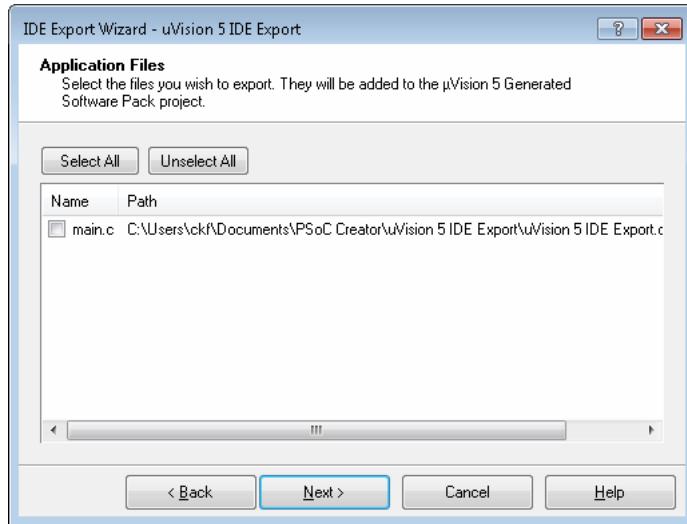
Optionally, you can download µVision 5 from the link provided.

- Path to µVision 5 tools ini:** This shows the default path to the *TOOLS.INI* file in the µVision 5 installation directory.

If you would like to export to another installed version of µVision 5 IDE on your computer, click the ellipsis [...] button and navigate to the appropriate directory.

10. Click **Next >** to continue.

The Application Files step opens.



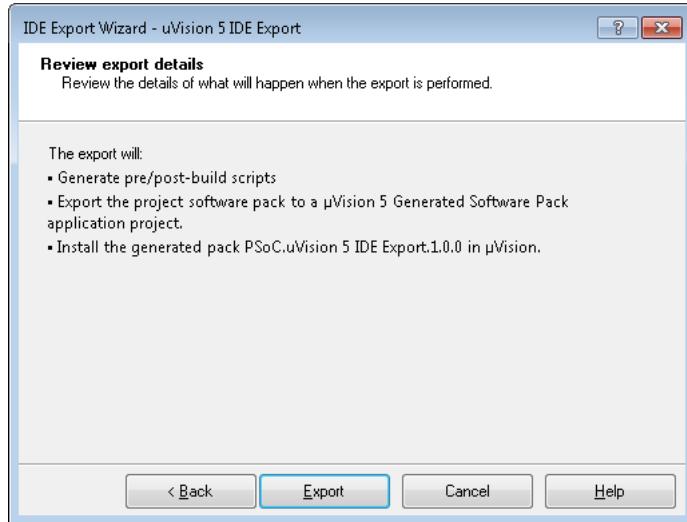
- Select the files you want to add to your new µVision application project.

Note The wizard does not copy these files to the pack when exporting. It simply adds references to the existing files to the µVision application project.

- To start with an empty µVision application project, click the **Unselect All** button.
- Once the initial export is complete, build settings and file management must be performed within the µVision environment. For example, if you want to add a new source file, select **File > New** in µVision.

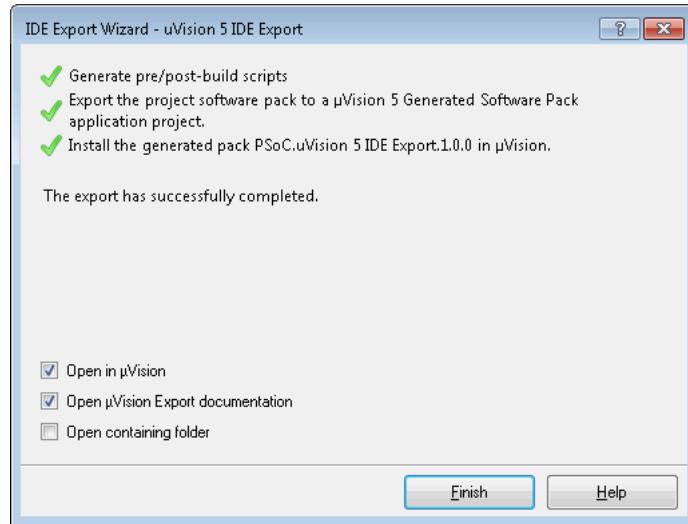
11. Click **Next >** to continue.

The Review export details step opens.



12. Review the export details and click **Export**.

The final step opens showing the completed export.



Optionally select the following action(s) when the export process completes:

- Open in μVision
This option is available if you selected the **Install the generated pack in μVision** check box on the Pack Generation step.
- Open μVision Export documentation
- Open the folder containing the project files

13. Click **Finish** to close the wizard.

Note Refer to the Notes for μVision 5 listed in [Miscellaneous Export Notes](#).

See Also:

- [Key IDE Export Files/Projects](#)
- [Opening Generated CMSIS-Pack Projects \(μVision 5 IDE\)](#)
- [Registering MiniProg3/KitProg Drivers](#)
- [Miscellaneous Export Notes](#)

Exporting a Design to Makefile

The GNU Make is a utility that uses a Makefile to automatically determine which pieces of a large program need to be recompiled, and then issues commands to recompile those pieces. The Makefile tells the Make utility what to do and how to compile and link a program. Most IDEs support a simple Makefile-based build option. The PSoC Creator Export to Makefile feature allows you to build PSoC designs in those IDEs, as well as from the command-line.

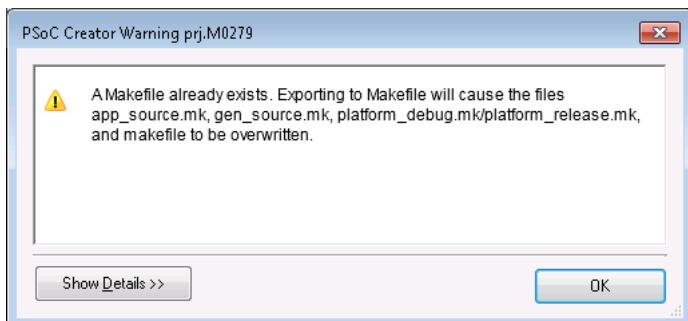
Note For PSoC 6 devices, use the [Target IDEs](#) section of the [Build Settings](#) dialog, and enable the **Makefile** option.

To Export to Makefile:

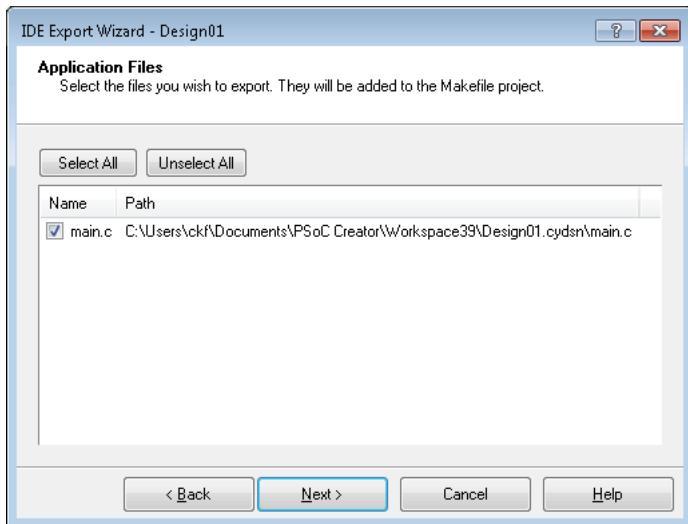
To export your PSoC Creator design to a Makefile, select the "Makefile" option on the IDE Export Wizard dialog and click **Next >**.



Note If you previously used this feature, PSoC Creator will re-create and overwrite the Makefile and .mk files, but will prompt you to confirm the action.



The Application Files step opens.

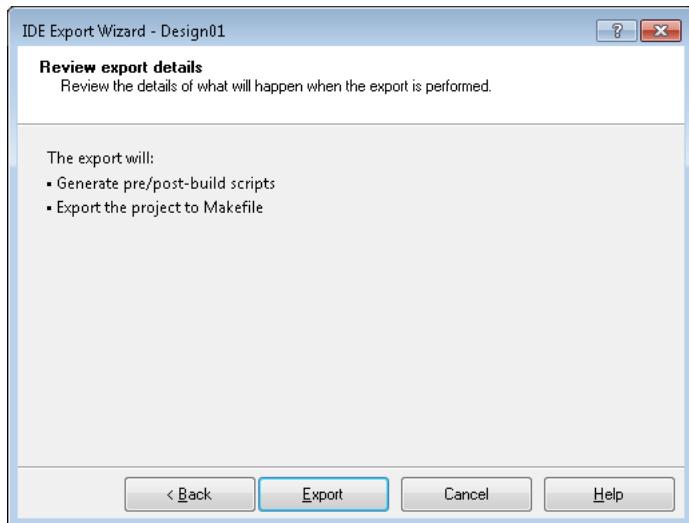


Select which non-generated code to export to the project.

Note Only the files that are compatible with the selected toolchain will appear in the dialog. The wizard adds references to the selected files to *app_source.mk*.

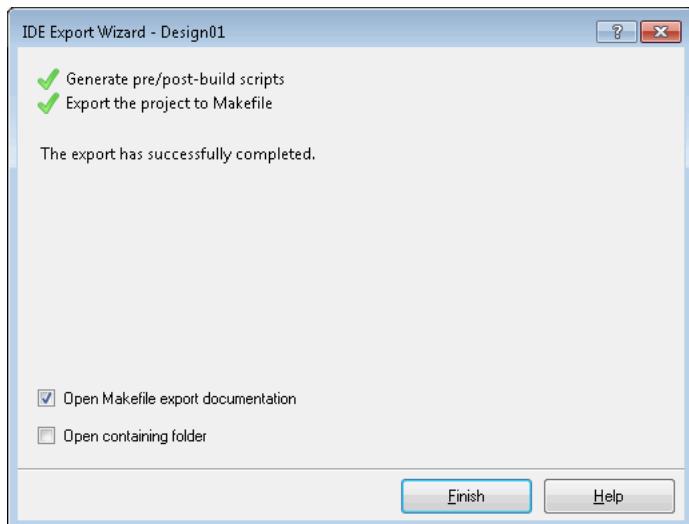
Click **Next >** to continue.

The Review export details step opens.



Review the export details and click **Export**.

The final step opens showing the completed export.



Click **Finish** to close the wizard.

Using PSoC 4/PSoC 5LP Designs with 3rd Party IDEs

This section contains the following topics:

- [PSoC 4/PSoC 5LP Eclipse Information](#)
- [Setting up a PSoC 4/PSoC 5LP IAR Project](#)
- [Opening PSoC 4/PSoC 5LP Projects in µVision IDE](#)
- [Opening Generated CMSIS-Pack Projects \(µVision 5 IDE\)](#)
- [Opening PSoC Creator Designs in Makefile](#)
- [Setting Up for Segger J-Link/J-Trace Debugger for PSoC 5LP](#)
- [Setting Up for ULink2/ULink Pro and Segger J-Link Debugger Probes](#)

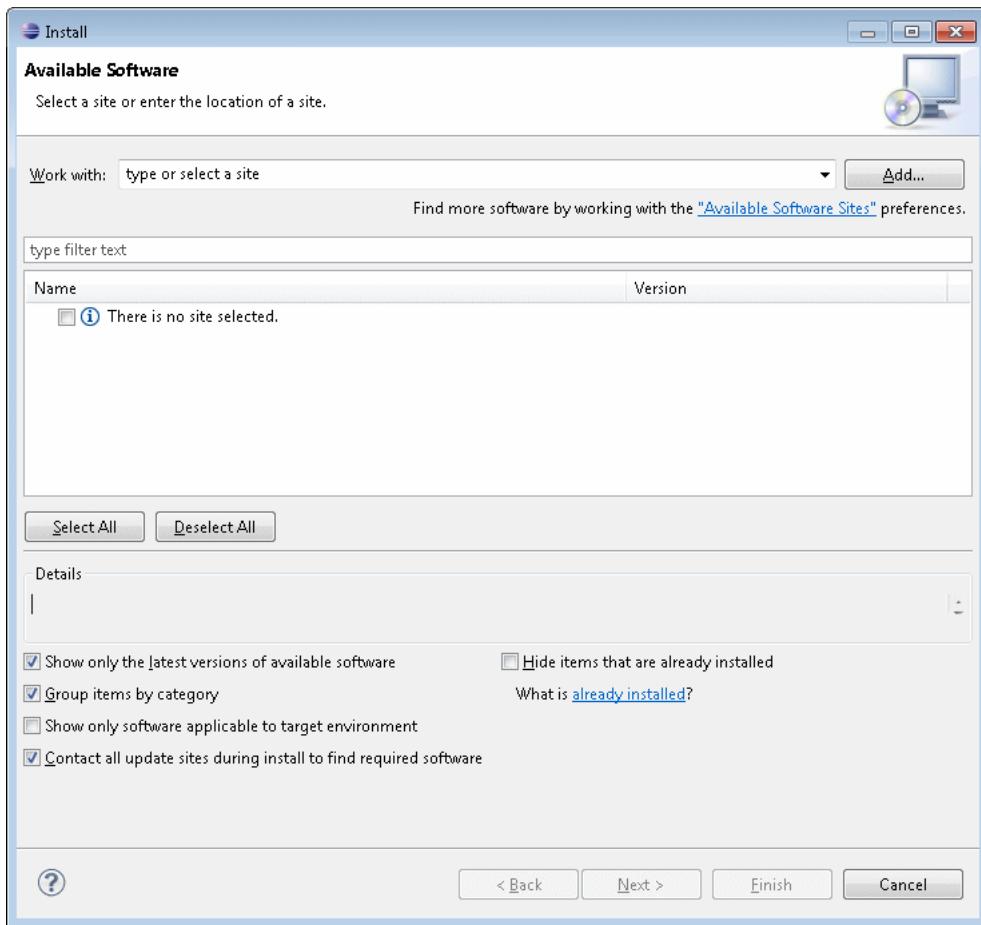
PSoC 4/PSoC 5LP Eclipse Information

Eclipse Installation Configuration

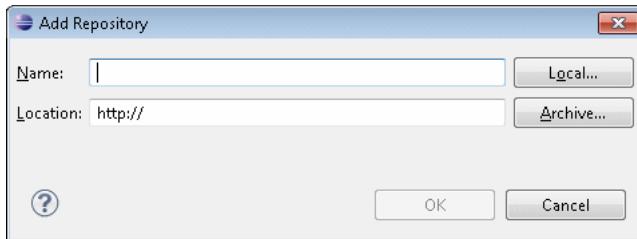
- When using the Eclipse IDE, you do not need a full PSoC Creator installation during firmware development.
- You must have a Java JRE or JDK installed (Java version 7 or higher).
- You must have a GCC Arm compiler installed. The GNU Tools Arm Embedded toolset is available at <https://launchpad.net/gcc-arm-embedded>.
- You must install Eclipse CDT from <http://www.eclipse.org/cdt>. Look for the "Eclipse C/C++ IDE" download package. The Eclipse Luna and later releases support the Cypress-provided Eclipse feature.
 - If you are running a 32-bit operating system, download the 32-bit version of Eclipse.
 - If you are running a 64-bit operating system, you should download the version of Eclipse that matches the Java runtime version that you have installed; that is, 32-bit Eclipse for 32-bit Java or 64-bit Eclipse for 64-bit Java.
- You must also have the new Cypress-provided Eclipse feature (`com.cypress.psoccreatorimport`) installed, which provides the following functionality:
 - Speeds the creation of an Eclipse project associated with a PSoC Creator design with a new PSoC Creator project type and pre-populating tool chain build options.
 - Provides project resource synchronization between PSoC Creator and Eclipse. Changes made in the set of files that constitute a project in PSoC Creator are reflected in subsequent builds in Eclipse.

Download this Cypress-provided Eclipse feature from www.cypress.com/go/creator/eclipseimportdownload, and install the downloaded zip file as follows:

- In Eclipse, select **Help > Install New Software...** to open the Install dialog.



- Check the Contact all update sites during install to find required software check box in the Details section of this dialog.
- Select the **Add...** button at the top of the dialog to open the Add Repository dialog.



- Click **Archive...** and browse to the location where you saved the Cypress-provided Eclipse plugin zip file that you downloaded above. Select the file and click **Open**; then click **OK**.
- Back on the Install dialog, select the check box next to "PSoC Creator Import Feature" and then click **Next >**.
- The next wizard page summarizes the features you are about to install. Click **Next >** and read and accept the license for this feature.
- Click **Finish** to install the plugin. There might be a warning about the software not being signed; click

OK.

- Restart Eclipse when prompted.
- In order to use the Segger J-Link debug probe, you must download the Segger J-Link toolset from <http://www.segger.com/jlink-software.html>. You will need release 5.12 or later, which is required for PSoC 4000S flash support.

Import into Eclipse

Create New Eclipse CDT Firmware Application Project

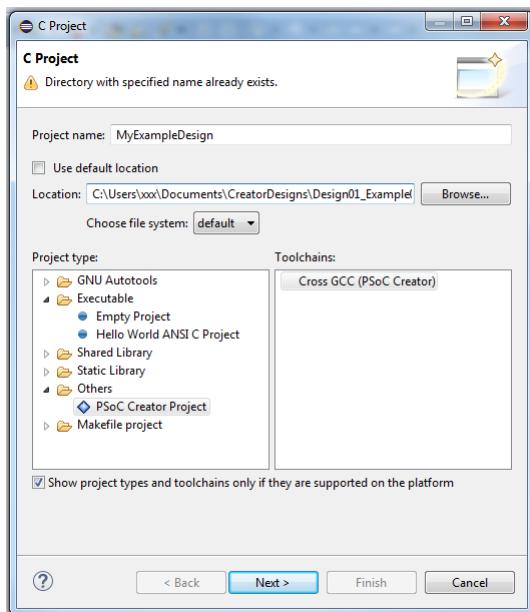
Create an Eclipse CDT executable project for your firmware. This must be created as a project rooted not in the Eclipse workspace, but instead sharing the source code folder of your PSoC Creator design.

Note When you start Eclipse, you will create a new workspace or select an existing one. You must choose one that does not include the folder where your PSoC Creator design is located. This is an Eclipse limitation.

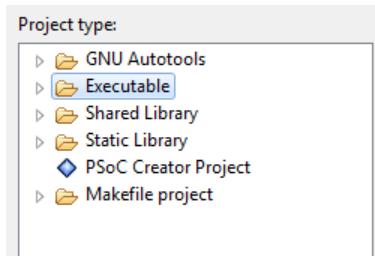
1. Create a project via the **File > New > C Project** menu entry.

2. On the wizard's C Project page:

- Type a **Project name**.
- Unselect the **Use default location** check box, and browse to and select your PSoC Creator project's top-level folder (the one ending in .cydsn). This allows PSoC Creator and Eclipse to share the same source files.
- Under **Project Type > Others**, select "PSoC Creator Project."
- Under **Toolchains**, select "Cross GCC (for PSoC Creator)."



Note In Eclipse releases earlier than the Luna release, the **Others** category does not exist. The "PSoC Creator Project" entry can be found in the Project Type list as shown.



3. Click **Next** to move to the Select Configurations wizard page. No changes are needed here. Click **Next** again.

4. On the **Cross GCC Command** wizard page:

- Prefix will have a default value of: arm-none-eabi-
- Set path by navigating to the bin folder of your compiler's install directory, where you can find *arm-none-eabi-gcc.exe*. This is typically found using the following path:

<Arm_tools_install_path>/bin

(Look for *arm-none-eabi-gcc.exe* and not *gcc.exe*.)

Note You can use the Arm GCC installation included in PSoC Creator, if you have PSoC Creator installed.

- In PSoC Creator 3.1 and earlier, this is found at:

<PSoC_Creator_install_path>/import/gnu_cs/arm/<version>/bin

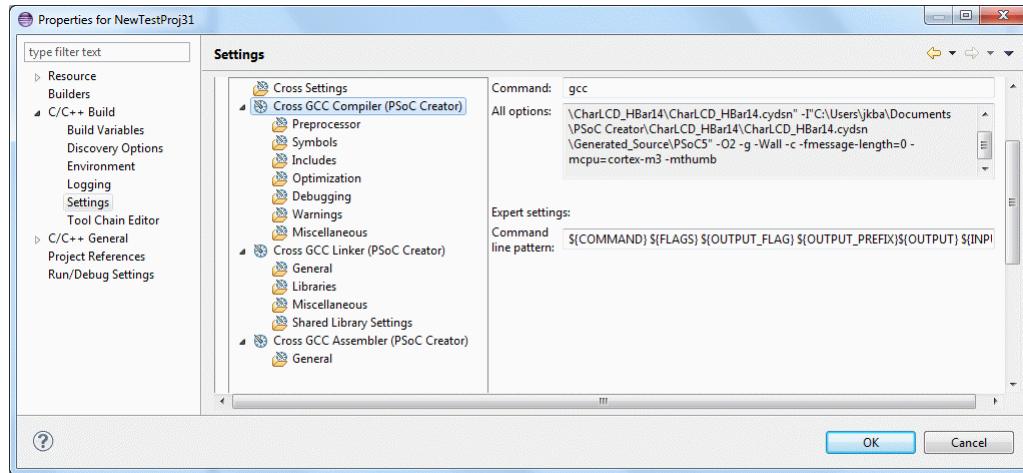
- In PSoC Creator 3.2 and later, this is found at:

<PSoC_Creator_install_path>/import/gnu/arm/<version>/bin

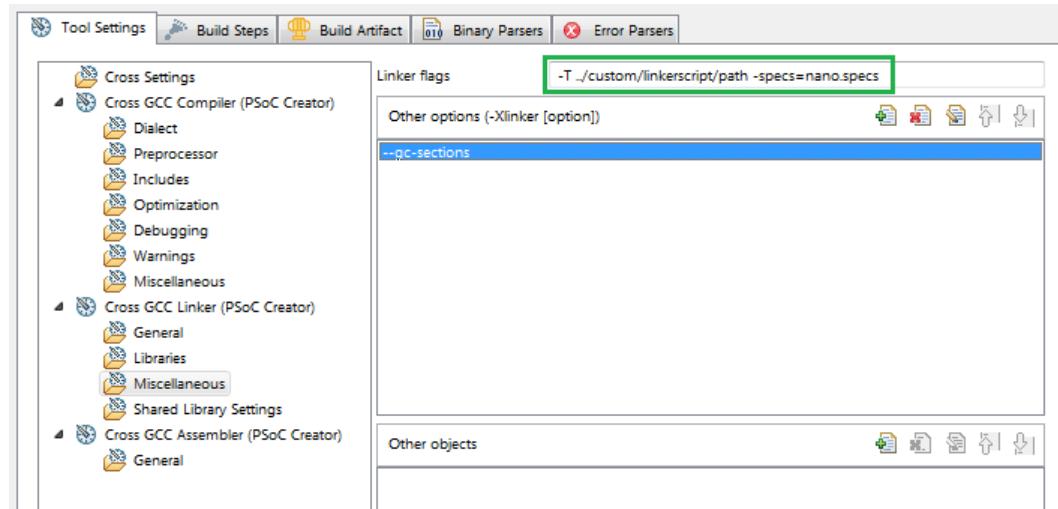
The newly created PSoC Creator project in Eclipse includes toolchain settings, accessible in the Project Explorer pane.

1. Right-click the project name and select **Properties > C/C++ Build > Settings**.
2. Then select the **Tool Settings** tab.

These settings are driven by the project's PSoC device. Certain options will be different depending on whether the project is for a PSoC 4 or PSoC 5LP device. (The pre-populated option settings are listed in the Project Build Settings section at the end of this document in case you decide to change them and later want to restore these values. These values are set by default; you typically do not need to modify them for your project.)



Note If you are using custom toolchain options, add those options manually to the tool setting in Eclipse. For example, if you are using custom linker scripts in the project, modify the **Linker Flags** option as shown in the following image:



Building with Cygwin/make Installed

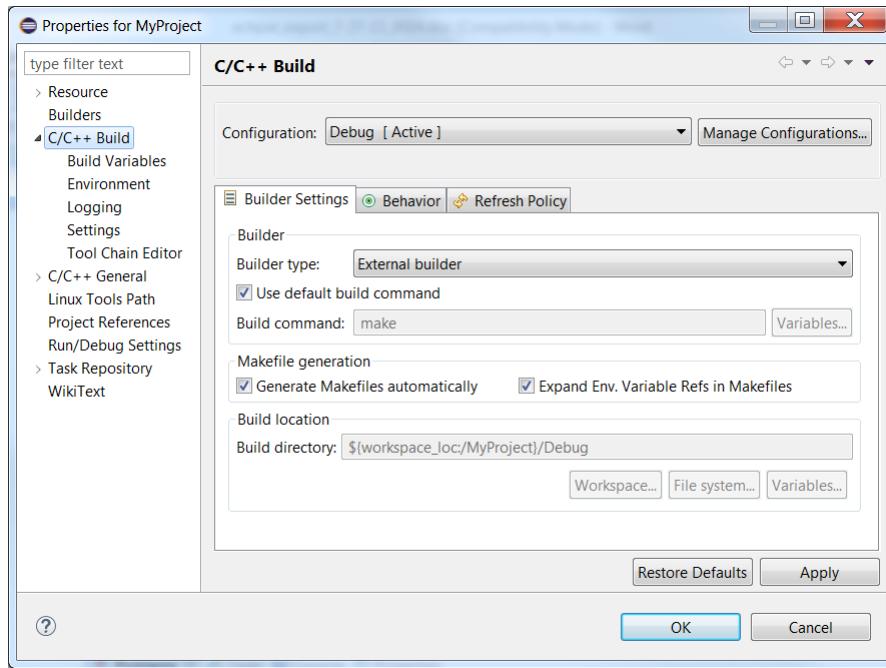
As of this feature version v2.2, designs are imported with the project builder set to the Eclipse Internal Builder. This means that you do not need the 'make' utility installed to build your design in Eclipse.

If you wish to use the 'make' utility, you will need to use the Eclipse External Builder. In this case, the Eclipse CDT Builder assumes that the 'make' utility is in your PATH, typically obtained through a cygwin installation on your machine. If using the Eclipse external builder, you'll need to have something like "C:\cygwin\bin" or C:\cygwin64\bin" in your Windows PATH environment variable or in the Eclipse environment settings.

Without 'make', you need to adjust your project build settings to use the Eclipse internal builder instead.

1. Right-click the project name and select **Properties > C/C++ Build**.

2. Select the **Builder Settings** tab.
3. Change the **Builder type** value to External builder. Click the **OK** button.



Note For more complex designs, Make files may be generated with absolute paths for files outside of the current project. This can expose a bug in GNU Make versions later than 3.8.0 and earlier than 4.0. You may see an error like: "Multiple targets. Stop.". The solution is to use the Eclipse Internal Builder, or to change your GNU Make version to one that does not contain this bug.

Design Iteration between PSoC Creator and Eclipse

Users are directed to create their Eclipse project using their PSoC Creator design folder. Users simply need to rebuild their design in PSoC Creator and run the IDE Export wizard again. The Cypress-provided plugin detects the build update to the PSoC Creator project and makes the corresponding change in the visible Eclipse project files. A corresponding check and update is performed when Eclipse is started as well.

Note that this synchronization support is one-way: changes in PSoC Creator can be seen in Eclipse, but file additions and deletions made in Eclipse will not be seen by PSoC Creator. This allows changes in the hardware configuration made upstream in PSoC Creator to be seen downstream in Eclipse for continued firmware development.

Change in Project Build Settings

Project build settings are not exported from PSoC Creator to Eclipse; any custom build settings that you add or change in PSoC Creator must be similarly added/changed in Eclipse, following the export step. As a starting point, the default build settings in PSoC Creator are also set as the default build settings on initial export of a project to Eclipse. You can then change these build settings as needed inside Eclipse. These default build settings are listed below for reference:

Under Cross GCC Compiler:

- Under **Includes**, these project include paths are provided as default values:
 - .." (i.e., the project design folder)

- "../Generated_Source/\${GEN_ARCH}"
- Under **Optimization**, this flag must be set:
 - ffunction-sections
- Under **Miscellaneous, Other flags**, these Arm-related flags default values are provided via the \${PROC_FLAGS} variable setting:
 - For PSoC 5LP: -mcpu=cortex-m3 -mthumb
 - For PSoC 4: -mcpu=cortex-m0 -mthumb
 - For PSoC 4000S/PSoC 4100S/PSoC 4100S Plus/PSoC Analog Coprocessor: -mcpu=cortex-m0plus -mthumb
- Under **Miscellaneous, Other flags**, these flags must be set:
 - c -fmessage-length=0

Under Cross GCC Assembler:

- Under **Includes**, these project include paths are provided as default values:
 - ".." (i.e., the project design folder)
 - "../Generated_Source/\${GEN_ARCH}"
- Under **General, Other flags**, these Arm-related flags default values are provided via the \${PROC_FLAGS} variable setting:
 - For PSoC 5LP: -mcpu=cortex-m3 -mthumb
 - For PSoC 4: -mcpu=cortex-m0 -mthumb
 - For PSoC 4000S/PSoC 4100S/PSoC 4100S Plus/PSoC Analog Coprocessor: -mcpu=cortex-m0plus -mthumb

Under Cross GCC Linker:

- Under **Libraries, Library Search Path**, this library path is provided:
 - L "../Generated_Source/\${GEN_ARCH}"
- Under **Miscellaneous, Linker flags**, these flag values are provided:
 - T ../Generated_Source/\${GEN_ARCH}/\${LINKER_FILE} -specs=nano.specs
- Under **Miscellaneous, Linker flags**, these Arm-related flags default values are provided via the \${PROC_LINK_FLAGS} variable setting:
 - For PSoC 5LP: -mcpu=cortex-m3 -mthumb -mfix-cortex-m3-lldr
 - For PSoC 4: -mcpu=cortex-m0 -mthumb
 - For PSoC 4000S/PSoC 4100S/PSoC 4100S Plus/PSoC Analog Coprocessor: -mcpu=cortex-m0plus -mthumb
- Under **Miscellaneous, Linker flags**, this default value is provided: -Xlinker --gc-sections

- Under **Other Objects**, the following object path must appear:
 - "\${EXPORT_FOLDER_PATH}/\${TOOL_NAME}/CyComponentLibrary.a"
- The link step command line is structured so that "-Wl,--start-group" and "-Wl,--end-group" options surround all input files and libraries, providing multiple linker passes to resolve cyclic references between libraries, if needed.

On the Build Steps tab:

The following default settings are provided:

- Pre-build step should read:
"../\${EXPORT_FOLDER_PATH}/prebuild.bat"
- Post-build step should read:
"../\${EXPORT_FOLDER_PATH}/postbuild.bat"
"\${POSTBUILD_HEXFILE_PATH}/\${ConfigName}/\${ProjName}"

Build Customization

Project Customization

- **Compiler optimization settings** – Other optimizations that might be useful in reducing your generated flash image size include:

Under Cross GCC Compiler:

- Under **Optimizations**, set Optimization level to Optimize for size (-Os)
- **Linker script changes** – A default GCC linker script is located at:
 - For PSoC 5LP: Generated_Source/PSoC5/cm3gcc.ld
 - For PSoC 4: Generated_Source/PSoC4/cm0gcc.ld
 - For PSoC 4000S/PSoC 4100S/PSoC 4100S Plus/PSoC Analog Coprocessor: Generated_Source/PSoC4/cm0plusgcc.ld

If you need to customize the link step for your executable image, do the following:

1. Copy the file from the above location to the top level folder of your Eclipse project. This will prevent any subsequent builds in PSoC Creator from overwriting your changes to this file.
2. Edit your new copy of the file as needed.
3. Change the Build Settings linker flags (under **Cross GCC Linker, Miscellaneous, Linker flags**) to be:
 - For PSoC 5LP: -T ..//cm3gcc.ld
 - For PSoC 4: -T ..//cm0gcc.ld
 - For PSoC 4000S/PSoC 4100S/PSoC 4100S Plus/PSoC Analog Coprocessor: -T ..//cm0plusgcc.ld

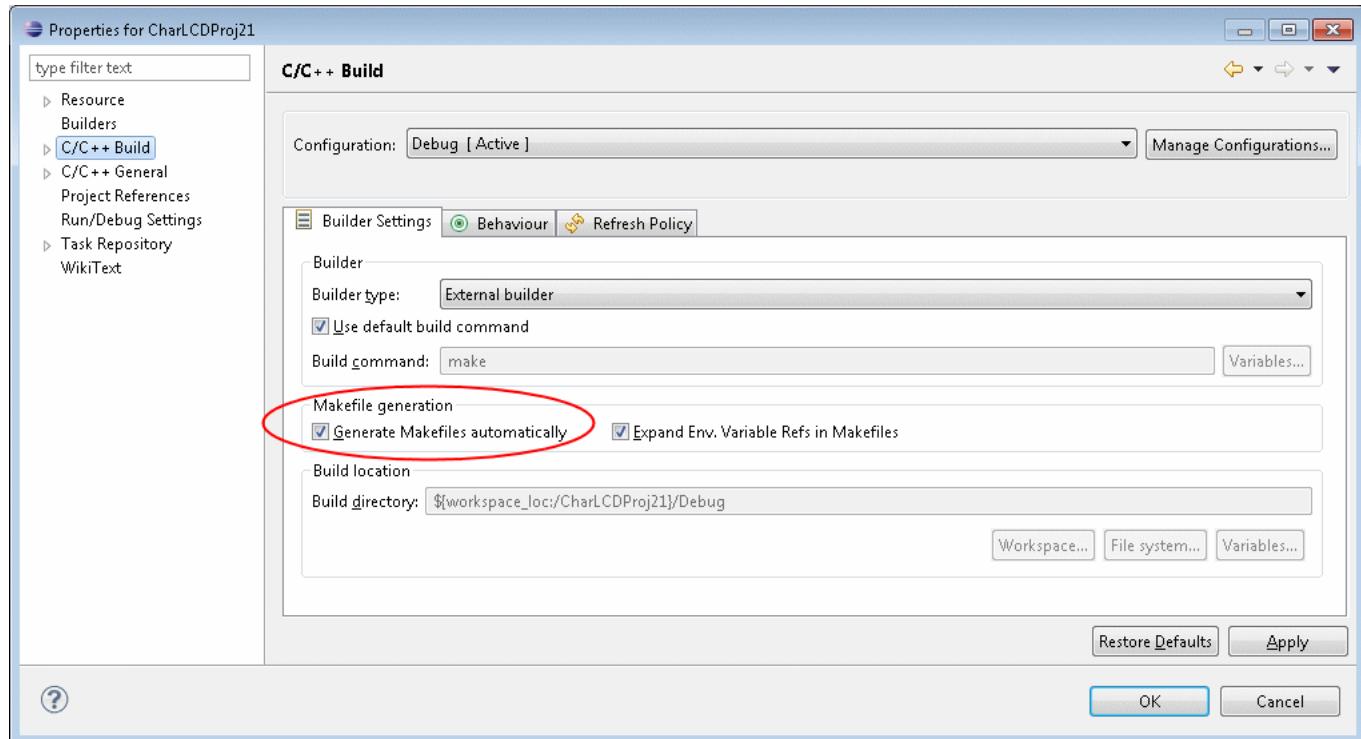
Build Customization

If your design requires specialized steps in order to build to completion (beyond the standard pre-build/assemble/compile/link/post-build stages of the standard Eclipse CDT Managed Build System [MBS]), you may want to

consider managing your own build for the project. Perhaps the easiest way to do this is to turn off the MBS automatic generation of Makefiles with every project build. You can then customize the already generated Makefiles to complete your project build as needed.

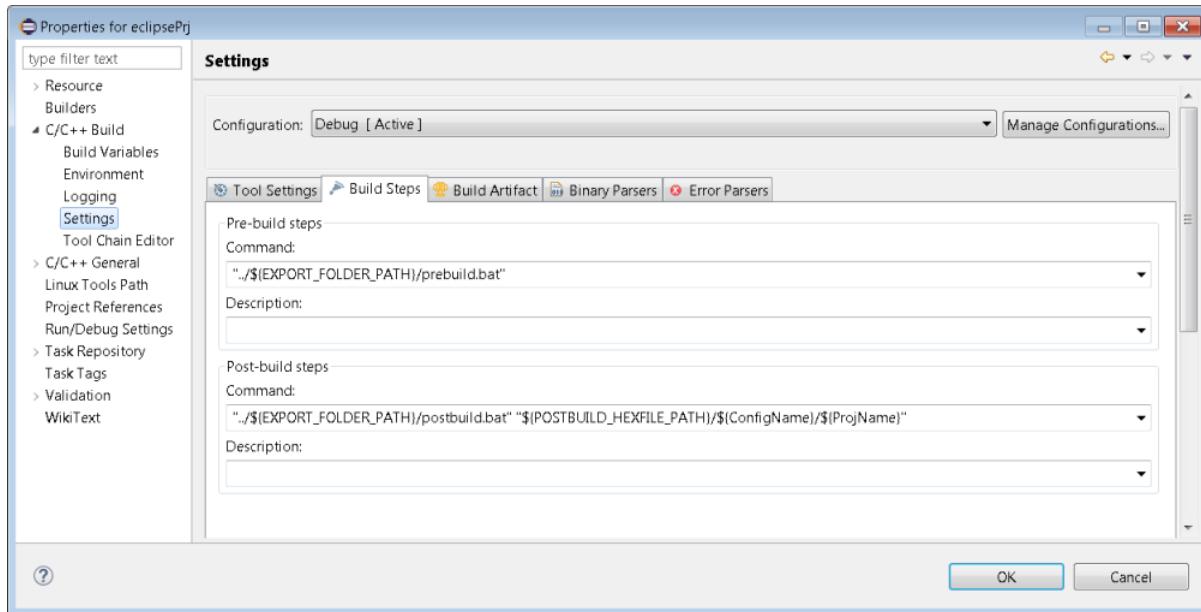
To turn off the automatic generation of Makefiles for your project, do the following:

- In the Eclipse Project Explorer, right-click on your project name and select **Properties...**
- Select **C/C++ Build** in the left hand pane
- Uncheck the Generate Makefiles automatically check box



User Commands

PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. For Eclipse, you can edit your project's settings (under the project's **Properties > C/C++ Build > Settings**) on the Build Steps tab. PSoC Creator provides a call to our generated pre-build and post-build scripts where needed; you can add your own commands, separating all commands with semicolons.



Flashing and Debugging in Eclipse

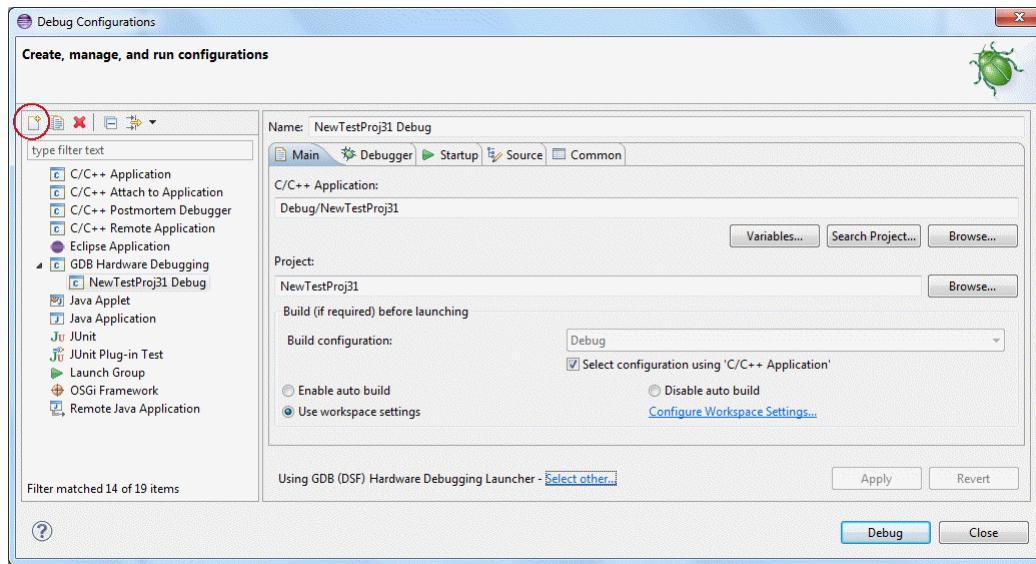
In order to use a J-Link or J-Trace debug probe, you will first need to download and install Segger's software and documentation pack from their web site (<https://www.segger.com/jlink-software.html>). This package provides USB drivers for their probes, several utilities, and a GDB server, which connects the Eclipse debugger to your Cypress hardware.

Debug Flow within Eclipse

Follow these steps to flash and debug an imported PSoC Creator project in Eclipse/CDT using a Segger J-Link probe.

- Provide power to your PSoC hardware, then connect a J-Link debug probe to your PSoC hardware and host machine. Run J-Link Commander.exe from the All Programs->SEGGER->J-Link Arm menu. This initializes the J-Link probe and only needs to be done once following each J-Link probe connection. (Without this step, the J-Link GDB Server cannot successfully connect to the target.) Verify the following before exiting the application:
 - Target device is set to your PSoC device.
 - Target interface is set to JTAG (PSoC 5LP only), or SWD (PSoC 4 or PSoC 5LP)
 - Type 'exit' at the tool's command prompt to exit the application.
- In Eclipse, create a new Debug Launch configuration for your design. Use the menu selection **Run > Debug Configurations...**

3. Create a new debug launch configuration by selecting the "GDB Hardware Debugging" category and clicking the **New** button (circled in the following figure).



4. Enter the debug launch configuration settings as follows:

Main tab:

- Provide path to the design's executable file produced by the project build. Navigate to the project's Release or Debug folder to find it, or simply click the **Search Project...** button and select the executable.
- The debugger launcher should be set to "GDB (DSF) Hardware Debugging Launcher."

Note However, for the Eclipse Kepler release, it is suggested that users select the "Legacy GDB Hardware Debugging Launcher." (This is found at the bottom of the **Main** tab, change using the "Select other..." link.)

Debugger tab:

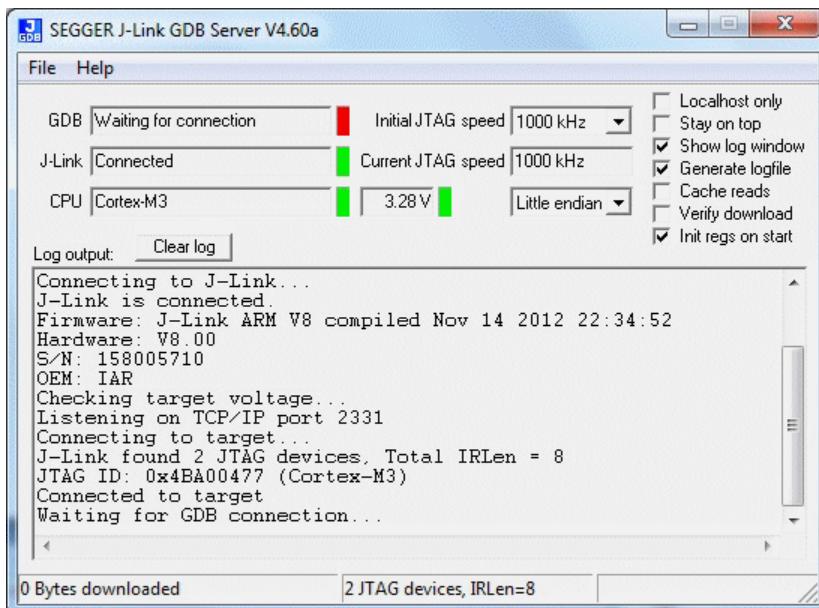
- Set the path to the GDB command by browsing to the GDB executable to use. Typically, this will be named arm-none-eabi-gdb.exe, located in the same folder as the "Cross compiler path" setting provided when [importing into Eclipse](#).
- "Use remote target" is checked
- JTag device is set to "Generic TCP/IP"
- Host name is "localhost" and port is "2331" (the default J-Link GDB server port).

Startup tab:

- Reset and Delay set to 3 seconds
- Halt is checked
- Enter "monitor reset" in the Init command field:
- Load Image and Load Symbols sections must have "Use project binary" selected.
- Runtime options: "Resume" box is checked

- Press Apply and Close to save this debug launch configuration for later use.
5. Launch the Segger J-Link GDB server by running "J-Link GDB Server" via All Programs->SEGGER->J-Link Arm menu. Verify the following are set:
- Connection to J-Link is set to USB
 - Target interface is set to JTAG or SWD as needed.
 - Target device is set to the Cypress PSoC device you are using. Press the "..." button to browse the supported devices. Select Cypress in the Manufacturers pull-down to list only Cypress devices. Select your device family and press OK.
 - OK the main dialog window.

The server should connect to the target at this point and indicate it is waiting for a GDB connection, as shown in figure below.



6. In Eclipse, set any needed breakpoints in your code. Run the Debug launch configuration you created in step 2 above by using the **Run > Debug Configuration...** menu selection to locate your debug launch configuration from the ones under the GDB Hardware Debugging heading. Click the **Debug** button to start the debug session. The GDB Server will display additional output when the Eclipse GDB session begins.
7. When your breakpoint is encountered, Eclipse will change to its debug perspective and halt. Normal Eclipse debug functionality (breakpoint manipulation, examining/changing variables and memory, etc) is available at this point.

Note If you experience problems with the PC not tracking as you would expect while stepping through code, you may be able to correct this by turning off compiler optimization in your code while debugging, as follows:

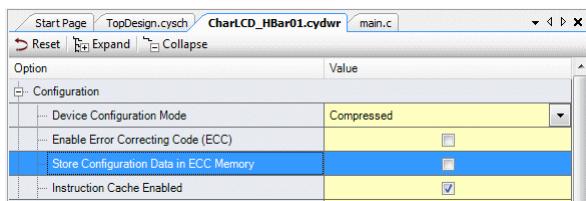
1. Right-click the project name and select Properties > C/C++ Build.
2. Select Settings.
3. Select Cross GCC Compiler (PSoC Creator).
4. Select Optimization.

5. Change the optimization level to "None".
6. Rebuild your design and start debugging again.

Flashing PSoC 5LP Designs using the J-Link Probe

If you are exporting your design to Eclipse for a PSoC 5LP design, it is important that you understand the following limitations. These limitations will be removed in a future release of the Segger J-Link flashloader for PSoC 5LP devices.

- Before building and exporting a PSoC 5LP based design to Eclipse, go to your project's Design Wide Resources System page and uncheck the "Store Configuration Data in ECC" check box. Storing configuration data in ECC memory is the default setting for PSoC Creator projects, but must be changed for designs exported to Eclipse in order for them to execute properly once flashed.



- Your design cannot use EEPROM memory with any initial values set from the PSoC Creator EEPROM Editor. The Segger-provided PSoC 5LP flashloader does not yet program this memory space.
- Your design cannot use ECC memory, as the Segger-provided PSoC 5LP flashloader does not yet program this memory space. Your project's [Design-Wide Resources \(DWR\) System Editor](#) settings should have the "Enable Error Correcting Code (ECC)" box unchecked, as shown above.
- The Segger-provided PSoC 5LP flashloader does not yet program the NVL memory space. This means the following settings, if changed in your design in PSoC Creator, must be programmed by building and downloading your design first in PSoC Creator with a Cypress MiniProg3 probe. You can then export your design to Eclipse and program via a J-Link probe:
 - DWR System setting for Enable Fast IMO During Startup
 - DWR System setting for Enable Device Protection
 - DWR System setting for Debug Select
 - DWR System setting for Use Optional XRES
 - Initial state settings on pins

Setting up a PSoC 4/PSoC 5LP IAR Project

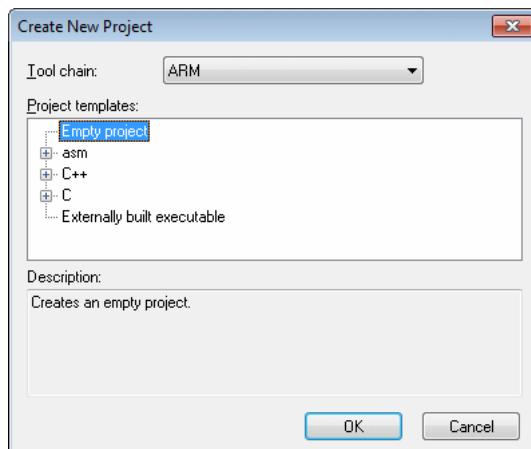
After exporting the PSoC Creator project for use in the IAR IDE, you need to follow a set of steps to set up the project. This is because IAR does not provide a means of exporting some of the important information needed to set up the project, such as the linker file to use, tool chain command line settings, or a way to add third party libraries to the project's linker additional library list.

The steps to use are as follows:

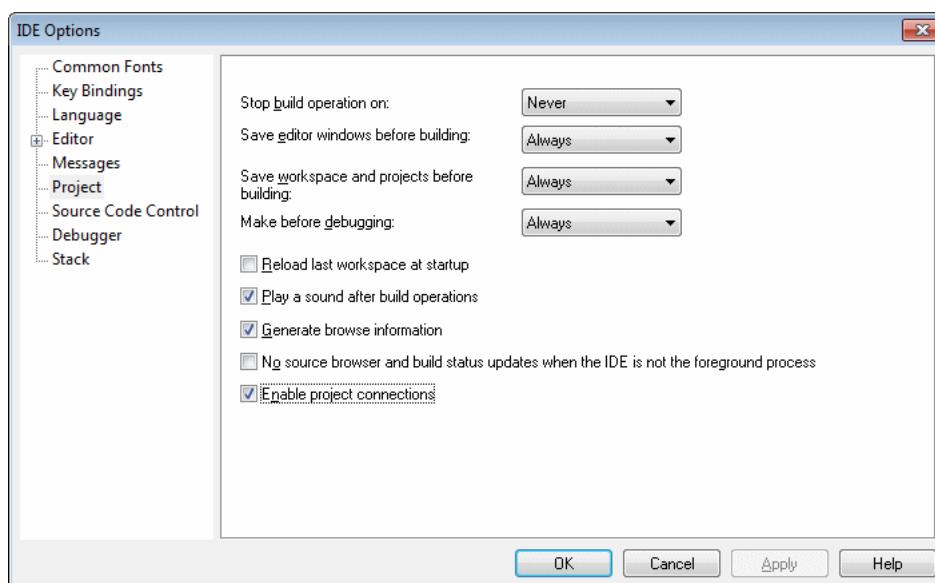
1. [Export your PSoC Creator project to IAR using the Export Wizard.](#)

Note If you do not have IAR installed, PSoC Creator performs the export with the assumption that IAR version 7.10.3 will be used to create an IAR project, based on the exported design. If IAR is installed, PSoC Creator will use that version of IAR.

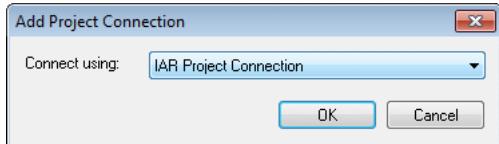
2. Launch IAR Embedded Workbench for Arm (EWArm).
3. Create a new Empty project (**Project > Create New Project**), and save the project in the .cydsn directory of your exported PSoC Creator project.



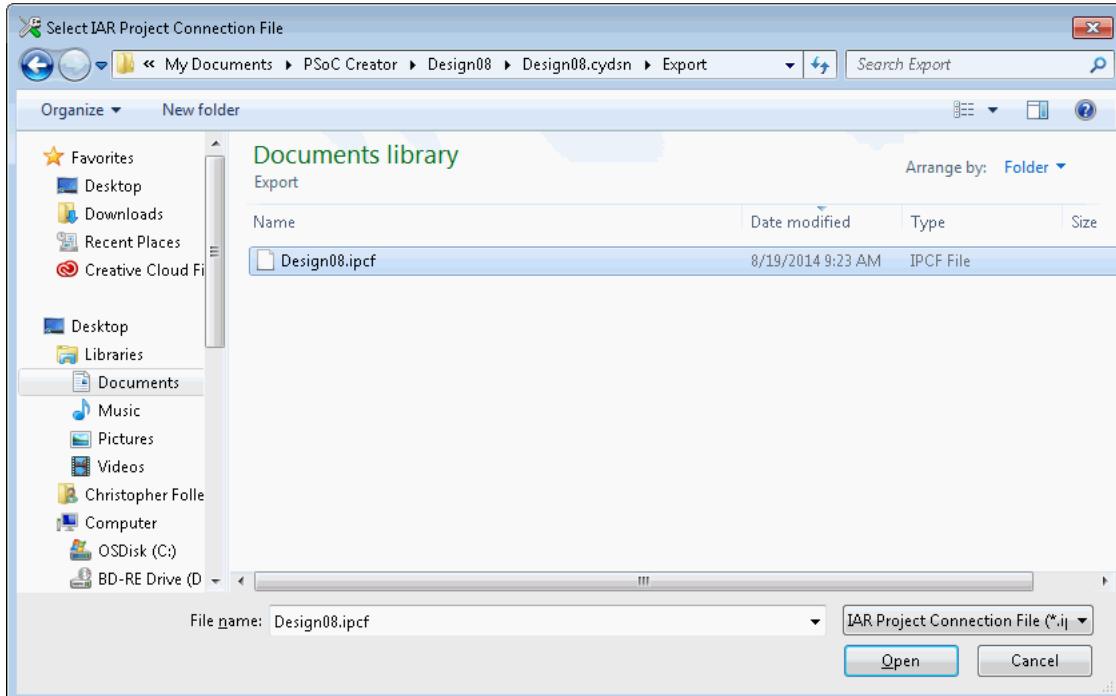
4. Open the IDE Options dialog (**Tools > Options**) and select **Project**. Then, select the **Enable project connections** option and click **OK**.



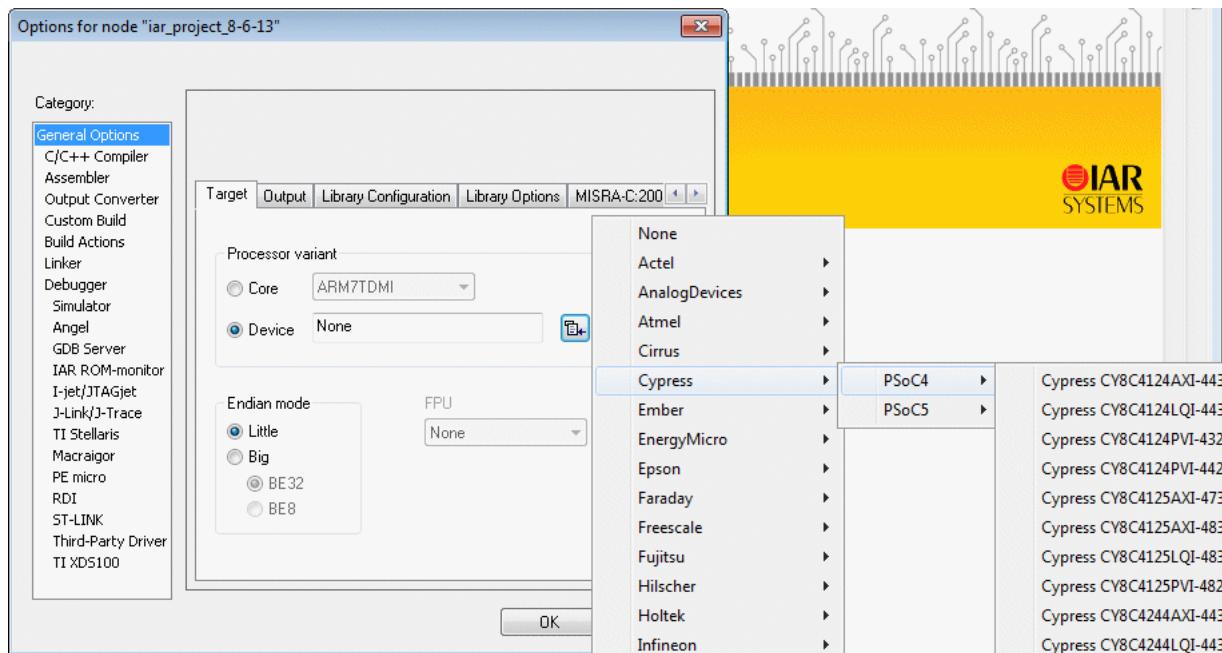
5. Open the Add Project Connection dialog (**Project > Add Project Connection**), select **IAR Project Connection**, and click **OK**.



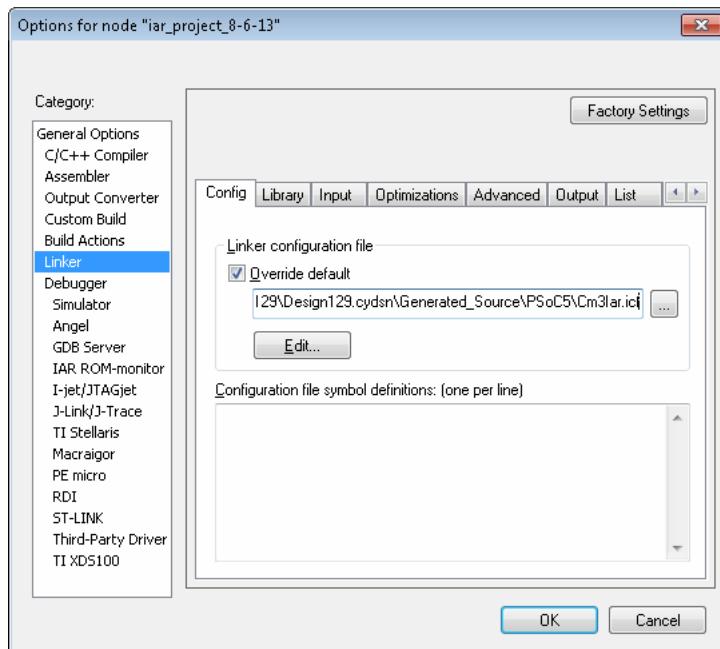
6. On the Select IAR Project Connection File dialog, browse to the PSoC Creator Export directory, select the .ipcf file, and click **Open**.



7. In the EWArm Workspace window, right-click on the project and select **Options...** to open the Options dialog.
8. For IAR versions earlier than 7.10.1 only, do the following:
9. On the **General Options** page, select **Device** under Processor variant. Then, click the **Select Device** button and select the appropriate Cypress device.

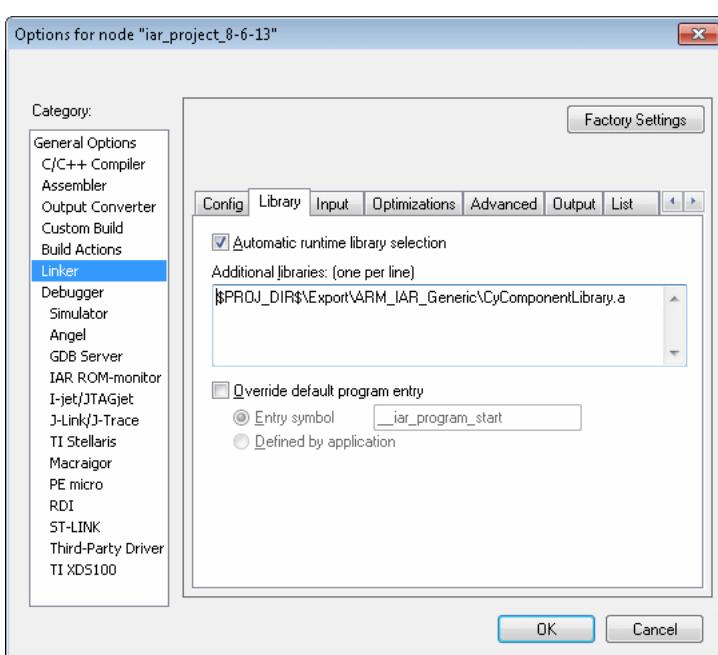


10. On the **Linker** page, select the **Config** tab, check the **Override default** for the Linker configuration file. Navigate to your project directory and go to the Generated_Source/{ARCH} directory and find the .icf file for your project:



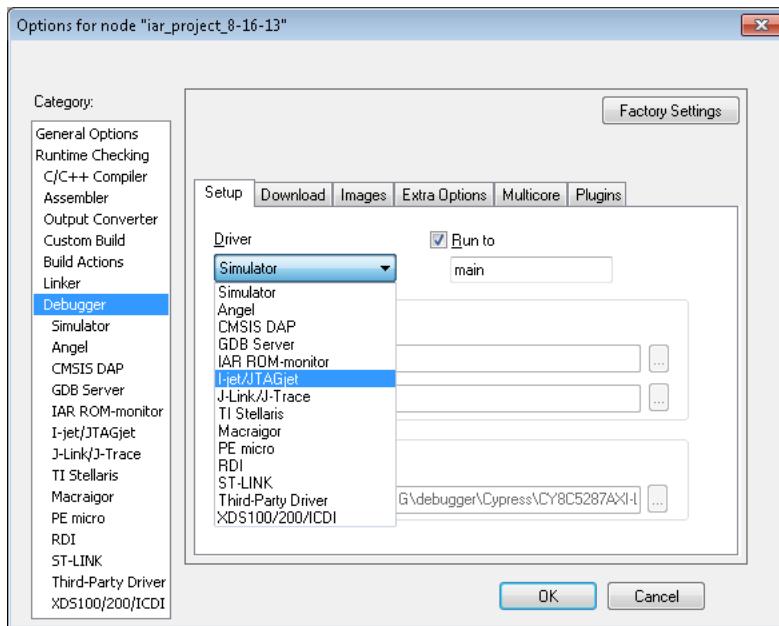
11. If there exists any additional library file such as *CyComponentLibrary.a* under the `<project_name>.cydsn\Export\Arm_IAR_Generic` directory, then on the **Linker** page, select the **Library** tab and add the path to each of the libraries found in the `Export\Arm_IAR_Generic` directory.

Prepend each of them with the IAR command `$PROJ_DIR$|Export\Arm_IAR_Generic\`*CyComponentLibrary.a*

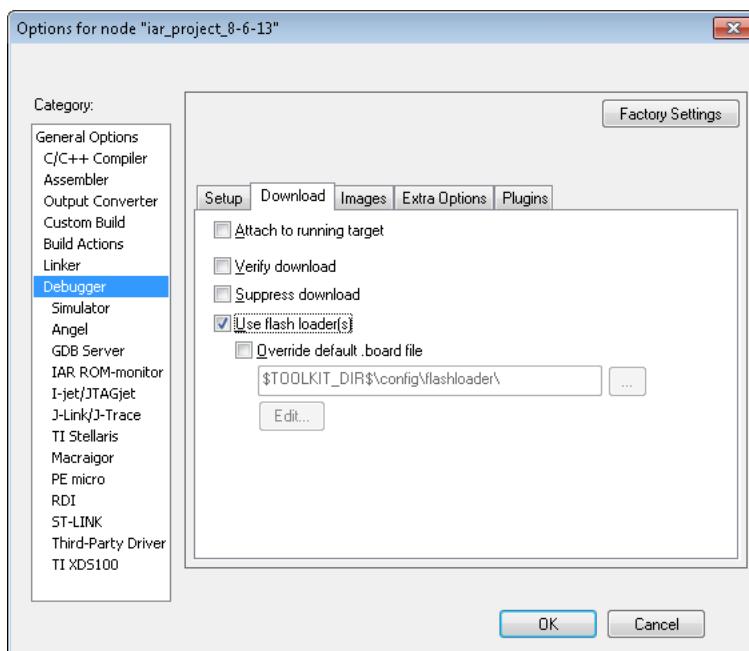


12. For **all IAR versions**, do the following:

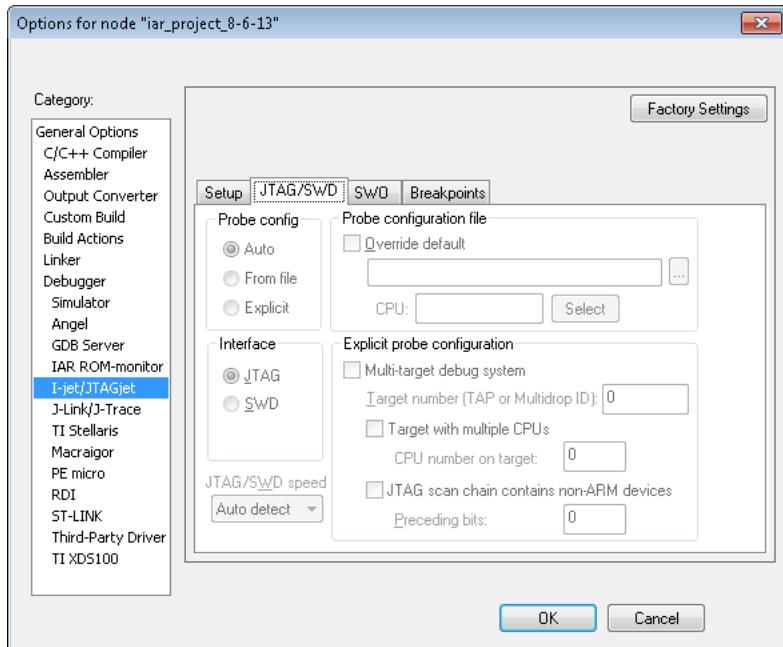
- On the **Debugger** page, click the **Driver** drop-down menu and select the appropriate debugger probe (I-jet and J-Link are currently supported).



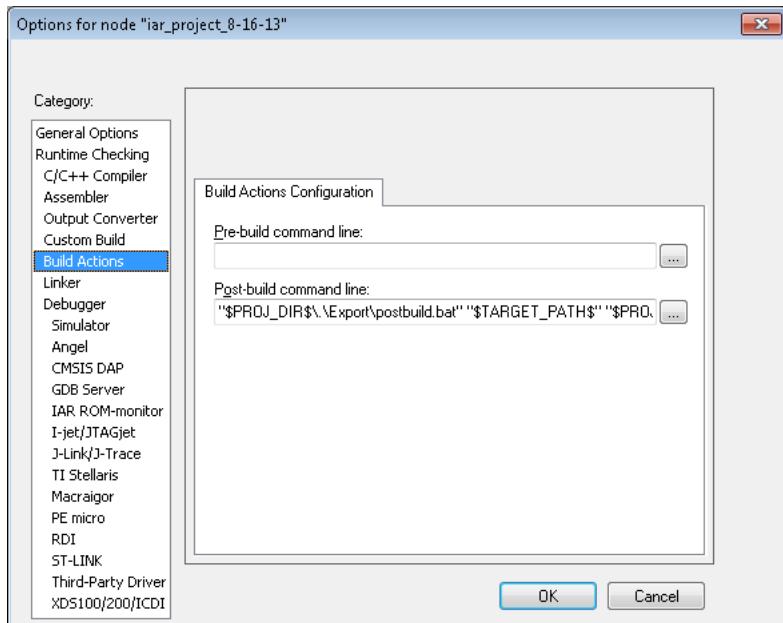
- On the **Debugger** page, select the **Download** tab and check the **Use flash loader(s)** option.



- Select the node for your debugger and select to the **JTAG/SWD** tab. Then, select the appropriate interface for your device.



- On the Build Actions page, enter appropriate pre-build and post-build commands.



PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. For IAR, add these commands to the projects options. PSoC Creator provides a call to our generated pre-build and post-build scripts where needed; you can add your own commands, separating all commands with semicolons.

13. Click **OK** to close the Options dialog.

You can now build, program and debug your EWArm project. Refer to the IAR documentation as needed.

Note For multi-core projects, you need to build the projects for each core in the ascending order from the lower core number to the higher. For example, if you have two projects based on CortexM0p and CortexM4, first build the CortexM0p project then build the CortexM4 project.

Setup for I-jet Programming and Debugging:

For **PSoC 5 LP**, connect the I-jet with 10-pin adapter to the PROG 10-pin connection on the PSoC 5 LP module of your CY8CKIT-001 kit.

Note If the PSoC 5 LP User NVLs are configured such that the device is using the JTAG programming/debugging protocol, the IAR i-Jet configuration must match this.



For **PSoC 4/PRoC BLE**, connect the I-jet with 10-pin adapter to the "PSoC 4/PRoC BLE Prog" 10-pin connector on the CY8CKIT-042 (Pioneer Kit).



For information about how to use the IAR I-jet/Debugger system, go to the IAR IDE. In the Help menu, open the documents named *C-SPY Debugging Guide* and *I-jet User Guide*.

See Also:

- [Exporting a Design to 3rd Party IDE](#)
- [Exporting a Design to IAR IDE](#)

Opening PSoC 4/PSoC 5LP Projects in µVision IDE

As described in [Key IDE Export Files/Projects](#), the export process creates a µVision application project along with several files.

Note The default toolchain (GCC or MDK) of the exported project is the one that was used to build the project in PSoC Creator just before exporting to µVision.

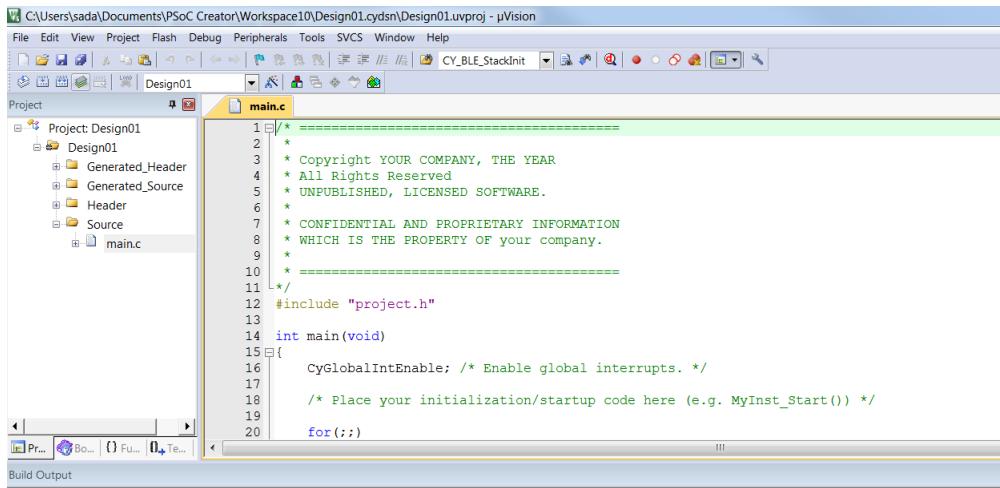
Note if using µVision 5 or later, download and install the µVision 5 legacy support executable from Keil website:

<http://www2.keil.com/mdk5/legacy>

For more information and help with installation, refer to:

http://www.keil.com/support/man/docs/license/license_sul_install.htm

The following is a snapshot of the exported project opened and built in µVision.



```

C:\Users\sada\Documents\PSoC Creator\Workspace10\Design01.cydsn\Design01.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
CY_BLE_StackInit CY_BLE_StackInit
Project Design01 main.c
1 /* =====
2 *
3 * Copyright YOUR COMPANY, THE YEAR
4 * All Rights Reserved
5 * UNPUBLISHED, LICENSED SOFTWARE.
6 *
7 * CONFIDENTIAL AND PROPRIETARY INFORMATION
8 * WHICH IS THE PROPERTY of your company.
9 *
10 * =====
11 */
12 #include "project.h"
13
14 int main(void)
15 {
16     CyGlobalIntEnable; /* Enable global interrupts. */
17
18     /* Place your initialization/startup code here (e.g. MyInst_Start()) */
19
20     for(;;)

```

Build Output

```

*** Using Compiler 'VS6.06 update 4 (build 422)', folder: 'C:\keil-522\ARM\ARMCC\Bin'
Rebuild target 'Design01'
compiling CyFlash.c...
compiling CmOpplusStart.c...
compiling CyDMA.c...
compiling cyPmc...
compiling CyLPClk.c...
compiling CyLib.c...
compiling cyfitter_cfg.c...
compiling cymetadatas.c...
compiling cyutils.c...
compiling main.c...
also compiling CyBootAmmRv.s...
linking
Program Size: Code=1582 RO-data=122 RW-data=40 ZI-data=1332
After Build - User command #1: .\Export\postbuild.bat "C:\Users\sada\Documents\PSoC Creator\Workspace10\Design01.cydsn\Design01.axf" "C:\Users\sada\Documents\PSoC Creator\Workspace10\Design01.cydsn\chdir /d .\Export C:\Users\sada\Documents\PSoC Creator\Workspace10\Design01.cydsn\Export>cyelftool.exe -C "C:\Users\sada\Documents\PSoC Creator\Workspace10\Design01.cydsn\Design01.axf" -O Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:02

```

Debug and Release Builds

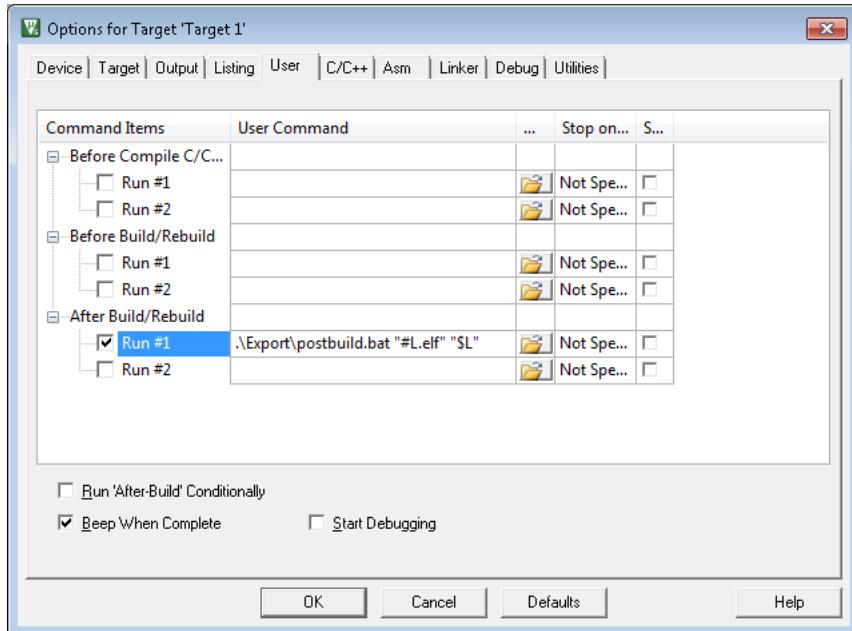
In PSoC Creator, there are two configurations in which the build can be performed: debug and release. A debug build is used for debugging purposes and the release build is for release to customers. The prime difference between the two is in the build options used.

In µVision, this is handled by creating two targets: one for "Debug" and another for "Release." µVision provides a drop down menu to switch between targets.

Setting Options in µVision

The following is a snapshot of the application project in µVision specifying the post build command.

Note PSoC 4/PSoC 5LP exports provide only bare-bones options. After an export for all devices, the user is fully responsible for settings.



Note PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. You can add these commands on the **User** tab. PSoC Creator provides a call to generated pre-build and post-build scripts where needed. You can add your own commands in the unused entry fields on this tab.

See Also:

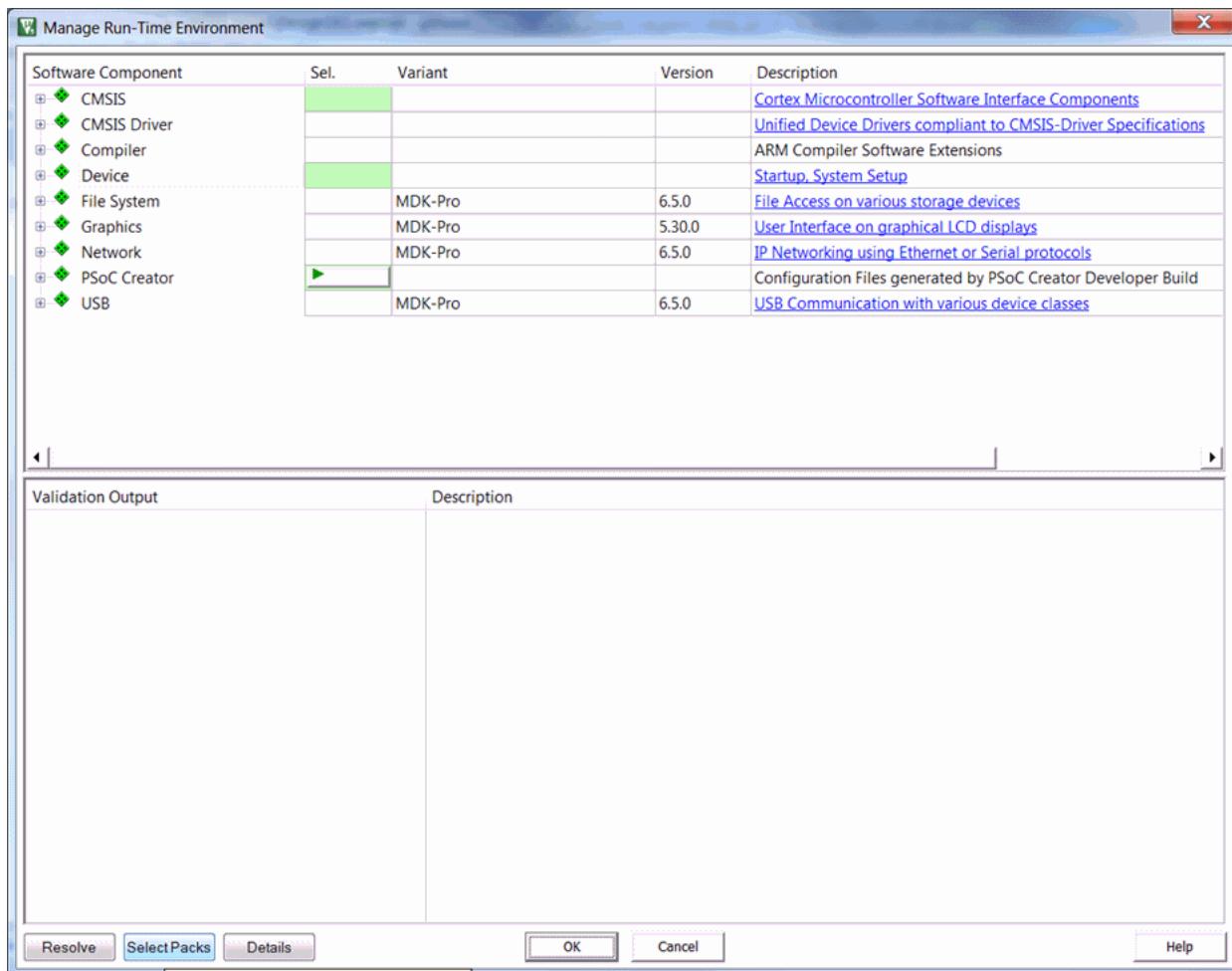
- [Exporting a PSoC 4/PSoC 5LP Design to Keil µVision IDE](#)
- [Key IDE Export Files/Projects](#)

Opening Generated CMSIS-Pack Projects (*μVision 5 IDE*)

As described in [Key IDE Export Files/Projects](#), the export process creates several *μVision* files. You can open the project directly from the PSoC Creator wizard, or open the project from the *μVision* 5 IDE.

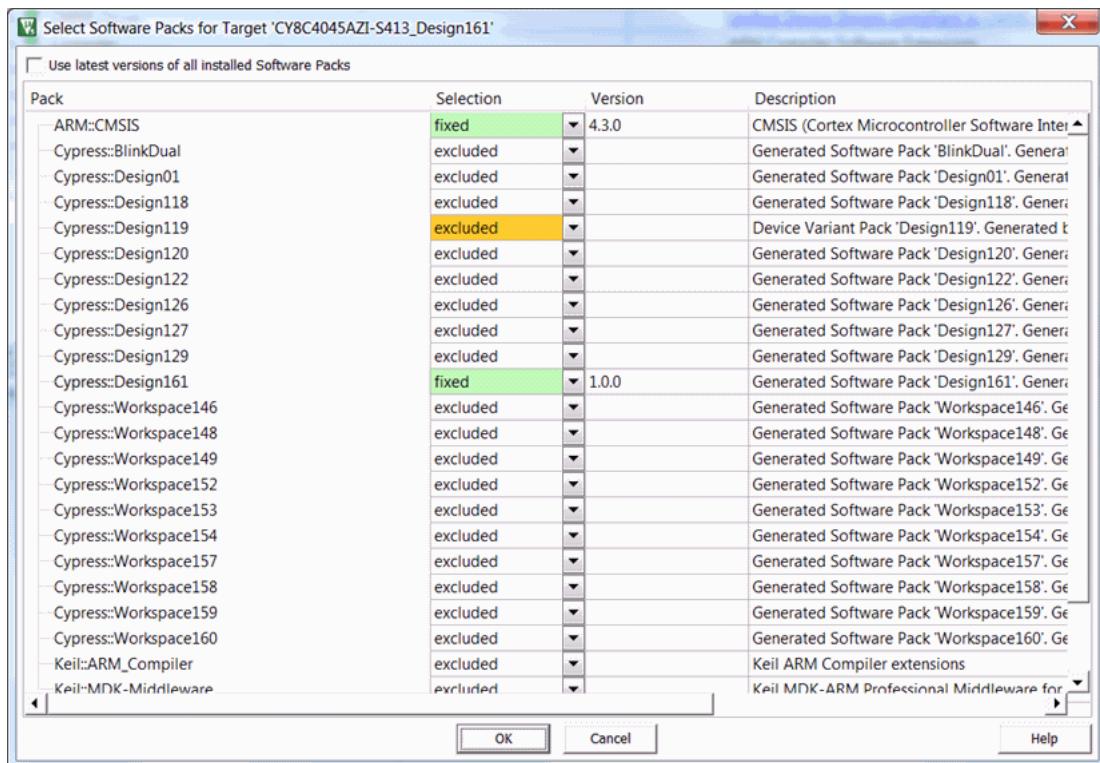
Note For PSoC 6 designs, you must create a *μVision* project per core. For more information, see [Creating uVision Projects for PSoC 6](#).

To make sure you are using the correct pack for your project, open the Manage Run-Time Environment dialog and click the **Select Packs** button.



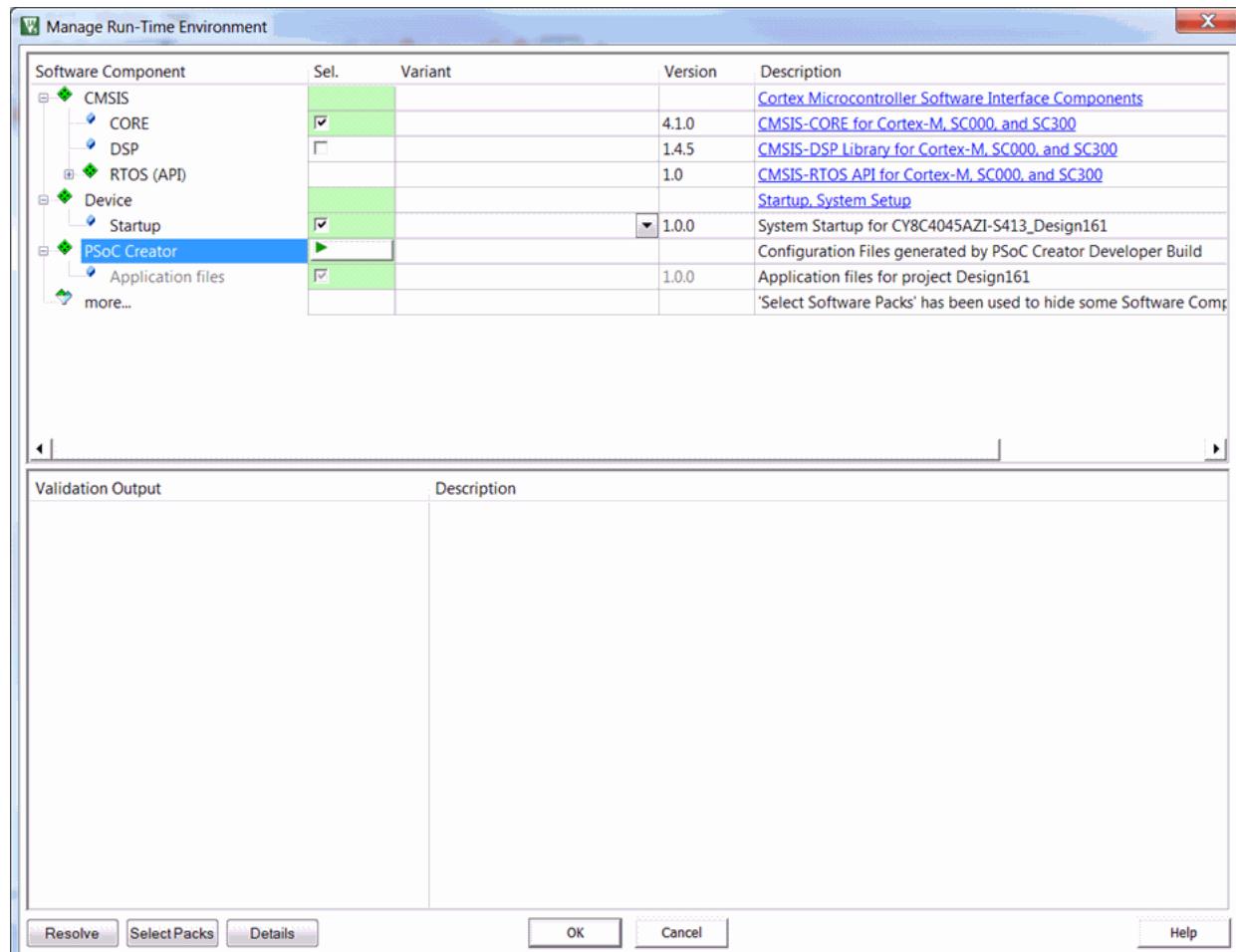
On the Select Software Packs for Target dialog:

- Unselect the Use latest versions of all installed Software Packs check box.
- Under the **Selection** column, select "fixed" from the drop down menu for the CMSIS Pack that PSoC Creator created for your configured device. All other unused packs should be excluded.



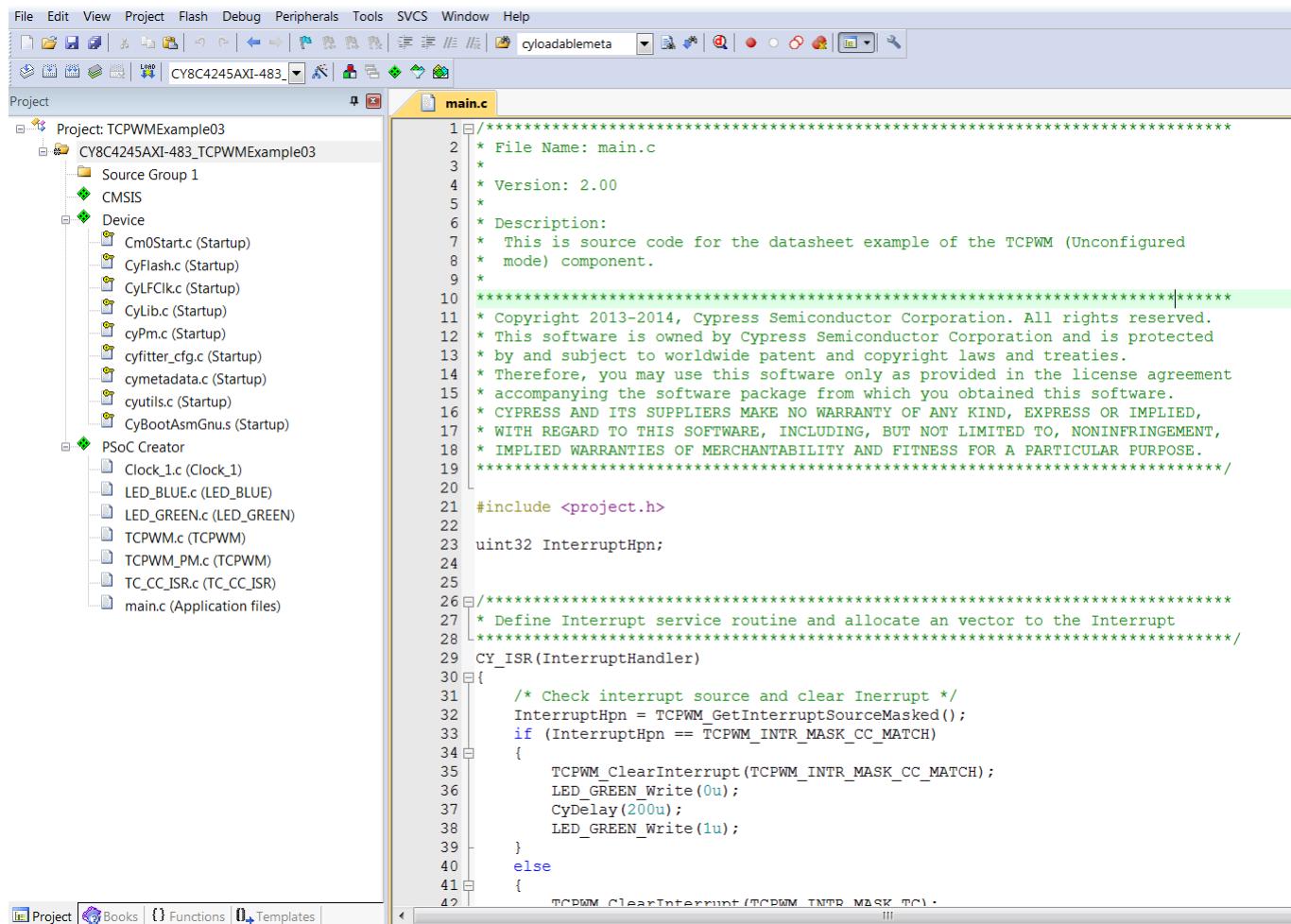
Click **OK**.

Now you should see the software Components **CMSIS Core** and **Device Startup** for your device are selected in Manage Run-Time Environment.



Click **OK** to close Manage Run-Time Environment dialog, and save project settings.

The following is a snapshot of a project exported to µVision 5.



The screenshot shows the PSoC Creator interface with a project titled "TCPWMExample03". The project structure on the left includes Device, CMSIS, and PSoC Creator subfolders. The main code editor window displays the "main.c" file content:

```

1 /* **** */
2 * File Name: main.c
3 *
4 * Version: 2.00
5 *
6 * Description:
7 * This is source code for the datasheet example of the TCPWM (Unconfigured
8 * mode) component.
9 *
10 ****
11 * Copyright 2013-2014, Cypress Semiconductor Corporation. All rights reserved.
12 * This software is owned by Cypress Semiconductor Corporation and is protected
13 * by and subject to worldwide patent and copyright laws and treaties.
14 * Therefore, you may use this software only as provided in the license agreement
15 * accompanying the software package from which you obtained this software.
16 * CYCRESS AND ITS SUPPLIERS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
17 * WITH REGARD TO THIS SOFTWARE, INCLUDING, BUT NOT LIMITED TO, NONINFRINGEMENT,
18 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
19 ****
20
21 #include <project.h>
22
23 uint32 InterruptHpn;
24
25
26 ****
27 * Define Interrupt service routine and allocate an vector to the Interrupt
28 ****
29 CY_ISR(InterruptHandler)
30 {
31     /* Check interrupt source and clear Inerrupt */
32     InterruptHpn = TCPWM_GetInterruptSourceMasked();
33     if (InterruptHpn == TCPWM_INTR_MASK_CC_MATCH)
34     {
35         TCPWM_ClearInterrupt(TCPWM_INTR_MASK_CC_MATCH);
36         LED_GREEN_Write(0u);
37         CyDelay(200u);
38         LED_GREEN_Write(1u);
39     }
40     else
41     {
42         TCPWM_ClearInterrupt(TCPWM_INTR_MASK_TC);
43     }
44 }

```

The following is a snapshot of the project built in *µVision* 5, but before building the project there are some options that need to be set up as explained in the section [Setting Options in *µVision* 5](#).

The screenshot shows the PSoC Creator IDE interface. The left pane displays the Project Explorer with the following structure:

- Project: TCPWMExample03
- CY8C4245AXI-483_TCPWMExample03

 - Source Group 1
 - CMSIS
 - Device
 - CyComponentLibrary.a (Startup)
 - Cm0Start.c (Startup)
 - CyBootAsmRv.s (Startup)
 - CyFlash.c (Startup)
 - CyLFClk.c (Startup)
 - CyLib.c (Startup)
 - cyPmc.c (Startup)
 - cyfitter_cfg.c (Startup)
 - cymetadata.c (Startup)
 - cyutils.c (Startup)
 - PSoC Creator
 - Clock_1.c (Clock_1)
 - LED_BLUE.c (LED_BLUE)
 - LED_GREEN.c (LED_GREEN)
 - TCPWM.c (TCPWM)
 - TCPWM_PM.c (TCPWM)
 - TC_CC_ISR.c (TC_CC_ISR)
 - main.c (Application files)

The right pane shows the content of the main.c file:

```
3 *  
4 * Version: 2.00  
5 *  
6 * Description:  
7 * This is source code for the datasheet example of the TCPWM (Unconfigured  
8 * mode) component.  
9 *  
10 ****  
11 * Copyright 2013-2014, Cypress Semiconductor Corporation. All rights reserved.  
12 * This software is owned by Cypress Semiconductor Corporation and is protected  
13 * by and subject to worldwide patent and copyright laws and treaties.  
14 * Therefore, you may use this software only as provided in the license agreement  
15 * accompanying the software package from which you obtained this software.  
16 * CYPRESS AND ITS SUPPLIERS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,  
17 * WITH REGARD TO THIS SOFTWARE, INCLUDING, BUT NOT LIMITED TO, NONINFRINGEMENT,  
18 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  
19 ****  
20  
21 #include <project.h>  
22  
23 uint32 InterruptHpn;  
24  
25  
26 /**  
27 * Define Interrupt service routine and allocate an vector to the Interrupt  
28 ****/  
29 CY_ISR(InterruptHandler)  
30 {  
31     /* Check interrupt source and clear Inerrupt */  
32     InterruptHpn = TCPWM_GetInterruptSourceMasked();  
33     if (InterruptHpn == TCPWM_INTR_MASK_CC_MATCH)  
34     {
```

The bottom pane shows the Build Output:

```
Build target 'CY8C4245AXI-483_TCPWMExample03'  
linking...  
Program Size: Code=1592 RO-data=156 RW-data=220 ZI-data=1284  
After Build - User command #1: C:\Users\sada\Documents\PSoC Creator\TCPWMExample03\TCPWMExample03.cydsn\Export\postbuild.bat "C:\Users\sada\Documents\PSoC Creator\TCPWMExample03\TCPWMExample03.cydsn\Export\chdir /d ..\Export C:\Users\sada\Documents\PSoC Creator\TCPWMExample03\TCPWMExample03.cydsn\Export\CyElfTool.exe -C "C:\Users\sada\Documents\PSoC Creator\TCPWMExample03\TCPWMExample03.axf" -0 Error(s), 0 Warning(s).  
Build Time Elapsed: 00:00:10
```

Debug and Release Builds

In PSoC Creator, there are two configurations in which the build can be performed: debug and release. A debug build is used for debugging purposes and the release build is for release to customers. The prime difference between the two is in the build options used.

In µVision, this is handled by creating two targets: one for "Debug" and another for "Release." µVision provides a drop down menu to switch between targets.

Setting Options in µVision 5

Note PSoC 4/PSoC 5LP exports provide only bare-bones options. After an export for all devices, the user is fully responsible for settings.

Open the Options for Target ... dialog for the µVision 5 project.

Scatter File

Select the **Linker** tab and unselect the **Use Memory Layout from Target Dialog** check box.

Navigate to the scatter file located in the pack installation directory. The path should be similar to the following:

```
C:\Keil_v5\Arm\Pack\<pack vendor name>\<pack name>\<pack version>\Device\<deviceName_packName>\Source\
```

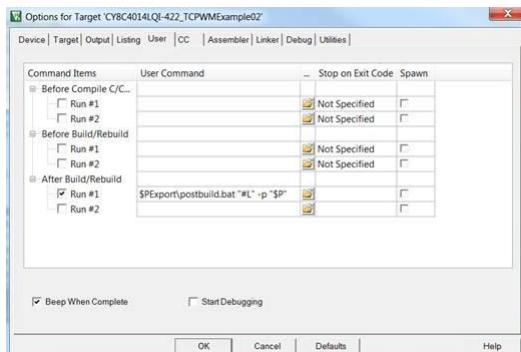
Depending on the selected toolchain, choose the corresponding scatter file:

- For CortexM0, choose *Cm0RealView.scat*
- For CortexM0+, choose *Cm0plusRealView.scat*
- For CortexM3, choose *Cm3RealView.scat*

Pre-Build/Post-Build Script

On the Options for Target ... dialog, select the **User** tab to add/check the following script to the **After Build/Rebuild** text field and check the **Run #1** check box.

```
$PExport\postbuild.bat "#L" -p "$P"
```



For "Bootloadable" and "Bootloader and Bootloadable" projects, add/check the following script to the **Before Build/Rebuild** text field and check the **Run #1** check box.

```
$PExport\prebuild.bat
```

Note Adding the pre-build/post-build commands will be done automatically for some newer versions of µVision, so you need to check it depending on what version of µVision you are using.

See Also:

- [Exporting a Design to Generated CMSIS-Pack](#)
- [Key IDE Export Files/Projects](#)

Opening PSoC Creator Designs in Makefile

Generated Files:

After completion of "Export to Makefile", the following files will be generated:

File	Notes
Makefile	Top-level GNU Make compatible Makefile. This file may be updated or altered as desired.
<i>platform_debug.mk</i> - or - <i>platform_release.mk</i>	Platform and toolchain specific configuration. The Export to Makefile feature will generate <i>platform_debug.mk</i> if Creator is configured to create debug builds, and <i>platform_release.mk</i> if Creator is configured to create release builds. These files may be updated or altered as desired.
<i>app_source.mk</i>	Application firmware source. This file may be updated or altered as desired.
<i>gen_source.mk</i>	PSoC Creator generated source code. This file should NOT be modified. It is automatically re-written by PSoC Creator as a part of the build process. This file is written out for the toolchain selected in the export wizard. Later, it will be updated during each Build process, based on the current toolchain in the Build Settings . That is, if you export to Makefile and change from GCC to MDK, the Makefiles will likely not work. The <i>gen_source.mk</i> file will contain source code that is not supported by the toolchain in <i>platform_debug.mk</i> (or <i>platform_release.mk</i>). Either update the platform configuration file, or use the "Export to Makefile" feature to regenerate the files for the new toolchain.

Important Notes:

- Since the Make utility does not reliably support files with spaces or \$ in the file name, PSoC Creator avoids using them. Also, the tool avoids using colons and slashes in the names of files and folders because some operating systems and drive formats use these characters as volume and directory separators. Furthermore, non-alphanumeric characters may not be supported by all file systems or operating systems. Punctuation marks, parentheses, quotation marks, brackets, and operators, such as following, are often reserved for special functions in scripting and programming languages:

, [] { } () ! ; " ' * ? < > |

Therefore, PSoC Creator checks the design project name and user source files for white spaces and those special characters in their names. PSoC Creator performs the check in the last page of the wizard and the error message specifies the affected files. In this situation you need to address the errors and try to re-export the design.

- The Makefile uses BASH scripts (*prebuild.sh* and *postbuild.sh*) by default. If you would like to use BATCH scripts, you need to modify the Makefile appropriately.
- If you are using Windows, you need CYGWIN (Make package) or MSYS installed in your machine. Cypress has tested this feature on CYGWIN_NT-6.3-WOW64 1.7.33-2(0.280/5/3) i686 cygwin (Make 4.0), and MinGW32_NT-6.1 1.0.18(0.48/3/2) 2012-11-21 i686 Msys (Msys 2013072300).
- If you would like to use another toolchain rather than the one used during the Export, you can modify the *platform_debug.mk*/*platform_release.mk* file to remove the text in front of the TOOLCHAIN_DIR variable and write the path to the target toolchain instead. Be sure to change back slashes to forward slashes in the path.

- You need to install Arm GCC on Linux OS. It can be downloaded from: <https://launchpad.net/gcc-arm-embedded/>
- The additional library files to link, which you added to your design through the build settings dialog (that is, by using the linkers -L and -l arguments), will be populated as linker options in the *platform_debug.mk/platform_release.mk* file. You may add more library files to the APP_LIBS section in the *app_source.mk* file.
- If you generate a makefile, but then make significant changes to your design (for example, changing the selected toolchain) and regenerate a new makefile, you must run "make clean" before running "make" to ensure your make-based build is properly up to date.
- PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. For generated makefiles, you can add these commands to the top-level makefile, as part of the *prebuild_** and *postbuild_** rules.
- PSoC Creator only propagates project-level Build Settings, such as compiler optimization level. The export process does not support propagation of file-level Build Settings to the makefile.

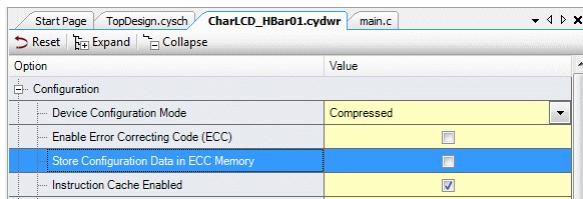
See Also:

- [Exporting a Design to Makefile](#)

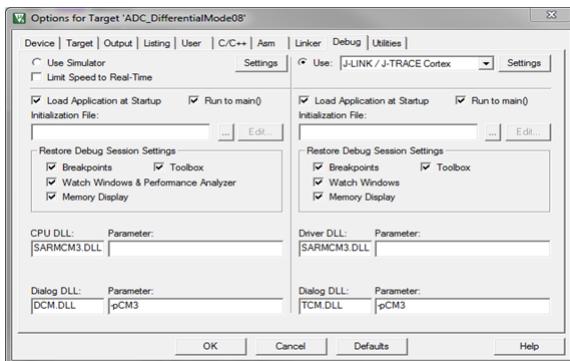
Setting Up for Segger J-Link/J-Trace Debugger for PSoC 5LP

Use the following steps to set up the Segger J-Link in μVision for programming and debugging PSoC 5LP Device Family. For PSoC 4 and PRoC BLE, refer to [Setting Up for ULink2/ULink Pro and Segger J-Link Debugger Probes](#).

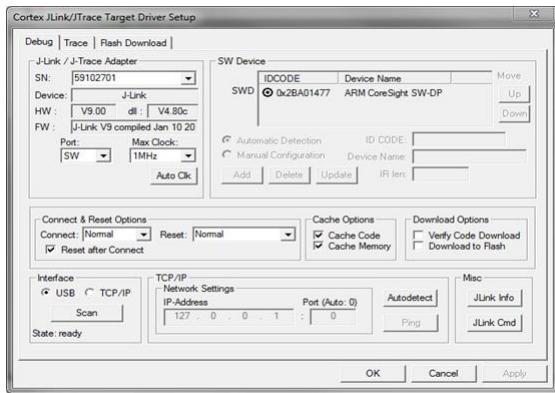
1. Before building and exporting a PSoC 5LP based design to μVision, go to your project's Design Wide Resources System Editor page and disable the "Store Configuration Data in ECC" parameter. Storing configuration data in ECC memory is the default setting for PSoC Creator projects. It must be changed for designs exported to μVision.



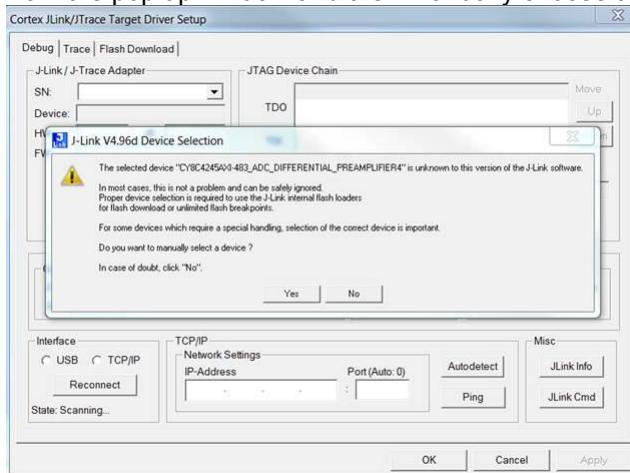
2. Launch the μVision IDE and select **Project > Options for Target** to open the dialog, and go to the **Debug** tab



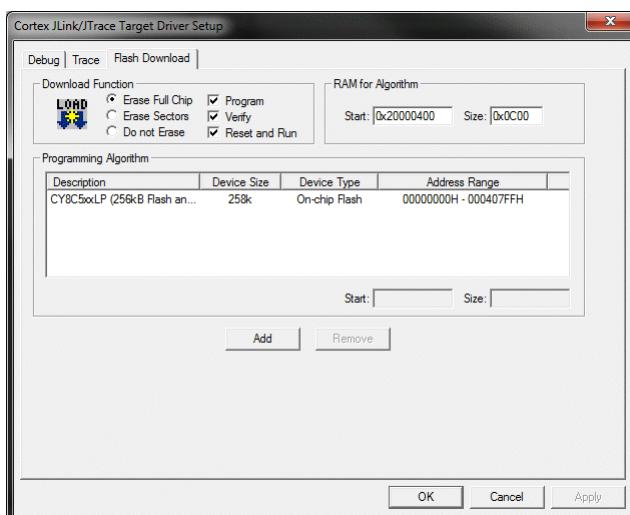
3. Select the appropriate debugger and click **Settings** to open the Driver Setup dialog.



Note Please notice that for µVision 5 exported projects, the target name is not automatically recognized by J-Link SW and you will get a message indicating that the selected device is unknown, so you need to select Yes from the pop up window and then manually choose the appropriate device in the J-Link SW.

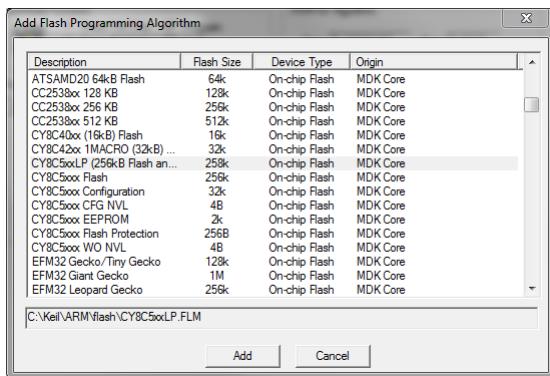


4. On the **Debug** tab, select the appropriate Port (SW or JTAG). Then, click the **Flash Download** tab.



5. Remove all other flash algorithms except CY8C5xxLP Flash. If you have flash algorithms of other memory types such as EERPOM, CFG, NVLs, etc. in HEX then J-Link loader will fail, because it doesn't support them.

6. If CY8C5xxLP isn't listed under Programming Algorithm, click **Add**.



Notes:

For projects exported through µVision: The Programming Algorithm should be present in `C:\Keil\Arm\flash*.FLM`. If µVision doesn't contain a Cypress flashloader, copy flashloaders from PSoC Programmer folder (`C:\Program Files (x86)\Cypress\Programmer\3rd_Party_Configuration_Files`).

For projects exported through Generated CMSIS-Pack: The Programming Algorithm should be present in a path similar to the following based on µVision 5 installation directory and your chosen pack name information:

`C:\Keil_v5\Arm\PACK\<PackVendor>\<PackName>\<PackVersion>\FLM\<CypressDeviceName>*.FLM`

Setting Up for ULink2/ULink Pro and Segger J-Link Debugger Probes

These steps are for users of the ULink2/ULink Pro and Segger J-Link debugger probes. These instructions apply as follows:

- ULink2/ULink Pro is for PSoC 4, PRoC BLE, and PSoC 5LP devices.
- Segger J-Link is for PSoC 4 and PRoC BLE devices. To set up Segger J-Link for PSoC 5LP devices, refer to [Setting Up for Segger J-Link/J-Trace Debugger for PSoC 5LP](#).

In order to fully enable PSoC 4/PRoC BLE/PSoC 5LP programming/debugging in µVision 4.72a you will need to download an "add-on" for µVision.

Note If you are using a version later than 4.72a, you do not need this download.

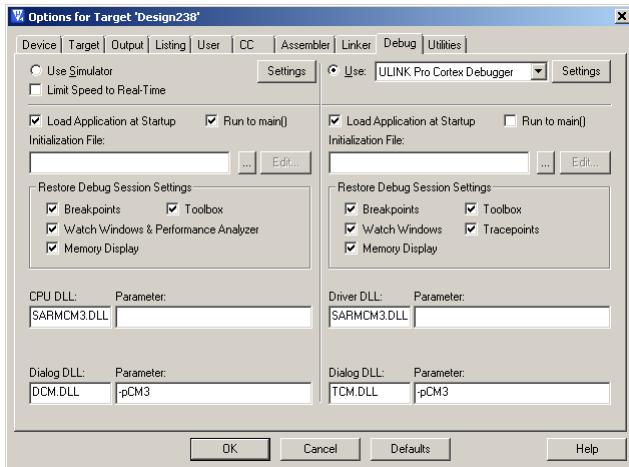
The add-on can be found at the following link: <http://www.cypress.com/go/creator/uvisionimportdownload>

Once you have downloaded the add-on, unzip the archive and run the executable and follow the steps to install it.

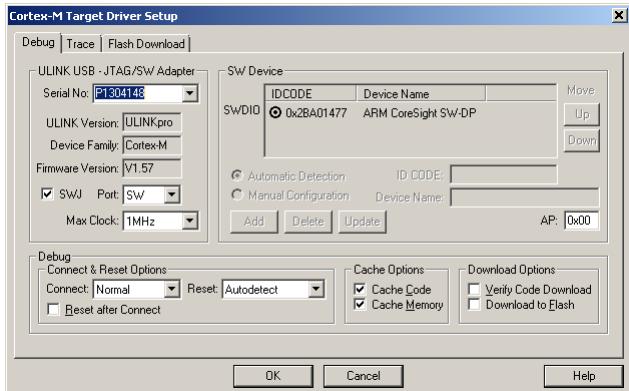
Then, run µVision and follow these steps to set it up:

1. Before building and exporting a PSoC 5LP based design to µVision, go to your project's Design Wide Resources System page and Disable the "Store Configuration Data in ECC" parameter. Storing configuration data in ECC memory is the default setting for PSoC Creator projects, but must be changed for designs exported to µVision.

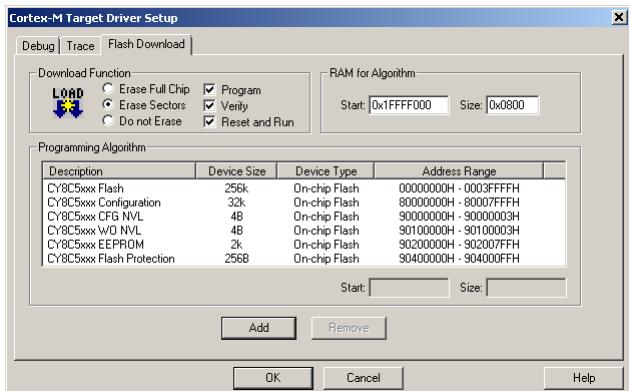
2. Select **Project > Options for Target** to open the dialog, and go to the **Debug** tab.



3. Select the appropriate debugger and click on **Settings**.



4. Select the appropriate Port (SW or JTAG), and click on the **Flash Download** tab.



5. Select **Erase Full Chip** and if the flash algorithms have not been added automatically, Click the **Add** button and add the proper On-chip Flash algorithm based on the following table. Then set the size of RAM for Algorithm as indicated in the following table.

"RAM for Algorithm" values for Keil ULink (PSoC 4/PRoC BLE/PSoC 5LP) and Segger J-Link (PSoC 4/PRoC BLE) debuggers

PSoC Device family	RAM for Algorithm		Programming Algorithm *
	Start	Size	
PSoC 4000	0x20000200	0x0600	CY8C40xx (16/8kB) Flash
PSoC 4000S/PSoC 4700S	0x20000200	0x0600	CY8C44xx (32/16kB) Flash
PSoC 4100/PSoC 4200	0x20000300	0x0D00	CY8C42xx 1 MACRO (32/16kB) Flash
PSoC 4100 BLE/ PSoC 4200 BLE, PRoC BLE	0x20000200	0x3E00	CY8C42xx/41xx-BLE, CYBL10xx (128kB) Flash
PSoC 4100 BLE/ PSoC 4200 BLE, PRoC BLE	0x20000400	0x7C00	CY8C42xx/41xx-BLE, CYBL10xx (256kB) Flash
PSoC 4100M/PSoC 4200M/ SHM35x2M	0x20000200	0x0C00	CY8C42xx/41xx-M (128/64/32kB) Flash
PSoC 4200L/SHM35x2L	0x20000400	0x1C00	CY8C42xx-L (256/128/64kB) Flash
PSoC 4100S / PSoC 4000DS / PSoC 4200DS	0x20000200	0x1E00	CY8C42xx-D (64/32/16kB) Flash
PSoC 4100S Plus	0x20000300	0x1D00	CY8C41xx-S3_128, CY8C41xx-S3_64 (128/64kB) Flash
PSoC Analog Coprocessor	0x20000200	0x0E00	CY8C44xx (32/16kB) Flash
PSoC 5LP/SHM35x3L	0x20000400	0x0C00	CY8C5xxxx Flash CY8C5xxxx Configuration CY8C5xxxx CFG NVL CY8C5xxxx WO NVL CY8C5xxxx EEPROM CY8C5xxxx Flash Protection

Notes:

For projects exported through µVision : The Programming Algorithm should be present in C:\Keil\Arm\flash*.FLM. If µVision doesn't contain a Cypress flashloader, copy flashloaders from PSoC Programmer folder (C:\Program Files (x86)\Cypress\Programmer\3rd_Party_Configuration_Files).

For projects exported through Generated CMSIS-Pack: The Programming Algorithm should be present in a path similar to the following based on µVision installation directory and your chosen pack name information:

C:\Keil_v5\Arm\PACK\<PackVendor>\<PackageName>\<PackVersion>\FLM\<CypressDeviceName>*.FLM

For more information about how to set up your project; please refer to the "Third-Party Tools for Cypress Devices User Guide.pdf" file in "Documents" folder in the '3rd_Party_Configuration_Files' folder located in the root installation folder of PSoC Programmer.

PSoC 3 Designs

This section contains the following topics:

- [Exporting a PSoC 3 Design to Keil µVision IDE](#)
- [Updating PSoC 3 Projects for µVision IDE Export](#)
- [Opening PSoC 3 Projects in µVision IDE](#)

Exporting a PSoC 3 Design to Keil µVision IDE

PSoC Creator supports multiple PSoC devices as well as multiple versions of the Keil µVision IDE. This topic applies to PSoC 3 devices only.

- PSoC 3 device projects can be exported to either µVision 4 or µVision 5 IDEs.
- The export to µVision IDE process is slightly different for PSoC 3 devices than it is for PSoC 4 and PSoC 5LP devices. See [Exporting a PSoC 4/PSoC 5LP Design to Keil µVision IDE](#).
- The **Export to Generated CMSIS-Pack** option applies only to PSoC 4 and PSoC 5LP devices, and it applies only to the µVision 5 IDE. See [Exporting a Design to Generated CMSIS-Pack](#).

Note The Export to IDE wizard does not apply to PSoC 6 devices. For PSoC 6 devices, use the [Target IDEs](#) section of the [Build Settings](#) dialog to choose third party IDEs for which to generate files.

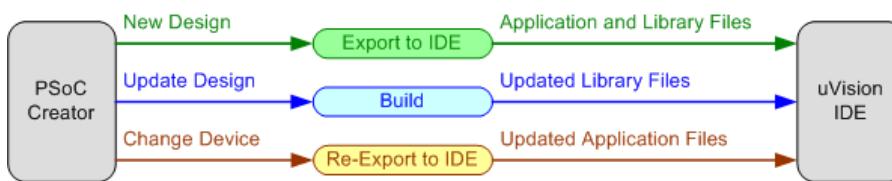
Use PSoC Creator to develop a PSoC hardware design. Then use either PSoC Creator or Keil's µVision IDE for firmware development. To use the µVision IDE, you must [build the design in PSoC Creator](#) and then use the Export to IDE feature.

Note PSoC Creator supports the Advanced System Viewer in Keil µVision. Ensure that you have the appropriate version of µVision to support CMSIS-SVD.

Note When building a project in µVision, all output is written into the UV4Build directory found in your project directory.

For PSoC 3, the build and export process creates a two-project approach in µVision: a library project that contains the design APIs and source files, plus an application project that contains the startup file and firmware template file.

The following diagram shows the high-level workflow for creating a new PSoC Creator design and using the export process to generate projects and files for µVision. It also shows the workflows for updating a PSoC Creator design. See [Key IDE Export Files/Projects](#) for more information about the projects and files.



Each of these flows is described in the following sections:

- [Exporting a New PSoC Creator Design](#)
- [Updating a PSoC Creator Design](#)
- [Changing a PSoC Device](#)

Note Creating and exporting a PSoC 3 project and then changing the device to a PSoC 4/PRoC BLE/PSoC 5LP and re-exporting is not supported. If you wish to do this, you must manually delete your µVision project (*.uvproj) before re-exporting. Then follow instructions for [Exporting a Design to Keil µVision IDE \(PSoC 4/PRoC BLE/PSoC 5LP\)](#).

Exporting a New PSoC Creator Design:

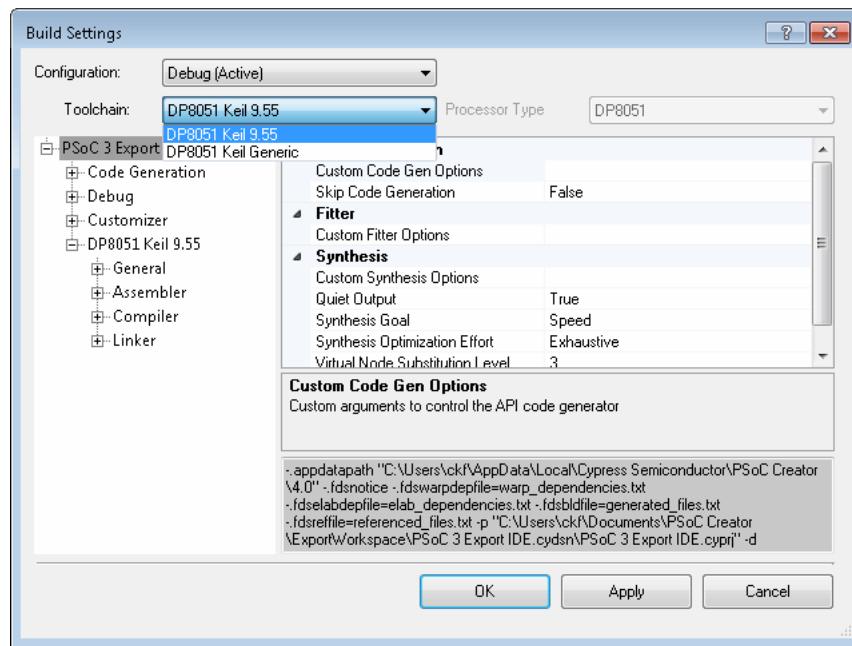
The initial flow is to create a design in PSoC Creator and export the design to the µVision IDE.

1. Create your design in PSoC Creator in the usual manner.

Note If there are any spaces in the project name, the exported project cannot be built in µVision.

Note If you want to program the device using JTAG in your desired 3rd party IDE, you must set the Select Debug option to JTAG in [PSoC Creator System Editor](#).

2. Open the [Build Settings dialog](#) and select the appropriate toolchain for your project. See [PSoC Creator Toolchain Settings](#) for more information.



3. Use the **Project > Export to IDE** menu option to open the IDE Export Wizard dialog.

Note PSoC Creator does not support PSoC 3-based bootloader/bootloadable application development in µVision.

4. If you have not built the design, you will be prompted to build at this time.



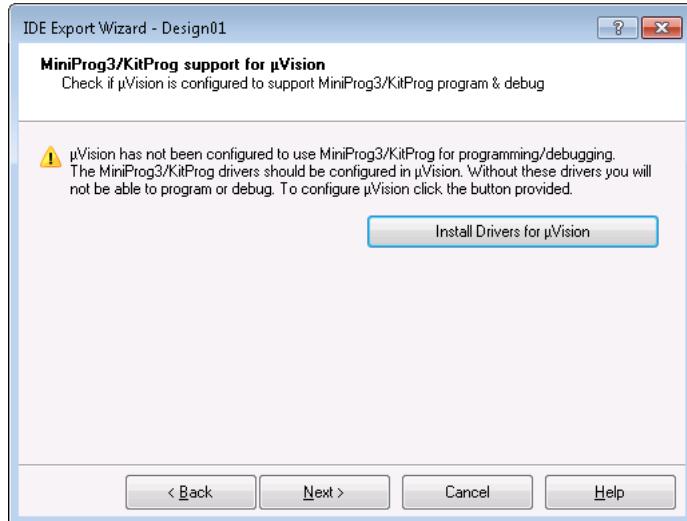
5. Click **Yes** to build. If you click **No**, the export process will be cancelled.

After a successful build, the IDE Export Wizard opens.



- If not already selected, select the µVision option. Click **Next >** to continue.

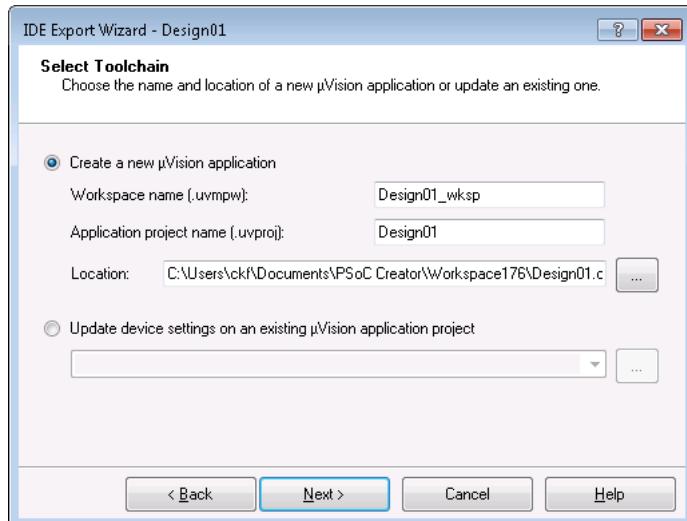
PSoC Creator checks whether MiniProg3/KitProg drivers have been registered with µVision. If they are not registered, the MiniProg3/KitProg support for µVision step opens. However, if PSoC Creator locates a µVision installation with MiniProg3/KitProg support enabled, then it will skip this step. Go to [step 7](#).



Note PSoC Creator only examines the first µVision installation found to see if it has MiniProg3/KitProg drivers properly registered. If you have multiple copies of µVision installed on your computer, the auto-detection process may not be accurate. If you are sure you already have drivers registered, you may skip this step of the wizard.

- If you need to register the drivers, click **Install Drivers for µVision** and follow the prompts on the installation wizard. This process is only required once. See [Registering MiniProg3/KitProg Drivers](#) for more information.
- If you need to register drivers with another µVision installation, go to the PSoC Creator **Tools** menu and select **Install drivers for µVision** to launch the installation wizard.
- When complete with this step, click **Next >** to continue.

The Export Project step opens.



7. Select the **Create a new µVision application** option.

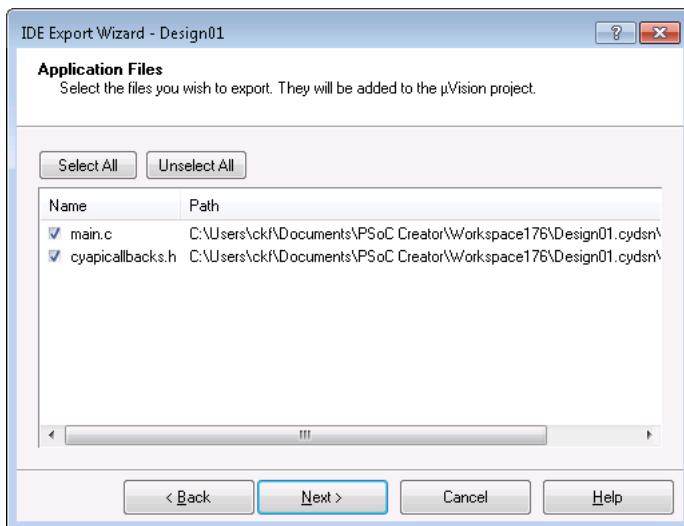
Use this option when exporting the design for the first time to generate a new µVision application project. If you want to update a previously exported design, see [Updating µVision Projects](#).

- Optionally, modify the name for the µVision workspace and project. By default, these are based on the names of the PSoC Creator project being exported.
- The location needs to be the <Project_Name>.cydsn directory.

Note The wizard will not overwrite an existing project or workspace file. If you want to replace an existing µVision project, you must delete it using Windows Explorer first.

8. Click **Next >** to continue.

The Application Files step opens.



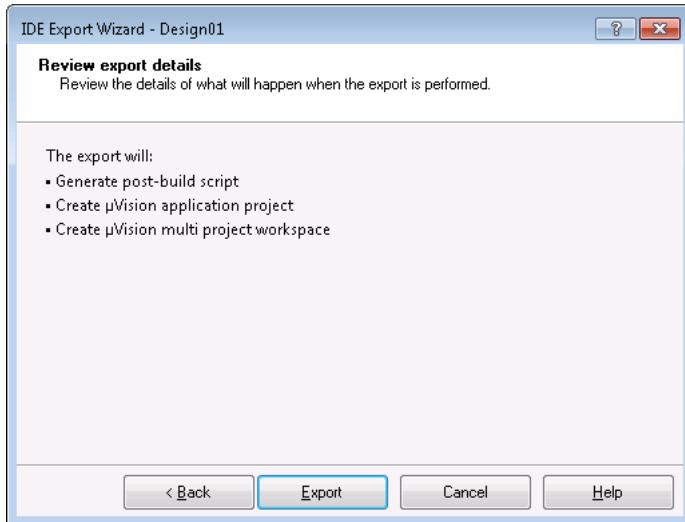
- Select the files you want added to your new µVision application project.

Note The wizard does not copy these files when exporting. It simply adds references to the existing files to the µVision application project.

- To start with an empty µVision application project, click the **Unselect All** button.
- Once the initial export is complete, build settings and file management must be performed within the µVision environment. For example, if you want to add a new source file, select **File > New** in µVision.

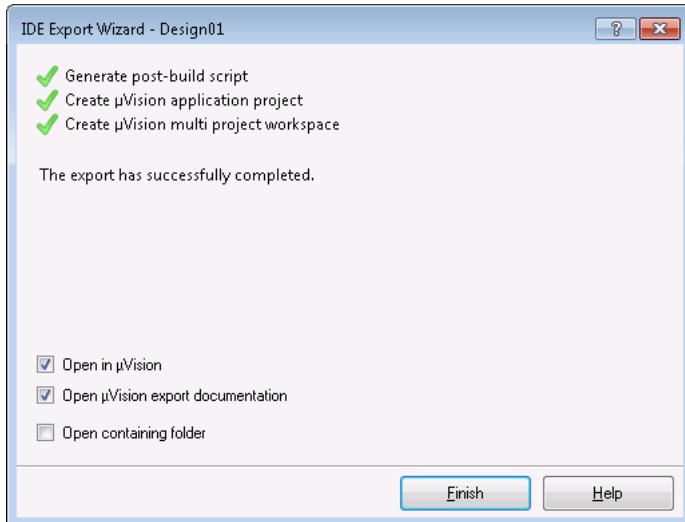
9. Click **Next >** to continue.

The Review export details step opens.



10. Review the export details and click **Export**.

The final step opens showing the completed export.



Optionally select the following action(s) when the export process completes:

- Open the project in μVision

Note If you have multiple versions of μVision IDE installed (for example version 4 and version 5), this option will launch the version you installed most recently. If version 4 does not launch, close the μVision IDE, and launch version 4 manually.

- Open μVision Export documentation
- Open the folder containing the project files

11. Click **Finish** to close the wizard.

See Also:

- [Key IDE Export Files/Projects](#)
- [Opening Projects in µVision IDE](#)
- [Updating µVision Projects](#)
- [Flash Programming/Debugging using µVision](#)
- [PSoC Creator Toolchain Settings](#)
- [Registering MiniProg3/KitProg Drivers](#)
- [Miscellaneous Export Notes](#)

Updating PSoC 3 Projects for µVision IDE Export

This section is for **PSoC 3 only**. For PSoC 4/PRoC BLE and PSoC 5LP, any changes to your design, other than code changes, require a re-export.

In most cases, when you update your design in PSoC Creator, such as adding a Component or configuring a pin, you only need to rebuild your design in PSoC Creator. The list of files in the µVision library project is automatically updated. Simply rebuild the library project in µVision to gain access to the new Component APIs. However, if you select a different PSoC device in PSoC Creator, then you have to re-export the design to update the µVision application project.

Library Project Updates (Change PSoC Creator Design):

To update the library project:

1. Make the necessary changes in PSoC Creator, such as add a pin to your design.
2. Save and build the design in PSoC Creator to generate an updated µVision library project.
3. Open the library project in µVision and rebuild it.
4. In most cases, you do not have to re-export the application project.

What Gets Updated:

- Device name (same as application project)
- CPU information (same as application project)
- Output file name (<OutputName> key)
- Create library and create executable (<CreateLib> and <CreateExecutable> keys)
- SRAM/flash start address and size just like application project
- List of files in the 'Source Files' group in the project file

Application Project Updates (Change PSoC Device):

If you have previously exported the design, and then change the PSoC device in PSoC Creator, the µVision output window will contain an error message indicating the precise problem. Use the IDE Export Wizard to update the key device information in order to clear the error.

Note The µVision IDE does not allow post-build script to report errors. You will have to manually inspect the text of the output window to identify these errors.

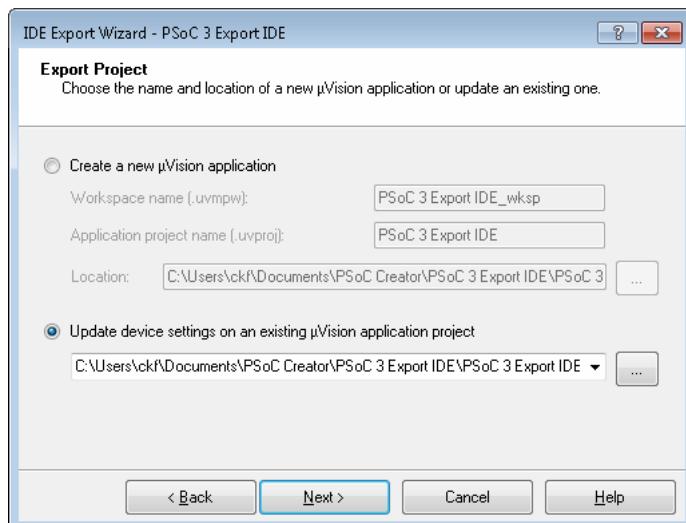
To update the application project:

1. Save and build the design in PSoC Creator.
2. Then use the **Project > Export to IDE** menu option to open the IDE Export Wizard dialog.



3. If not already selected, select the µVision option. Click **Next >** to continue.

The Export Project step opens.

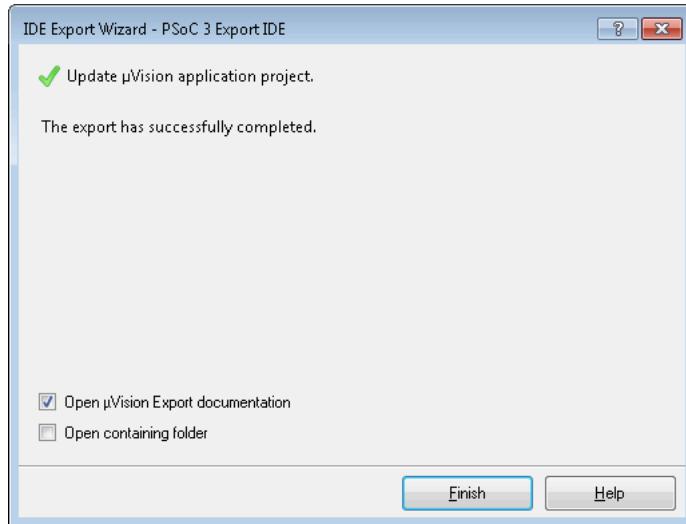


4. Choose the **Update device settings...** option, and navigate to where the application project is stored.

This option will update the necessary application-level files, but it **will not** update any firmware files.

Note If you have the application project open in the µVision IDE, Cypress recommends that you close it before re-exporting from PSoC Creator.

- Click **Next >**. The wizard shows the changes that were made to the application project.



You can select the option to open the documentation and/or the containing folder.

- Click **Finish** to complete the export process from PSoC Creator. Then re-open the project in µVision and rebuild it

Note The **Update device settings...** option only modifies the Application project. No other files mentioned in [Key IDE Export Files/Projects](#) will be changed.

What Gets Updated:

- Device name (the <Device> key in the project file)
- CPU information (the <Cpu> key in the project file)
- Flash start address (the <StartAddress> key in the <OnChipMemories><IRO> section)
- Flash size (the <Size> key in the <OnChipMemories><IRO> section)
- SRAM start address (the <StartAddress> key in the <OnChipMemories><Ocr1> section)
- SRAM size (the <Size> key in the <OnChipMemories><Ocr1> section)
- all Include path entries that point to the PSoC Creator Generated_Source directory (all <IncludePath> keys)

See Also:

- [Exporting a PSoC 3 Design to Keil µVision IDE](#)
- [Files/Projects Exported](#)

Opening PSoC 3 Projects in µVision IDE

As described in [Key IDE Export Files/Projects](#), the export process creates several µVision files and projects. You can open the library project and application project separately, or you can open the multi-project workspace that contains both the library and application projects.

Note if using µVision 5 or later, download and install the µVision 5 legacy support executable from Keil website:

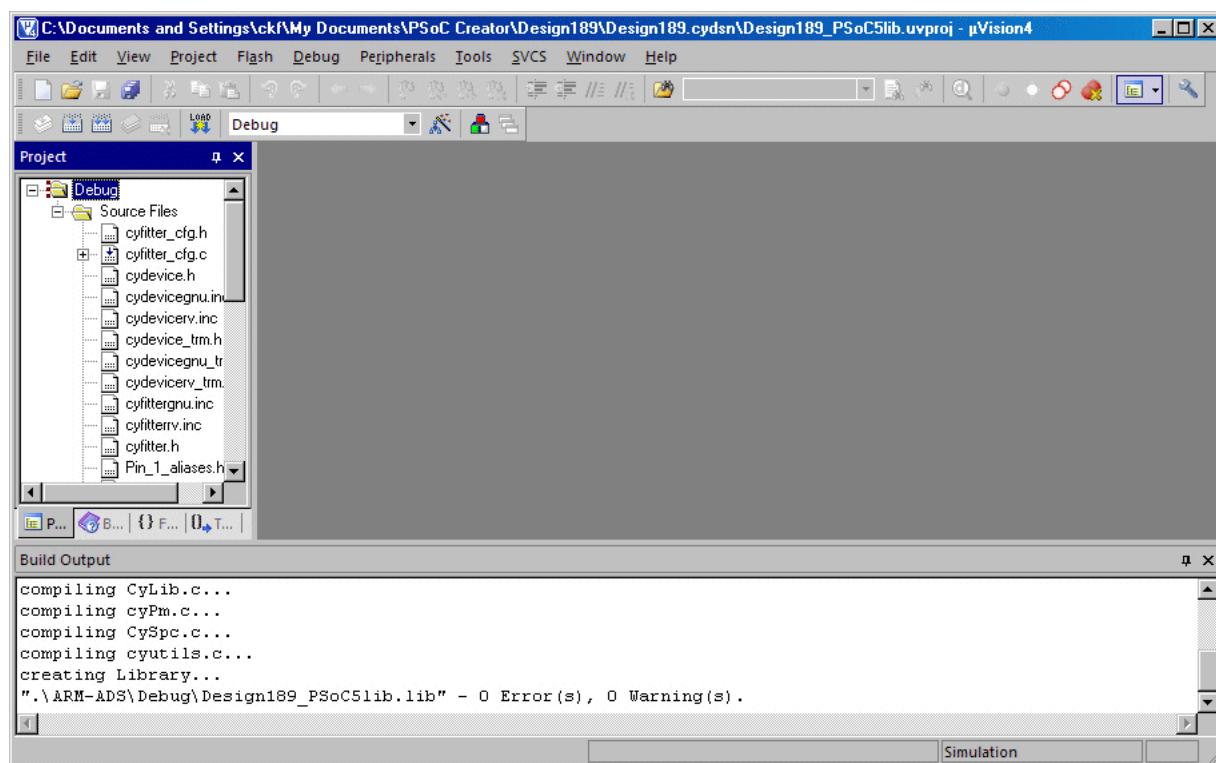
<http://www2.keil.com/mdk5/legacy>

For more information and help with installation, refer to:

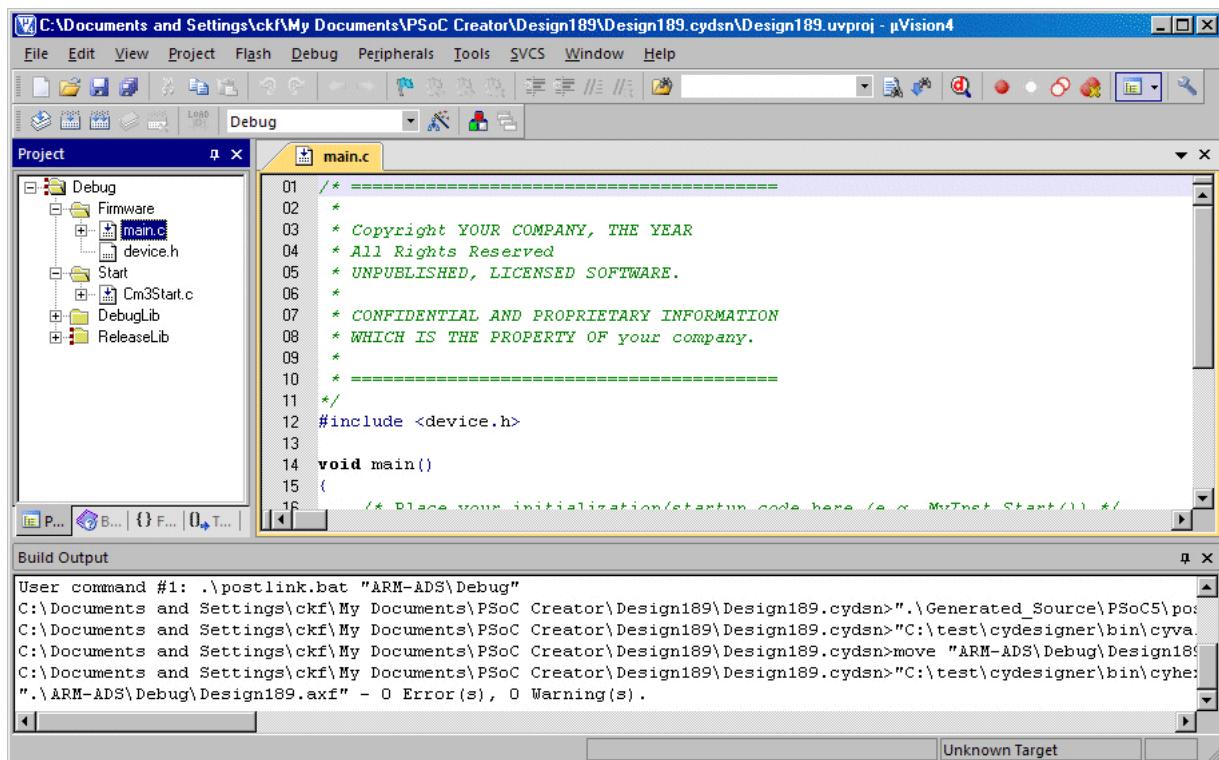
http://www.keil.com/support/man/docs/license/license_sul_install.htm

Note The library project must be up to date in order to build the application successfully. If the application project fails to build because it cannot find the library it is likely because that project needs to be built in µVision. Open the library project or use the batch build option from the multi-project workspace to correct this situation.

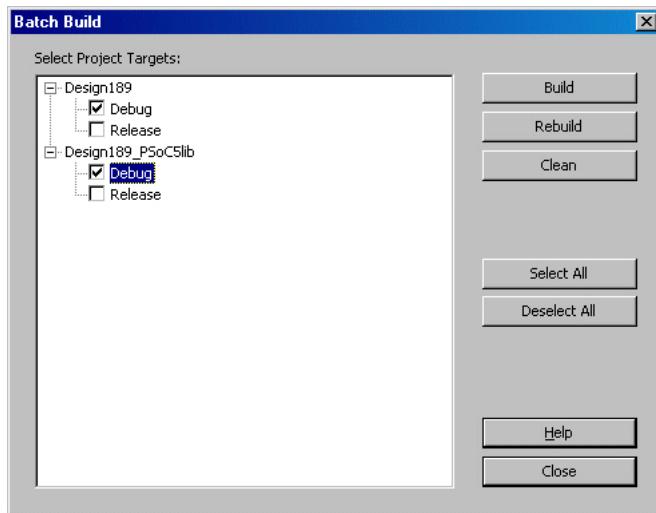
The following is a snapshot of the library project opened and built in µVision.



The following is a snapshot of the application project opened and built in µVision.



Both of these projects can be opened together in a µVision workspace <*ProjectName.uvmpw*>. Both projects can be built together using **Project > Batch Build** as shown in the following image.



Debug and Release Builds

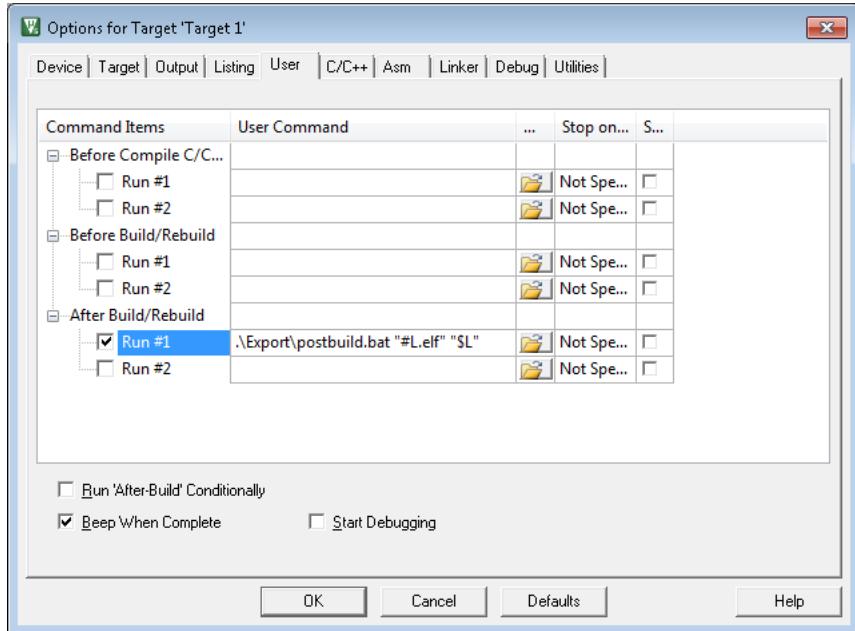
In PSoC Creator, there are two configurations in which the build can be performed: debug and release. A debug build is used for debugging purposes and the release build is for release to customers. The prime difference between the two is in the build options used.

In µVision, this is handled by creating two targets: one for "Debug" and another for "Release." µVision provides a drop down menu to switch between targets.

Setting Options in µVision

The following is a snapshot of the application project in µVision specifying the post build command.

Note After an export for all devices, the user is fully responsible for settings.



See Also:

- [Exporting a PSoC 3 Design to Keil µVision IDE](#)
- [Key IDE Export Files/Projects](#)

FM0+ Designs

This section contains the following topics:

- Exporting a FM0+ Design to Makefile
- Opening FM0+ Designs in Makefile

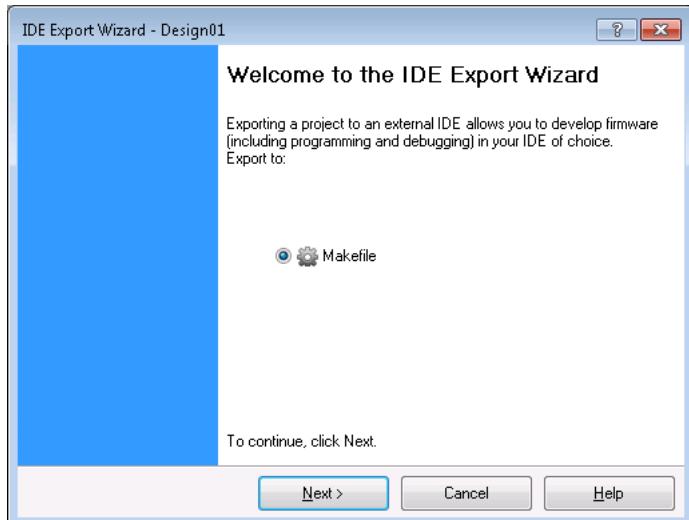
Exporting a FM0+ Design to Makefile

The GNU Make is a utility that uses a Makefile to automatically determine which pieces of a large program need to be recompiled, and then issues commands to recompile those pieces. The Makefile tells the Make utility what to do and how to compile and link a program. Most IDEs support a simple Makefile-based build option. The PSoC Creator Export to Makefile feature allows you to build PSoC designs in those IDEs, as well as from the command-line.

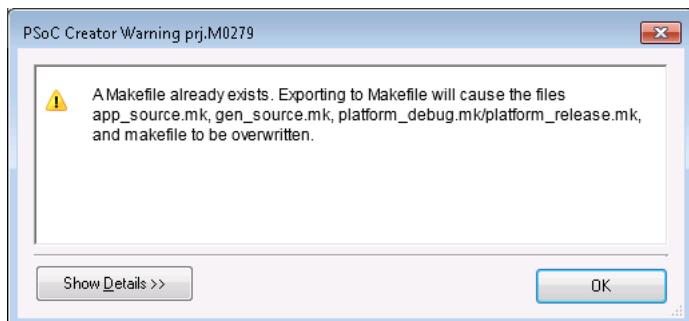
Note For PSoC 6 devices, use the [Target IDEs](#) section of the [Build Settings](#) dialog, and enable the **Makefile** option.

To Export to Makefile:

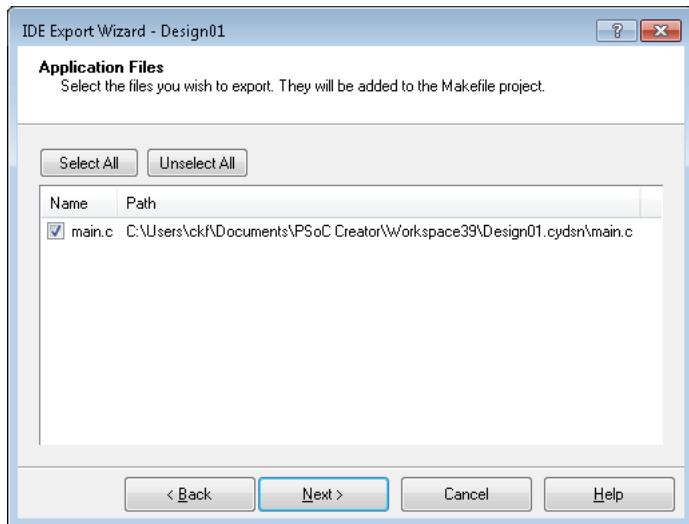
To export your PSoC Creator design to a Makefile, select the "Makefile" option on the IDE Export Wizard dialog and click **Next >**.



Note If you previously used this feature, PSoC Creator will re-create and overwrite the Makefile and .mk files, but will prompt you to confirm the action.



The Application Files step opens.

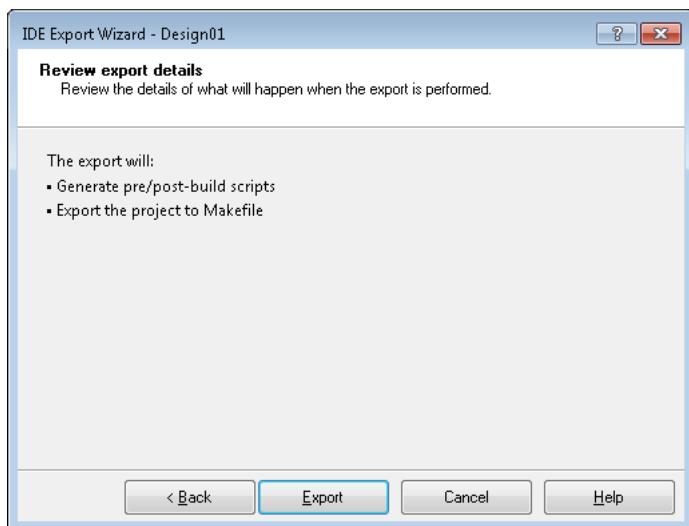


Select which non-generated code to export to the project.

Note Only the files that are compatible with the selected toolchain will appear in the dialog. The wizard adds references to the selected files to *app_source.mk*.

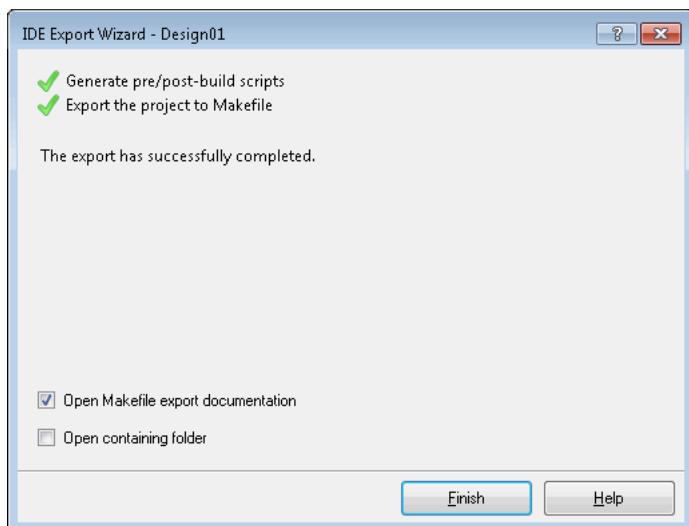
Click **Next >** to continue.

The Review export details step opens.



Review the export details and click **Export**.

The final step opens showing the completed export.



Click **Finish** to close the wizard.

Opening FM0+ Designs in Makefile

Generated Files:

After completion of "Export to Makefile", the following files will be generated:

File	Notes
Makefile	Top-level GNU Make compatible Makefile. This file may be updated or altered as desired.
<i>platform_debug.mk</i> - or - <i>platform_release.mk</i>	Platform and toolchain specific configuration. The Export to Makefile feature will generate <i>platform_debug.mk</i> if Creator is configured to create debug builds, and <i>platform_release.mk</i> if Creator is configured to create release builds. These files may be updated or altered as desired.
<i>app_source.mk</i>	Application firmware source. This file may be updated or altered as desired.
<i>gen_source.mk</i>	PSoC Creator generated source code. This file should NOT be modified. It is automatically re-written by PSoC Creator as a part of the build process. This file is written out for the toolchain selected in the export wizard. Later, it will be updated during each Build process, based on the current toolchain in the Build Settings . That is, if you export to Makefile and change from GCC to MDK, the Makefiles will likely not work. The <i>gen_source.mk</i> file will contain source code that is not supported by the toolchain in <i>platform_debug.mk</i> (or <i>platform_release.mk</i>). Either update the platform configuration file, or use the "Export to Makefile" feature to regenerate the files for the new toolchain.

Important Notes:

- Since the Make utility does not reliably support files with spaces or \$ in the file name, PSoC Creator avoids using them. Also, the tool avoids using colons and slashes in the names of files and folders because some operating systems and drive formats use these characters as volume and directory separators. Furthermore, non-alphanumeric characters may not be supported by all file systems or operating systems. Punctuation marks, parentheses, quotation marks, brackets, and operators, such as following, are often reserved for special functions in scripting and programming languages:

, [] { } () ! ; " ' * ? < > |

Therefore, PSoC Creator checks the design project name and user source files for white spaces and those special characters in their names. PSoC Creator performs the check in the last page of the wizard and the error message specifies the affected files. In this situation you need to address the errors and try to re-export the design.

- The Makefile uses BASH scripts (*prebuild.sh* and *postbuild.sh*) by default. If you would like to use BATCH scripts, you need to modify the Makefile appropriately.
- If you are using Windows, you need CYGWIN (Make package) or MSYS installed in your machine. Cypress has tested this feature on CYGWIN_NT-6.3-WOW64 1.7.33-2(0.280/5/3) i686 cygwin (Make 4.0), and MinGW32_NT-6.1 1.0.18(0.48/3/2) 2012-11-21 i686 Msys (Msys 2013072300).
- If you would like to use another toolchain rather than the one used during the Export, you can modify the *platform_debug.mk*/*platform_release.mk* file to remove the text in front of the TOOLCHAIN_DIR variable and write the path to the target toolchain instead. Be sure to change back slashes to forward slashes in the path.

- You need to install Arm GCC on Linux OS. It can be downloaded from: <https://launchpad.net/gcc-arm-embedded/>
- The additional library files to link, which you added to your design through the build settings dialog (that is, by using the linkers -L and -l arguments), will be populated as linker options in the *platform_debug.mk*/*platform_release.mk* file. You may add more library files to the APP_LIBS section in the *app_source.mk* file.
- If you generate a makefile, but then make significant changes to your design (for example, changing the selected toolchain) and regenerate a new makefile, you must run "make clean" before running "make" to ensure your make-based build is properly up to date.
- PSoC Creator provides the ability to insert pre-build and post-build user commands during project builds. However, these steps are not included in the files generated for integration into third-party IDEs. For generated makefiles, you can add these commands to the top-level makefile, as part of the *prebuild_** and *postbuild_** rules.
- PSoC Creator only propagates project-level Build Settings, such as compiler optimization level. The export process does not support propagation of file-level Build Settings to the makefile.

See Also:

- Exporting a FM0+ Design to Makefile

Keil µVision IDE Notes

There are also various general topics about µVision to be aware of, including:

- [Key IDE Export Files/Projects](#)
- [GCC Settings in µVision](#)
- [PSoC Creator Toolchain Settings](#)
- [Registering MiniProg3/KitProg Drivers](#)
- [Flash Programming/Debugging using MiniProg3](#)
- [Miscellaneous Export Notes](#)

Key IDE Export Files/Projects

The Export to IDE feature generates several important files. These include:

Path/File Name	Purpose	Details
<DesignName>.cydsn/ <DesignName>_<Arch>lib.uvproj (For PSoC 3 only)	µVision library project. This is a static library file that enables access to PSoC Creator Component APIs.	<ul style="list-style-type: none"> ▪ Corresponds to the contents of the “Generated Source” folder in the PSoC Creator design project. ▪ Created in the same directory as the corresponding PSoC Creator design project. It cannot be moved or renamed. ▪ Created, if needed, by the PSoC Creator build process. When creating this file, build settings are copied from PSoC Creator. ▪ If this file already exists, the PSoC Creator build process will update the list of files, but will not touch any other setting. This ensures that firmware developers get updated Component APIs as the PSoC design is changed.
<DesignName>.cydsn/ <DesignName>_<Arch>.uvopt	µVision library options file. This is used by µVision to store some settings, such as breakpoint locations.	<ul style="list-style-type: none"> ▪ Created in parallel with the library project file. ▪ Not modified by PSoC Creator after it has been created.
PSoC 3: <DesignName>.cydsn/ Generated_Source/ PSoC3/post_link.bat PSoC 4/PSoC 5LP: <DesignName>.cydsn/ Export/postbuild.bat	Post-build script file. This is run by µVision after building the application project.	<ul style="list-style-type: none"> ▪ Replaced as a part of every PSoC Creator build. ▪ Patches key information in to the final HEX file. ▪ Checks to make sure that key settings like the selected device, SRAM/flash address and size are in sync between PSoC Creator and the µVision application project.

Path/File Name	Purpose	Details
<p>μVision 4: <code><User_Selected_Path>/<DesignName>.uvproj</code></p> <p>μVision 5: <code><User_Selected_Path>/<DesignName>.uvprojx</code></p>	μVision application project. This links against the library project to create the final HEX file for a design.	<ul style="list-style-type: none"> ▪ Only created/updated by PSoC Creator as a part of the Export to IDE wizard. It may be named/saved to any reasonable location. ▪ When creating this file, the list of files, build settings, and key device information (device name, SRAM/flash address and size) are synchronized with PSoC Creator. ▪ When updating this file, PSoC Creator only updates key device information (device name, SRAM/flash address and size) and updates references to the <code>Generated_Source/<Arch></code> folder in the include path. ▪ When updating this file, PSoC Creator preserves all other settings (including any other directories found in the include path). ▪ The API files in the “Firmware” group in μVision may be freely edited at any time. ▪ The API files in the “Start” group are generated by PSoC Creator and will be overwritten during the PSoC Creator build process. These files contain key device startup code. ▪ The files in the DebugLib and ReleaseLib ensure that the generated library project output files get linked in correctly for the Debug and Release targets.
<p>μVision 4: <code><User_Selected_Path>/<DesignName>.uvopt</code></p> <p>μVision 5: <code><User_Selected_Path>/<DesignName>.uvoptx</code></p>	μVision application options file. This is used by μVision to store some settings, such as breakpoint locations.	<ul style="list-style-type: none"> ▪ Created in parallel with the application project file. ▪ Not modified by PSoC Creator after it has been created.
<code><User_Selected_Path>/postlink.bat</code> (For PSoC 3 only)	Post-build script. This is run by μVision after building the application project.	<ul style="list-style-type: none"> ▪ Created in parallel with the application project file. ▪ Not modified by PSoC Creator after it has been created. ▪ A simple wrapper script that calls the <code>post_link.bat</code> script in the PSoC Creator <code>Generated_Source/<Arch></code> directory.
<code><User_Selected_Path>/<DesignName>_wksp.uvmpw</code> (For PSoC 3 only)	μVision multi-application workspace. This is a μVision multi-application project file that contains references to both the library and application projects.	<ul style="list-style-type: none"> ▪ Created in parallel with the application project file. ▪ Not modified by PSoC Creator after it has been created.
<p>μVision 4: <code><DesignName>.cydsn/<DesignName>_<Arch>.sfr</code></p> <p>(For PSoC 4 and PSoC 5LP only)</p>	μVision System Viewer file. Contains Component register details. This is used by μVision for peripheral register debug. This is the file format expected by Keil μVision for enabling peripheral register debug through its System Viewer.	<p>This file is obtained by converting the Arm CMSIS standard SVD file using the <code>SVConv.exe</code> utility provided along with Arm-MDK.</p> <p>This file is generated along with the μVision library project and updated whenever there is a change in the register map of the design.</p>

Path/File Name	Purpose	Details
Generated CMSIS-Pack: <DesignName>.cydsn/ <DesignName>.gpdsc	Generator Package Description file, is working as the µVision project file. Once it is created, it is shown at PSoC Creator workspace/Results, and gets updated during the build process. If something happened and it cannot be updated, there would be a message in Output window that µVision 5 project is not up to date.	This file is generated as part of export to Generated CMSIS-Pack. The Generator Software Pack (GPDSC file) references the PSoC Creator executable and some of the project files and application files which need to be exported. These files include the files necessary for implementation of PSoC framework, and the user's entire firmware/Component source and header files, except the files of cyboot and design_wide Components as these files will be specified in the pack.
Generated CMSIS-Pack: <DesignName>.cydsn/Export/Pack/ <vendor.packName.version>.pack	This is the generated/exported pack which will be installed in µVision. If you remove the installed version of µVision the installed pack will be removed too, so we store the pack file in the project directory to save it. Once it is generated it will be updated during the build process, if something happened and it cannot be updated, there would be a message in Output window of Creator that µVision 5 project is not up to date.	This file is generated as part of export to Generated CMSIS-Pack. The exported software pack includes the system configuration files, system startup files, library files, flash algorithms, post-build commands, and the project documentation. This includes a Package Description (PDSC) file which describes the contents of the pack and its dependencies to other software packs.

See Also:

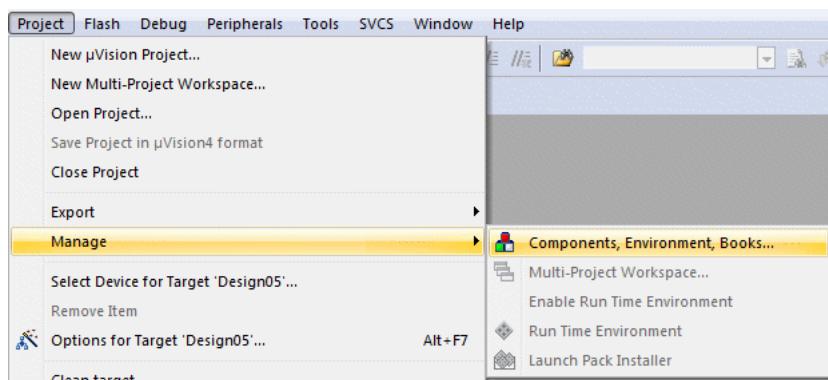
- [Exporting a PSoC 4/PSoC 5LP Design to Keil µVision IDE](#)
- [Exporting a Design to Generated CMSIS-Pack](#)
- [Exporting a PSoC 3 Design to Keil µVision IDE](#)

GCC Settings in µVision

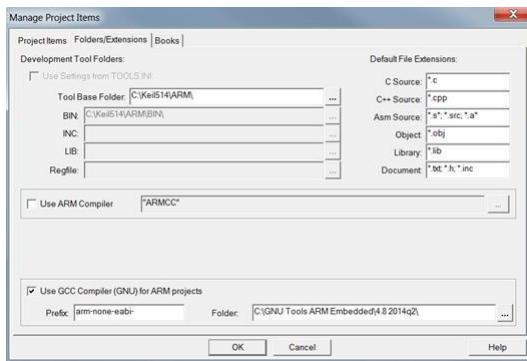
For projects using the GCC toolchain, you must set up a few options in µVision after opening the project.

Note This section is not applicable to CMSIS-Pack projects.

1. Select the Project menu and point to **Manage > Components, Environments, Books...**

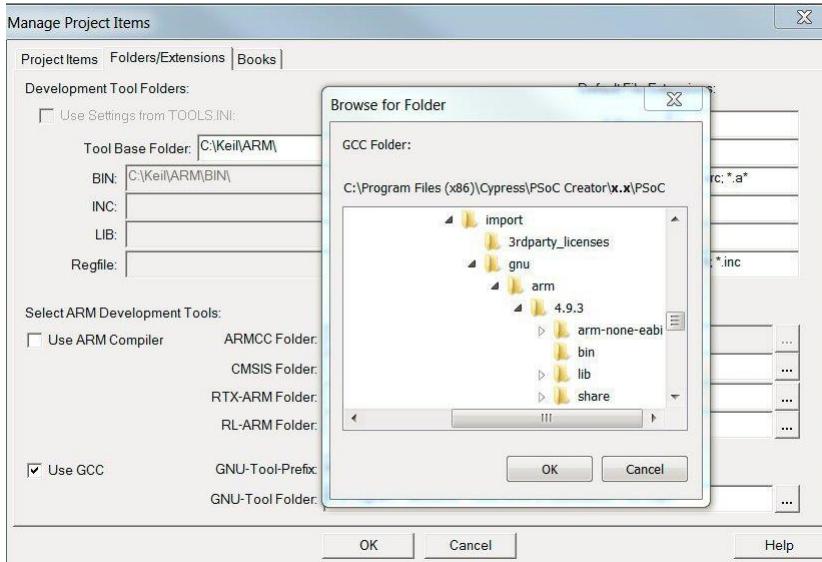


2. On the Components, Environment, and Books dialog, select the **Folders/Extensions** tab and make sure the **Use GCC** check box is selected. Notice that changing the toolchain will reset all Options for Target settings.



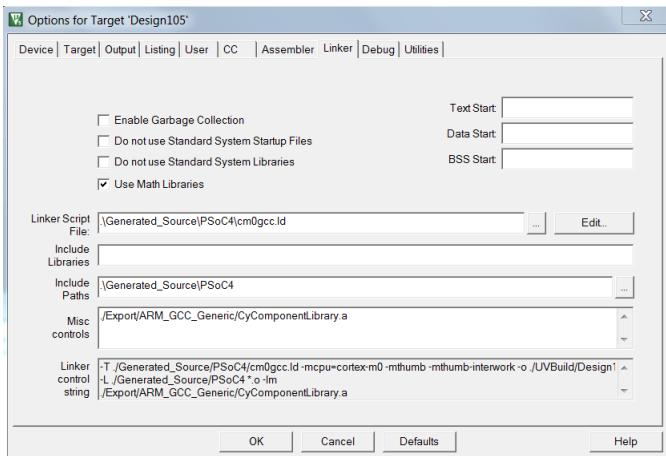
3. Click the ellipsis [...] button and browse to the directory above the "bin" directory in your GNU GCC installation. The path will be similar to the following:

C:\Program Files (x86)\Cypress\PSoC Creator\x.x\PSoC Creator\import\gnu\arm\4.9.3



4. Click **OK** to close the Browse dialog, and then click **OK** to close the Components, Environment, and Books dialog.

5. Right-click on the project and select **Options**, then select the **Linker** tab.

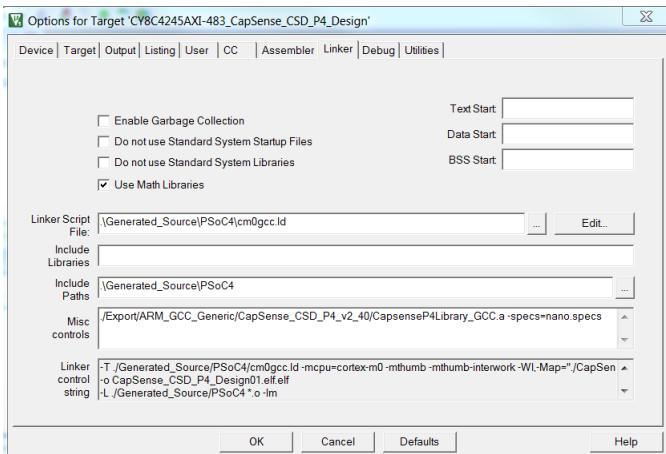


- Deselect the **Do not use Standard Startup Files** check box.
- Add **\Generated_Source\PSoC4** to the **Include Paths** field.
- Click **OK** to close the dialog.

6. If the project has any additional libraries, add them in the **Misc controls** section on the **Linker** tab. Notice that the path to library files in the **Misc controls** section must be separated by forward slashes to be recognized by μVision.

Also add the following command line option in the **Misc controls** section.

`-specs=nano.specs`



Note When Using μVision 4, you may get the following error while building the project:

assembling CyBootAsmGnu.s...

```
C:\Program Files (x86)\Cypress\PSoC Creator\3.3\PSoC Creator\import\gnu\arm\4.9.3\bin\arm-none-eabi-as: unrecognized option `--pd'
```

...

```
arm-none-eabi-gcc: error: ./uvbuild/cybootasmgnu.o: No such file or directory
```

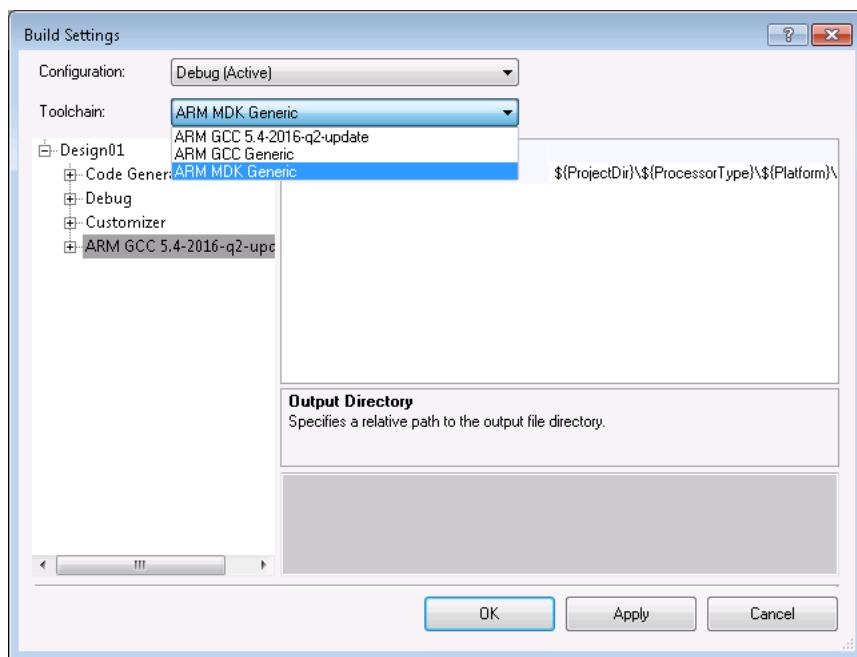
Removing NDEBUG from the Define field in the Conditional Assembly Control Symbols section on the Assembler tab will resolve the problem.

Note If you try to build your exported design in µVision and get the error that some section will not fit in region `rom' or region `rom' overflowed, you need to add the command line option: `-Wl,--gc-sections` to the **Misc controls** section to eliminate unused code and data from firmware binaries.

PSoC Creator Toolchain Settings

In order to use the Arm MDK Generic or DP8051 Keil Generic toolchains in a PSoC Creator project, you need to specify the appropriate toolchain settings.

1. As a one-time step for all PSoC Creator projects, open the [Options dialog](#), and navigate to **Project Management > Generic Toolchains**.
 2. Specify the appropriate binary directory for the Arm MDK Generic (for PSoC 4 and PSoC 5LP) and/or DP8051 Keil Generic (PSoC 3) toolchains.
- These are the binaries you have installed as part of your toolchain applications.
3. Then for each PSoC Creator project, open the Build Settings dialog, and choose the appropriate generic toolchain to use.



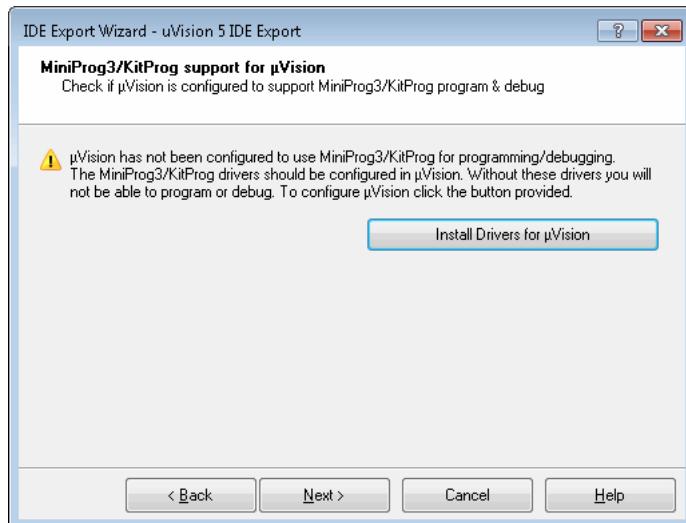
See Also:

- [Exporting a PSoC 4/PSoC 5LP Design to Keil µVision IDE](#)
- [Exporting a PSoC 3 Design to Keil µVision IDE](#)
- [Project Management Options](#)
- [Build Settings](#)

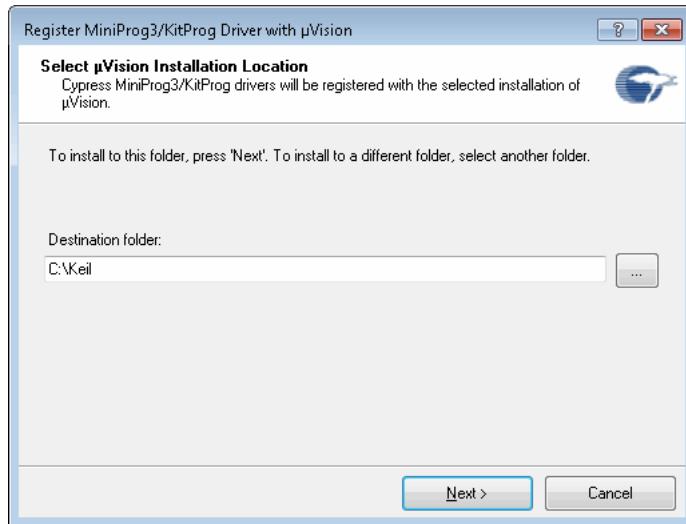
Registering MiniProg3/KitProg Drivers

MiniProg3/KitProg drivers enable the µVision programming/debugging interface to communicate with the device using the MiniProg3 or any KitProg-based hardware assist. This configuration process updates the µVision *tools.ini* file with entries in the [C51], [Arm], and [ArmADS] sections with paths to DLLs in the PSoC Creator install directory.

1. The first time you run the [IDE Export Wizard](#), PSoC Creator will prompt to install the drivers.



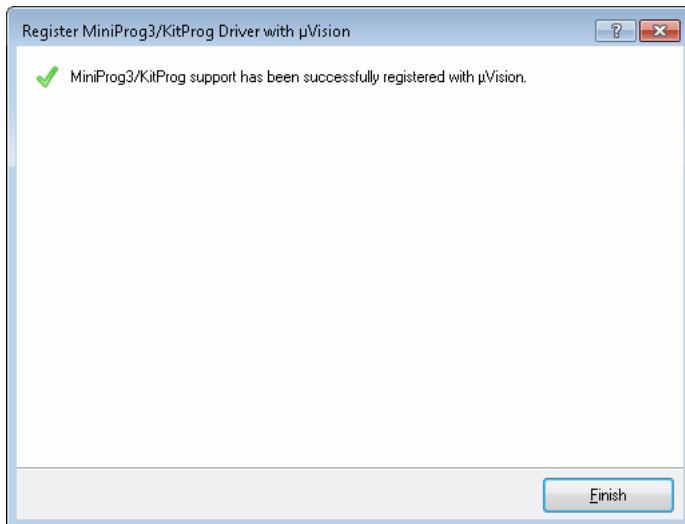
2. Click the **Install Drivers for µVision** button and the registration window will open.



You can also open this dialog from the PSoC Creator **Tools** menu.

3. Type the path or click [...] and navigate to the path to the Keil installation. This is the path to the directory that contains the µVision *tools.ini* file.
 - For µVision 4 IDE, this is usually C:\Keil.
 - For µVision 5 IDE, this is usually C:\Keil_v5.
4. Click **Next >**.

A confirmation screen will display to indicate that the driver was successfully registered.



Note If there is no *tools.ini* file in the selected directory, an error will display. Make sure you have chosen the correct directory.

5. When complete, click **Finish**.

Flash Programming/Debugging using MiniProg3

Flash programming/debugging can be done in μVision using the MiniProg3 or any KitProg device (FTK-3, FTK-5, DVK-030, DVK-050, etc.) (Hardware Assist). A driver used for this purpose can be selected in μVision.

The MiniProg3/KitProg driver implements the μVision interface called AGDI (Advanced Generic Debugger Interface) that interfaces directly with the Keil μVision programmer/debugger.

Note Cypress provides a flash loader for the Keil ULink2 debugger probe for PSoC 5LP. This flash loader allows Keil μVision to use the Keil ULink2 to program and debug the PSoC 5LP device. However, the flash loader does not support programming the NVL's of the device, so you must program the device once using the MiniProg3 in order to set those values.

For flash programming, the following functions are supported:

- Erasing All
- Downloading Flash, NVL &Protect
- Verifying Flash, NVL & Protect
- Resetting the device after programming

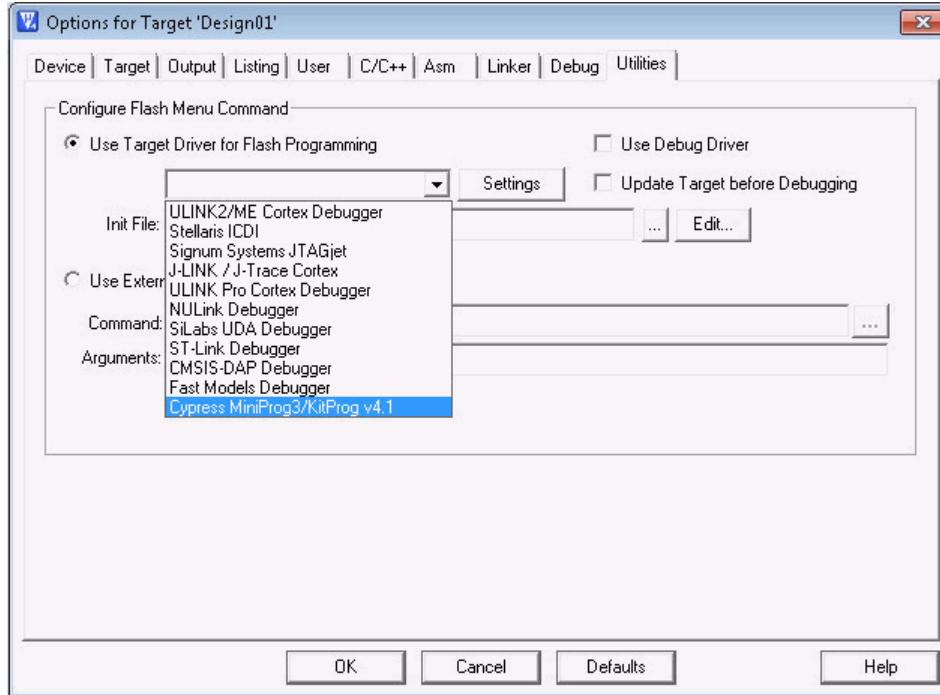
For debugging, the following features are supported:

- Start/Stop/Reset/Step In/Step Out/Run to Cursor
- Insert/Remove Breakpoints
- Enable/Disable Breakpoints

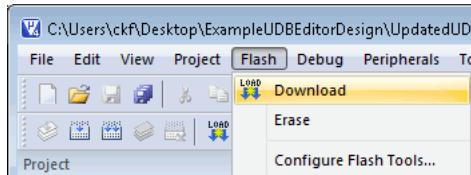
- Disable All Breakpoints
- Kill All Breakpoints
- Read/Write Registers
- Read/Write Memory

For flash programming in µVision, before initiating the flash download, select the “Cypress MiniProg3/KitProg vX.Y” driver from the drop-down menu shown below. The settings button will bring up a dialog that can be used to select different operations to be performed when you click the “LOAD” button.

- Erase Flash
- Program Flash
- Verify Flash
- Reset and Run



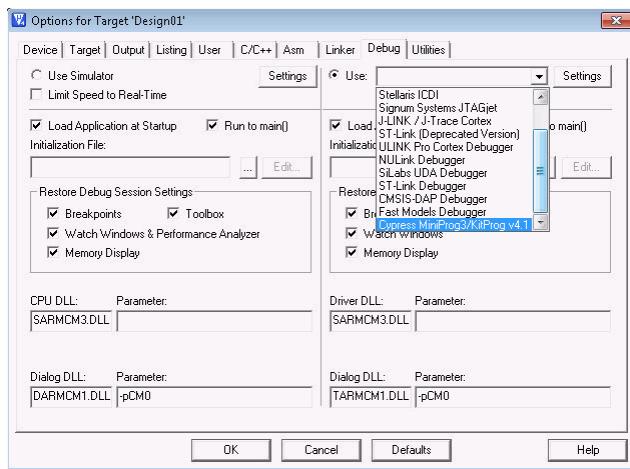
Select the **Flash > Download** menu option to initiate flash programming as shown in the following image.



The µVision debugger will be configured to use the MiniProg3/KitProg debug driver by selecting the “Cypress Miniprog3/KitProg vX.Y” driver in the **Debug** tab, as shown in the following image. The “Settings” button will bring up a dialog that contains the following settings:

- Protocol: SWD/JTAG
- Target Voltage

- Clock speed
- Programming Mode: Reset/Power Cycle
- Active Port: 10 pin/5 pin



Initiate debugging by clicking the “Start/Stop Debug session” under the **Debug** menu. You can save these debug settings by clicking the **Save All** button in the μVision IDE.

Refer to the μVision IDE documentation.

Miscellaneous Export Notes

This section identifies certain key points that you should know:

Notes for All μVision Versions

- There is no support for programming/debugging simultaneously through multiple MiniProg3/KitProg devices connected to USB ports.
- To ensure the MiniProg3/KitProg drivers are successfully registered, verify that the *Tools.ini* file includes the paths to MiniProg3 drivers under the “C51”, “Arm”, and “ArmADS” sections. The following example shows entries in the *Tools.ini* file for the default installation of PSoC Creator:

```
[Arm]
TDRV12=<INSTALL_DIR>\PSoC
Creator\export\ide\uVision\4.x\driver\cyuvdriver_8051.dll ("Cypress
Miniprog3/KitProg v<version>")
[ArmADS]
TDRV12=<INSTALL_DIR>\PSoC
Creator\export\ide\uVision\4.x\driver\cyuvdriver_arm.dll ("Cypress Miniprog3/KitProg
v<version>")
[C51]
TDRV9=<INSTALL_DIR>\PSoC
Creator\export\ide\uVision\4.x\driver\cyuvdriver_arm.dll ("Cypress Miniprog3/KitProg
v<version>")
```

The number after “TDRV” can vary depending on the μVision installation. Also, the version number in the path and name of the driver will change based on the installed version of PSoC Creator.

- If the `tools.ini` file is not updated with the path to the MiniProg3/KitProg drivers, the file can be manually updated with the appropriate path.
- Firmware code written in PSoC Creator before the µVision application project is created for the first time will be synced. Once the µVision application project has been created, build settings and files are maintained in the µVision GUI.
- The Export to IDE feature only supports the following PSoC Creator project variants:
 - PSoC 3 projects may use the "DP8051 Keil 9.51" or "DP8051 Keil Generic" toolchain.
 - PSoC 5LP projects may use the "Arm MDK Generic", "GCC 4.9-2015-q1-update", or "GCC Generic" toolchain.
 - All projects must either be "Normal" or "Bootloadable". You can verify your project variant by looking in Build Settings -> Code Generation and examining the "Application Type" setting.

Note Regular Bootloadable projects are supported, but bootloadable projects that target multi-application bootloaders are not.

- Code in the "Start" group in the µVision application project is automatically re-generated by PSoC Creator.
- µVision post-build steps are unable to report errors in a way that influences the final error/warning count. You should examine your µVision output window to make sure that there are no issues reported by the post-build scripts.
- Windows BAT files do not work reliably when run from UNC style paths (\server\path\to\script.bat). In particular, the BAT file processor may report that it can't set the current working directory to \server\path. You may work around this issue by using the Windows "Map Network Drive" feature to assign a drive letter to your shared server directory.
- µVision performs dynamic syntax checking to validate the program and detect potential code violations in real time. However, sometimes when a function declaration is nested in several layers of header files, µVision dynamic syntax checking might not find it. In these cases, you may see some warnings such as implicit function declaration is not valid. If you are able to build your project in µVision, you can ignore these warnings.

Notes for µVision 5 generated software pack

- There is a dependency between `<project_name>.gpdsc` and the generated pack, which lets µVision select the appropriate generated pack for the project. The selected packs are shown in the Manage Run-Time Environment dialog in µVision. If your exported project does not include any application file or any Component on the schematic (non-design-wide Component), this dependency is not resolved and the CMSIS CORE and proper generated pack will not be automatically selected. Therefore, when you build your project, you may see some errors during the link step as follows:

error: L6236E: No section matches selector - no section to be FIRST/LAST.

Not enough information to list image symbols.

Not enough information to list the image map.

To fix this problem, open the Manage Run-Time Environment dialog in µVision and manually select the CMSIS CORE, Device Startup, and Device Header (choose the Variant for the exported pack). Click **OK** and save the project. Then try to build the project.

- It is not possible to work on a project in PSoC Creator and µVision 5 simultaneously. You need to close µVision before reloading files modified by PSoC Creator. If you change your design in PSoC Creator and build it, PSoC Creator automatically updates the pack. If accidentally you have µVision open while changed the design in

Creator, µVision might use the out of date pack, so you have to close it and re-open it to access the updated pack.

- For µVision 5 exported projects, it is very important to keep `<project_name>.gpdsc` file and `<vendor_name.pack_name.pack_version>.pack` in sync. If you export a project to µVision 5 and select the **Install the generated pack in µVision** check box, the pack will be installed in the µVision installation directory automatically. Also a copy of the generated pack will be located under the `<project_directory>/Export/Pack` folder and a `<project_name>.gpdsc` file will be located in project directory. All three must be kept in sync.
- Once you have a `<project_name>.gpdsc` in the project directory, PSoC Creator detects it and updates it during the build process. It also detects if you previously selected the **Install the generated pack in µVision** check box, so it updates the pack in Export/Pack folder as well and installs it in µVision. If for any reason PSoC Creator cannot install the pack during the build process (for example, say you uninstalled µVision 5 from your system), you will see a message in PSoC Creator output window stating that the µVision 5 project is not up to date and a re-export is required.
- If you change a previously exported project in PSoC Creator and re-export the project with the same pack name and version, use caution. If you deselect the **Install the generated pack in µVision** check box (where previously you selected it), the pack in the Export/Pack folder is substituted with the up to date pack in sync with new gpdsc file. However, the installed pack in µVision is out of date (still the previous one), so you need to manually install the new pack before building the project in µVision 5. You can install the pack by double-clicking on it and following the dialog steps.
- There can be more than one pack in the Export/Pack directory, but the gpdsc file in the project directory is always in sync with the last generated pack. So, you may save the current exported µVision 5 project somewhere else to keep the current gpdsc file for the current pack file before changing the design and doing a new export.
- Follow the manual steps to refer to the linker script file and post build command in the µVision Options for Target dialog (See Opening Projects in µVision 5 IDE). If you do not add the linker script and try to build the project in µVision, you will see an error similar to the following:

```
.\UVBuild\ADC_Differential_Preamplifier22.axf:
```

```
Error: L6320W: Ignoring --entry command. Cannot find argument 'Reset_Handler'.
```

```
.\UVBuild\ADC_Differential_Preamplifier22.axf:
```

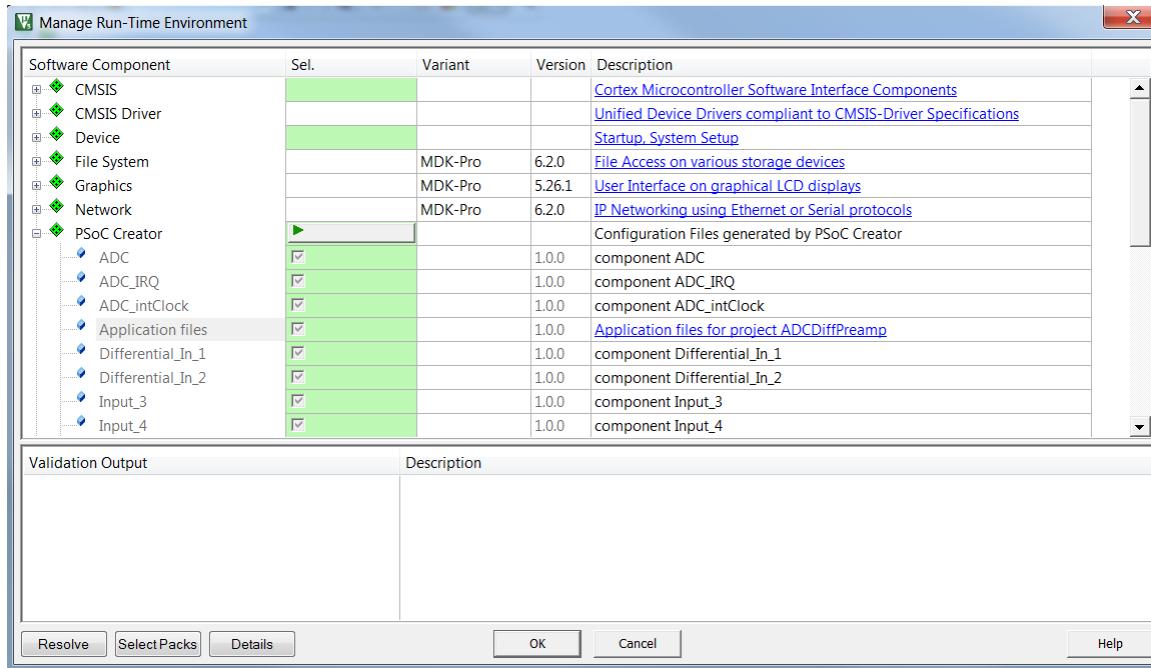
```
Warning: L6320W: Ignoring --first command. Cannot find argument '__Vectors'.
```

```
Not enough information to list image symbols.
```

```
Not enough information to list the image map.
```

- The GCC toolchain is not supported by CMSIS Pack µVision 5.

- If you added a pdf file as an application file during the export process, the pdf file will appear in the Manage Run Time Environment dialog. The other application files will appear under the PSoC Creator in µVision workspace. The following snapshot shows the pdf file named "Application file" in the Manage Run Time Environment dialog. If you click on the link under description, the pdf file will open.



- If you do not choose to add the application files automatically, you can add them manually to the Source Group in the µVision workspace. If you already added the application files automatically to the pack and also add them manually, you will get a build error similar to the following:

.\UVBuildADC_Differential_Preamplifier22.axf:

Error: L6200E: Symbol __asm__6_main_c_5f9501ff____REV16 multiply defined (by main_1.o and main.o).

See Also:

- [Exporting a PSoC 4/PSoC 5LP Design to Keil µVision IDE](#)
- [Exporting a PSoC 3 Design to Keil µVision IDE](#)

3rd Party Bootloader Support

This section contains the following topics:

- [Eclipse Bootloader Support](#)
- [IAR Bootloader Export Support](#)
- [µVision Bootloader Export Support](#)

Note For PSoC 6 devices, refer to the Bootloader SDK guide and related collateral.

Eclipse Bootloader Support

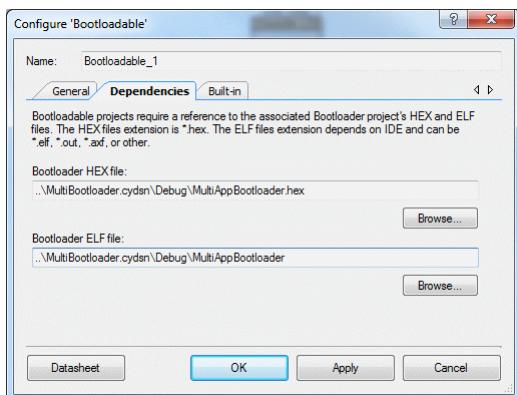
Note These steps apply to all Bootloader/Bootloadable type projects, including Launcher and Combination projects.

Note For PSoC 6 devices, refer to the Bootloader SDK guide and related collateral.

Bootloader/Bootloadable Project Export

This section describes the steps necessary to export a single application Bootloader/Bootloadable project for Eclipse IDE.

1. Export the bootloader project as usual. See [Exporting a Design to Eclipse IDE](#).
2. Import and build bootloader project in Eclipse.
3. In PSoC Creator, configure the Bootloadable Component for the bootloadable project to point at the hex/out files of the bootloader project in the Eclipse output directory.

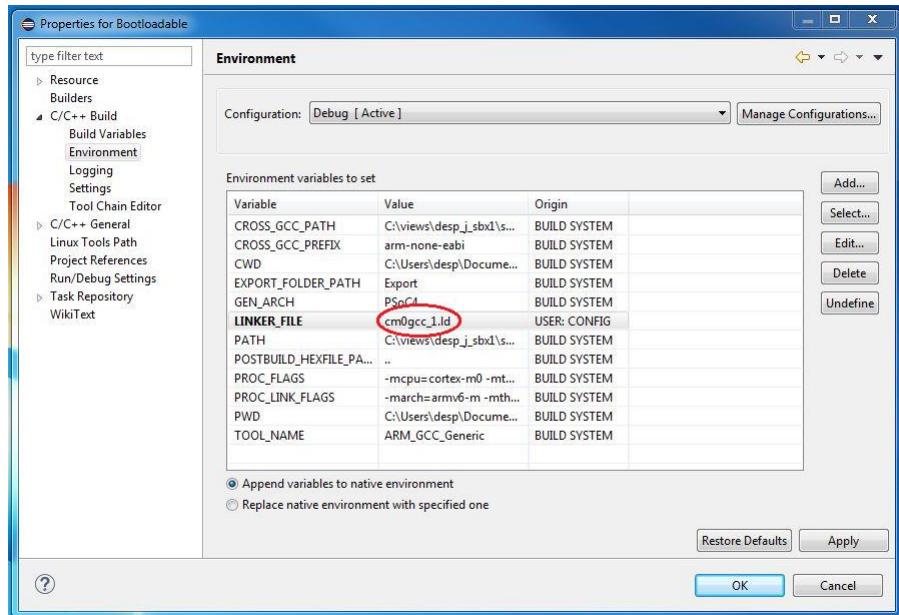


4. Export bootloadable project.

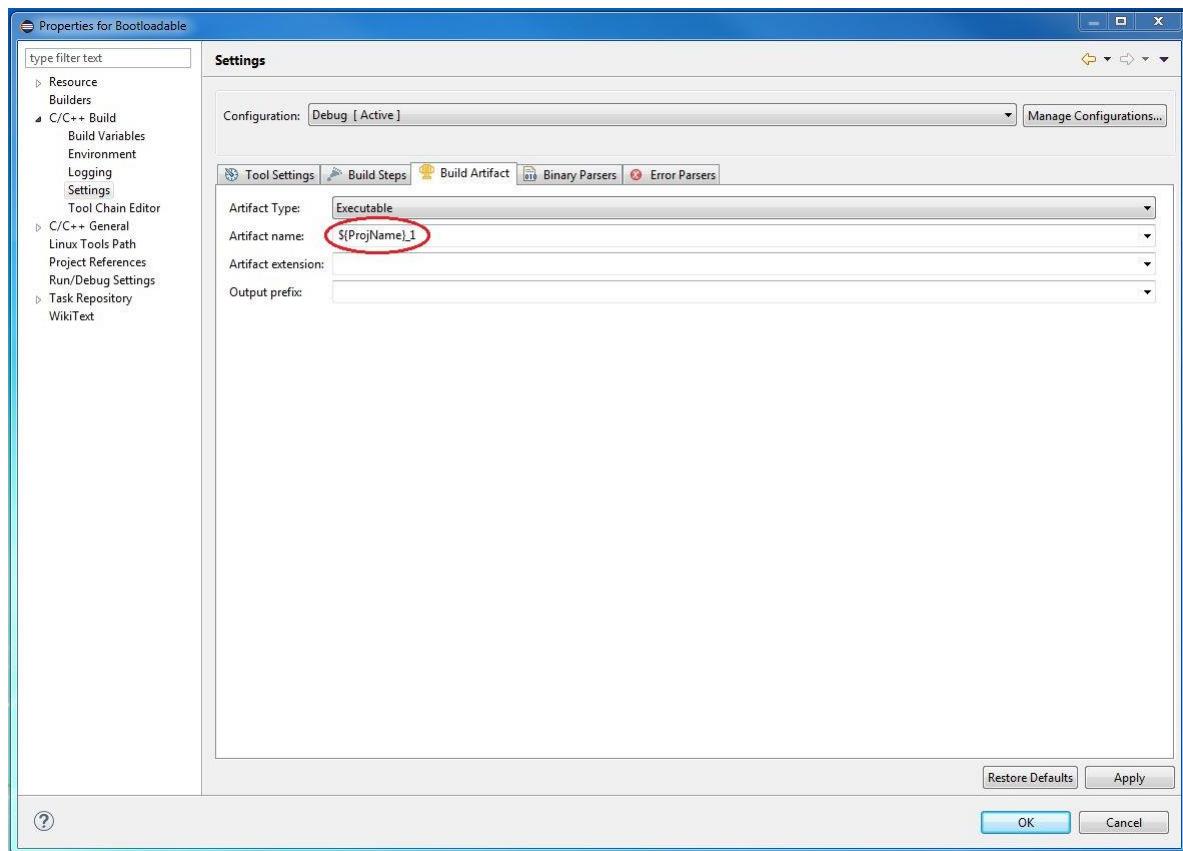
Multi-Application Bootloader/Bootloadable Support

This section describes how to import a PSoC Creator multi-application bootloader project for use in the Eclipse IDE.

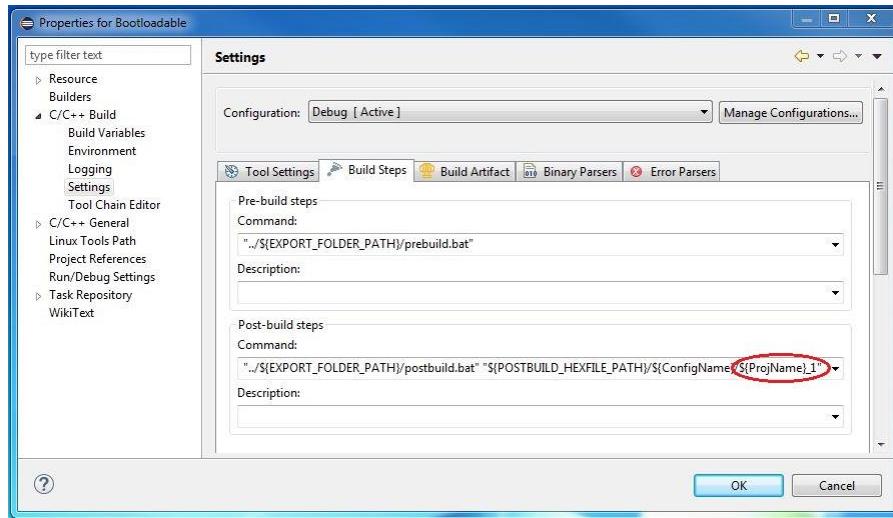
1. Export Bootloader and Bootloadable projects as usual. See [Bootloader/Bootloadable Project Export](#). Import them into Eclipse as separate projects.
2. In Eclipse, modify the Eclipse Bootloadable project's settings as follows:
3. Update the project's LINKER_FILE setting by appending "_1" before the ".ld" extension.



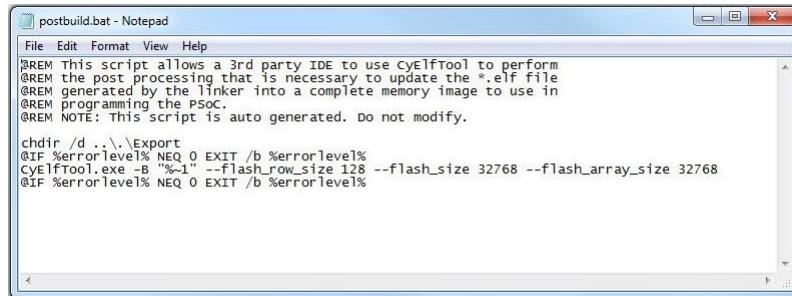
4. Update the Project's Artifact name by appending "_1".



5. Update Project Post-build steps by appending "_1".



6. Build Project
7. Repeat step 2.a to step 2.d, but replace "_1" with "_2".
8. Then, use *CyElfTool.exe* to create a combination hex file that has the bootloader and both bootloadable images in it.
9. Open the *postbuild.bat* file found in the \Export directory.



10. Note the `-flash_row_size`, `flash_array_size`, `flash_size` command line arguments. Note the `-ee_array` and `-ee_row_size` arguments as well, if they exist. You will use these values in the command line shown in step 3.d. below.
11. In a Windows command line, cd into the .cydsn directory of your bootloadable project.
12. Run the *CyElfTool.exe* tool using a command similar to the following:

```
Export\CyElfTool.exe -M Debug\{ProjectName}_1 Debug\{ProjectName}_2
Debug\{ProjectName}.hex --flash_row_size 128 --flash_size 32768 --
flash_array_size 32768
```

Note Flash and EE arguments are device-dependent and can be copied from the bootloadable's *postbuild.bat* file.

Note If you use the release configuration, make sure to change all instances of Debug in the above command line to Release. The above *CyElfTool.exe* command will generate a combination hex file that contains the bootloader image and both bootloadable images. It will have name *{ProjectName}.hex*.

13. You should now have two bootloadable .cyacd files, a bootloader elf file, and a combination hex file in the three Eclipse project Debug directories listed above. You can now program and debug your bootloader, either bootloadable or combination files as you wish.

IAR Bootloader Export Support

This section describes how to export a PSoC Creator bootloader project for use in the IAR IDE. Included are:

- [Bootloader/Bootloadable Project Export](#)
- [Multi-Application Bootloader/Bootloadable Project Export](#)

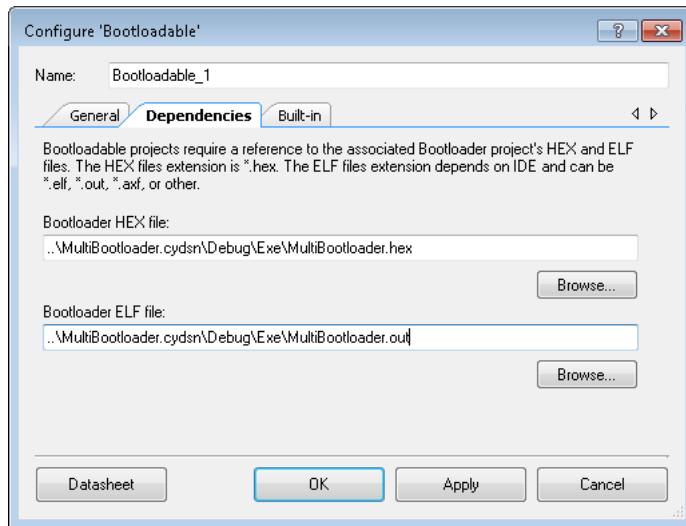
Note These steps apply to all Bootloader/Bootloadable type projects, including Launcher and Combination projects.

Note For PSoC 6 devices, refer to the Bootloader SDK guide and related collateral.

Bootloader/Bootloadable Project Export

This section describes the steps necessary to export a single application Bootloader/Bootloadable project for IAR IDE.

1. Export the multi-application bootloader project as usual. See [Exporting a Design to IAR IDE](#).
2. Import and build the bootloader project in IAR.
3. In PSoC Creator, configure the Bootloadable Component for the bootloadable project to point at the hex/out files for the multi-application bootloader in the IAR output directory.

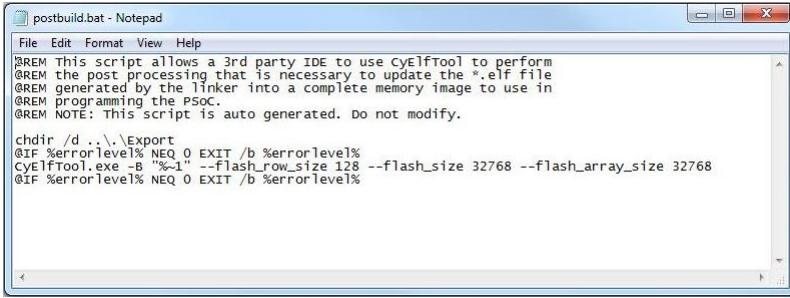


4. Export bootloadable project.
5. Import the bootloadable project into the IAR IDE.

Multi-Application Bootloader/Bootloadable Project Export

This section describes the steps necessary to export a multi application Bootloader.

1. Follow steps 1-3 under [Bootloader_Bootloadable_Project_Export](#).
2. Export the bootloadable project as usual, **except** with the following changes:
3. When naming your IAR bootloadable project, append "_1" to the end of the name.
4. When selecting the linker configuration file, make sure to select the one with the _1 in its name.
5. Build the project as usual.
6. Export a second IAR bootloadable from the same bootloadable project with the following changes:
7. When naming your IAR bootloadable project, append "_2" to the end of the name.
8. When selecting the linker configuration file, select the .icf with an _2 in its name.
9. Build the new project as usual.
10. Then, use *CyElfTool.exe* to create a combination hex file that has the bootloader and both bootloadable images in it.
11. Open the *postbuild.bat* found in the export directory.



```

postbuild.bat - Notepad
File Edit Format View Help
REM This script allows a 3rd party IDE to use CyElfTool to perform
REM the post processing that is necessary to update the *.elf file
REM generated by the linker into a complete memory image to use in
REM programming the PSoC.
REM NOTE: This script is auto generated. Do not modify.

chdir /d ..\.\Export
@if %errorlevel% NEQ 0 EXIT /b %errorlevel%
CyElfTool.exe -B "%-1" --flash_row_size 128 --flash_size 32768 --flash_array_size 32768
@if %errorlevel% NEQ 0 EXIT /b %errorlevel%

```

12. Copy the `-flash_row_size`, `flash_array_size`, `flash_size` command line arguments. Copy the `-ee_array` and `-ee_row_size` arguments as well, if they exist.
13. In the Windows command line, cd into the `.cydsn` directory of your bootloadable project.
14. Run the *CyElfTool.exe* using a command similar to the following:


```

Export\CyElfTool.exe -M Debug\Exe\{ProjectName}_1.out
Debug\Exe\{ProjectName}_2.out Debug\Exe\{ProjectName}.hex --flash_size 262144 --
flash_row_size 256 --ee_array 64 --ee_row_size16

```

Note Flash and EE arguments are device-dependent and can be copied from the bootloadable's *postbuild.bat* file.

Note If you use the release configuration, make sure to change all instances of Debug in the above command line to Release. The above *CyElfTool.exe* command will generate a Combination hex file that contains the bootloader image and both bootloadable images. It will have name `{ProjecName}.hex`.
15. You should now have two .out files and one hex file in the UVBuild directory. You can now program and debug your bootloader, either bootloadable or combination files as you wish.

μVision Bootloader Export Support

This topic describes how to export a PSoC Creator bootloader project for use in the μVision IDE. Included are:

- [Bootloader/Bootloadable Project Export](#)
- [Multi-Application Bootloader/Bootloadable Project Export](#)

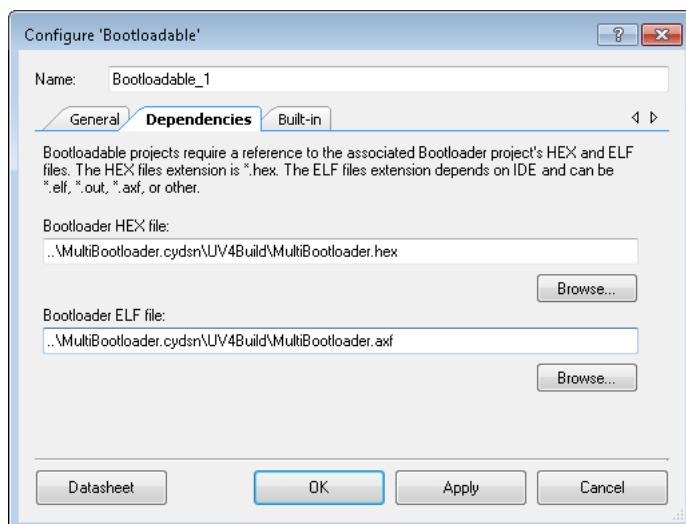
Note For PSoC 6 devices, refer to the Bootloader SDK guide and related collateral.

Bootloader/Bootloadable Project Export

This section describes the steps necessary to export a single application Bootloader/Bootloadable project for μVision IDE.

This section describes how to export a PSoC Creator multi-application bootloader project for use in the μVision IDE.

1. Export the bootloader project as usual. See [Exporting a Design to Keil μVision IDE](#).
2. Import and build the bootloader project in μVision.
3. In PSoC Creator, configure the Bootloadable Component for the bootloadable project to point at the hex/axf files of the bootloader in the μVision output directory.



4. Export the bootloadable project.
5. Import the bootloadable project into μVision.

Multi-Application Bootloader/Bootloadable Project Export

μVision

This section describes how to export a PSoC Creator multi-application bootloader project for use in the μVision IDE.

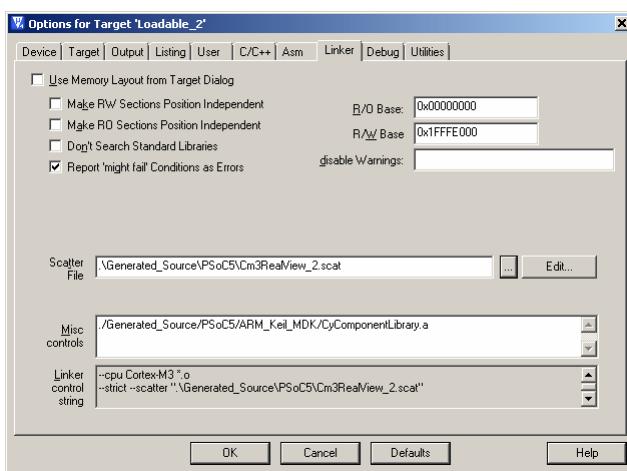
1. Follow steps 1-3 under [Bootloader/Bootloadable Project Export](#).
2. Export the bootloadable project as usual. This will automatically create a project file with the name `{ProjectName}_1.uvproj`.
3. Build the bootloadable project as usual in μVision.

4. Create a second project in **μVision** as follows:
5. Open a command line and navigate to your bootloadable project's .cydsn directory.
6. Use the **μVision** command line to re-export the {ProjectName}.xml.

The command should be similar to the following:

```
{KeilInstallDir}\UV4\Uv4.exe {ProjectName}_2.uvproj -t {ProjectName}_2 -i
".\Export\{ProjectName}.xml" -c
"\PSoCCreatorInstallDir}\dev\CyPSoC.cdb"
```

7. Select **Project > Options for Target** to open the dialog, and select the **Linker** tab.
8. In the **Scatter File** text box, change the _1 of the scatter file to _2.

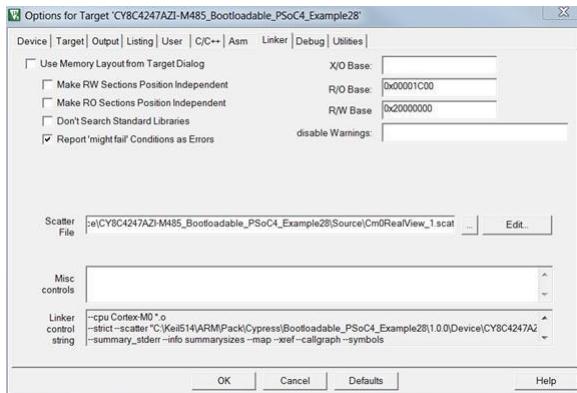


9. Build the new project as usual.

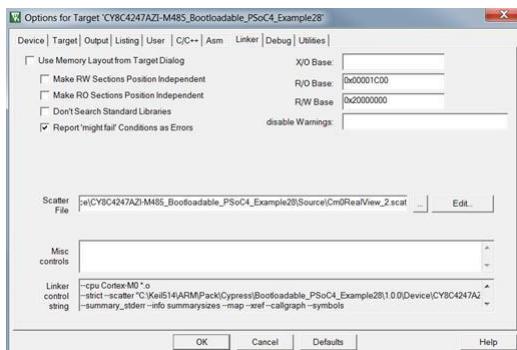
Generated CMSIS-Pack

This section describes how to export a PSoC Creator multi-application bootloader project for use in the Generated CMSIS-Pack.

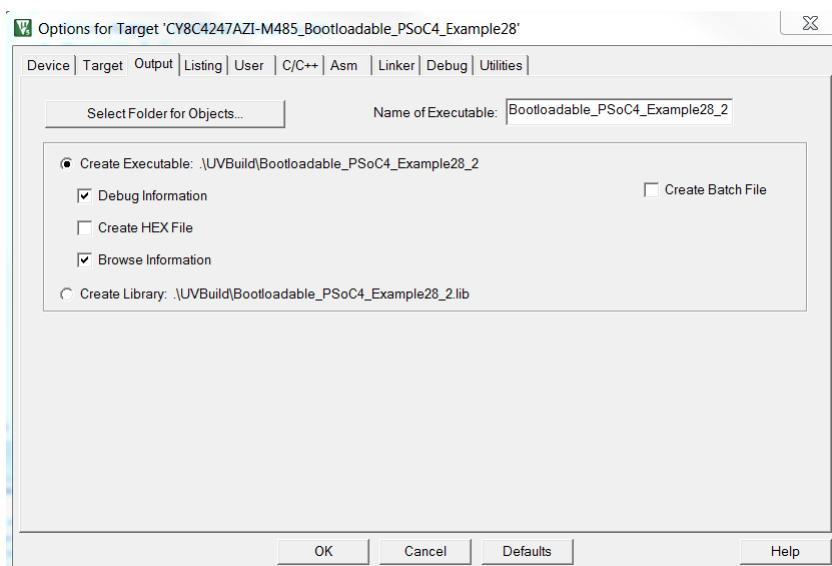
1. Follow steps 1-3 under [Bootloader/Bootloadable Project Export](#).
2. Export the bootloadable project as usual, this will automatically create a project file with the name {ProjectName}_1.uvprojx.
3. Then select **Project > Options for Target** to open the dialog, and select the **Linker** tab. In the **Scatter File** text box, choose the proper linker script with "_1" in the end such as Cm0RealView_1.scat for MDK.



4. Check that the pre-build script appears in the Options for Target dialog under the **User** tab; see [Pre-Build/Post-Build Script](#) section for more information.
5. Follow any other steps required as described in Opening Projects in µVision 5 IDE, and build the bootloadable project as usual in µVision. Then use the following steps:
6. Select **Project > Options for Target** to open the dialog, and select the **Linker** tab. In the **Scatter File** text box, choose the proper linker script with "_2" in the end such as Cm0RealView_2.scat for MDK.



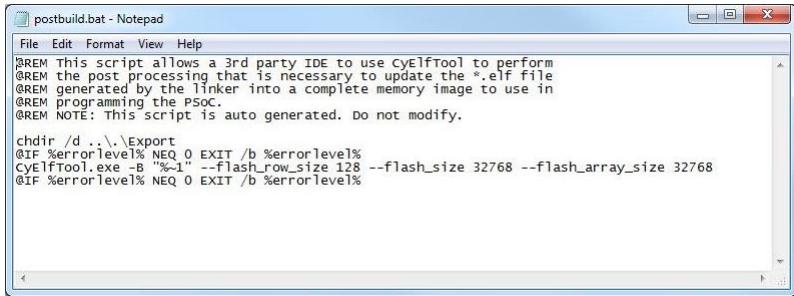
7. Select the **Output** tab and in the **Name of Executable** text box, and change "_1" to "_2" at the end of the project name.



8. Build the bootloadable project again as usual in µVision.

µVision and Generated CMSIS-Pack

1. Use the *CyElfTool.exe* tool to create a combination hex file that has the bootloader and both bootloadable images in it.
2. Open the *postbuild.bat* found in the \Export directory.



```

postbuild.bat - Notepad
File Edit Format View Help
REM This script allows a 3rd party IDE to use CyElfTool to perform
REM the post processing that is necessary to update the *.elf file
REM generated by the linker into a complete memory image to use in
REM programming the PSoC
REM NOTE: This script is auto generated. Do not modify.
chdir /d ..\..\Export
@if %errorlevel% neq 0 exit /b %errorlevel%
cyelftool.exe -B "%~1" --flash_row_size 128 --flash_size 32768 --flash_array_size 32768
@if %errorlevel% neq 0 exit /b %errorlevel%

```

3. Copy the `-flash_row_size`, `flash_array_size`, `flash_size` command line arguments. Copy the `-ee_array` and `-ee_row_size` arguments as well, if they exist.
4. In the Windows command line, cd into the .cydsn directory of your bootloadable project.
5. Run the *CyElfTool.exe* using a command similar to the following:

MDK toolchain:

```

Export\CyElfTool.exe -M UVBuild\{ProjectName}_1.axf UVBuild\{ProjectName}_2.axf
UVBuild\{ProjectName}.hex --flash_size 262144 --flash_row_size 256 --ee_array 64
--ee_row_size16

```

GCC toolchain:

```

Export\CyElfTool.exe -M {ProjectName}_1.elf {ProjectName}_2.elf {ProjectName}.hex
--flash_size 262144 --flash_row_size 256 --ee_array 64 --ee_row_size16

```

Flash and EE arguments are device-dependent and can be copied from the bootloadable's *postbuild.bat* file.

6. You should now have two .axf files and one hex file in the Output directory. You can now program and debug your bootloader, either bootloadable or combination files as you wish.
7. If you make any changes to either of your bootloadable projects, make sure those changes are mirrored in your other bootloadable project.
8. If you re-export a bootloadable project from PSoC Creator, make sure to regenerate your second bootloadable using the manual steps previously described.

7 Programming and Debugging



PSoC Programmer is integrated into PSoC Creator. Whenever you need to program a device and/or launch the debugger, PSoC Programmer is automatically invoked. This saves the time of manually launching and configuring PSoC Programmer as a separate application.

To Configure PSoC Programmer:

The first time you want to program/debug a device, there may need to be configuration options set, depending on the hardware configuration. The following is a typical setup scenario:

1. Connect the device to the computer through the [MiniProg3](#) / KitProg / Kit.
2. As needed, use the [Select Debug Target](#) dialog to configure the device settings. See also, [Device Configuration](#).
3. Launch the programmer/debugger.

To launch the programmer:

On the [Build Toolbar](#), click the **Program** button.

This will build the current project if needed and check that the selected device is compatible with the device specified in the project. PSoC Creator will then invoke PSoC Programmer to program the device. For more details about the programmer, refer to the PSoC Programmer documentation.

To launch the debugger:

On the [Build Toolbar](#), click the **Debug** button.

In addition to programming the device as specified above, this will also initiate a new debug session. For more details about the debugger, see [Using the Debugger](#).

See Also:

- [MiniProg3](#)
- [Select Debug Target](#)
- [Build Toolbar Commands](#)
- [Using the Debugger](#)

MiniProg3

The MiniProg3 is used for both programming and debugging a PSoC device. It supports a number of different transfer protocols, including:

- SWD
- JTAG
- I2C

To Program or Debug a PSoC Device:

Configure the MiniProg3 for SWD or JTAG mode.

You can configure all MiniProg3 devices by setting up the default MiniProg3 configuration in the PSoC Creator preferences ([Program/Debug Options > MiniProg3](#)). You can also configure the attached MiniProg3 by right clicking on its entry in the [Select Debug Target dialog](#). Both configuration options affect all attached MiniProg3s.

Note If the options for the MiniProg3 are configured for the MiniProg3 to supply voltage to the device, and the device is already powered by an external source, the Settings may not match what the MiniProg3 is actually doing.

To Use the MiniProg3:

Ensure it is properly connected.

1. Connect the MiniProg3 to the computer with the provided USB cable.
2. Using the 10-pin ribbon cable, connect the MiniProg3 to the DVK board's processor module.

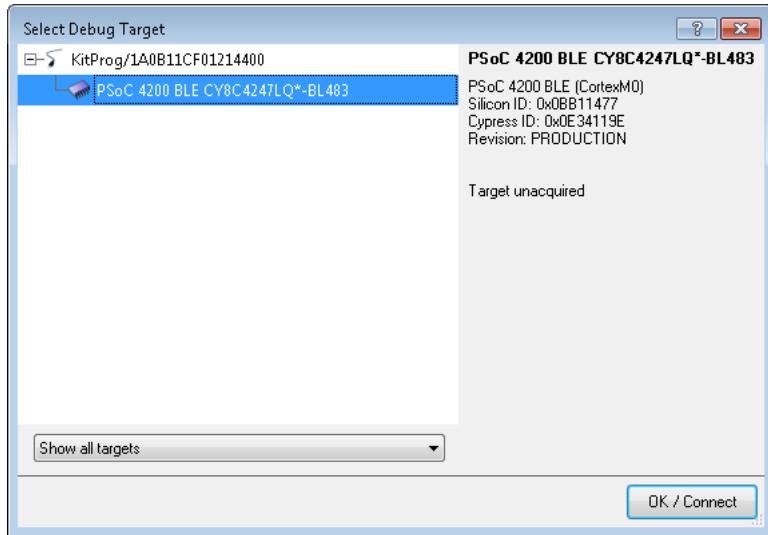
PSoC Creator will automatically detect the newly attached MiniProg3 and PSoC device. You can view both devices in the Select Debug Target dialog, under the MiniProg3 connection type.

See Also:

- [Program/Debug Options](#)
- [Select Debug Target](#)
- [Device Configuration](#)

Select Debug Target

The Select Debug Target dialog allows you to select which attached device you want to program and debug. It also provides a means of configuring the settings used to detect devices.



To Open the Dialog:

Open a project and choose **Select Debug Target** from the **Debug** menu.

This dialog may also display when you first click **Debug**  or **Program** . It will display if you have more than one device connected to your computer and if you have not already selected a device.

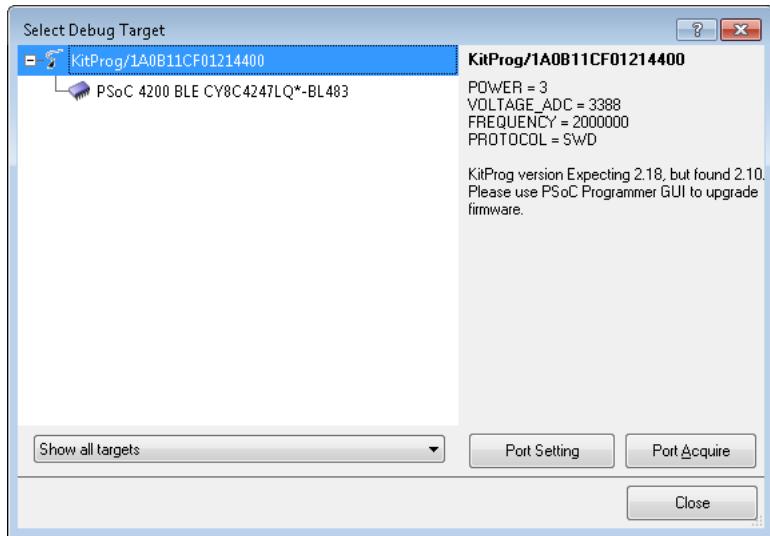
It uses the device type chosen in the [Device Selector](#) in order to filter what you can choose as the debug target. The debugger will remember this selection for all subsequent debug sessions while PSoC Creator is opened.

If the target is connected to multiple devices on a JTAG, it will display the PSoC 3 / PSoC 5LP devices connected in the chain.

If you wish to select a different debug target at any time, open the Select Debug Target dialog by choosing **Debug > Select Debug Target**.

To Reset Attached Devices:

Click on the connection node and click **Port Acquire** button.

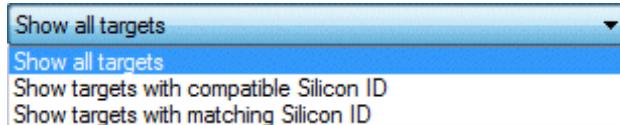


This can be used to reset any attached devices and make sure the debug port is active.

To Filter the Devices Shown:

The devices displayed can be filtered based on the active project. The dialog can be configured to show the following:

- all devices
- devices compatible with the selected device for the active project
- devices that are an exact match of the selected device for the active project

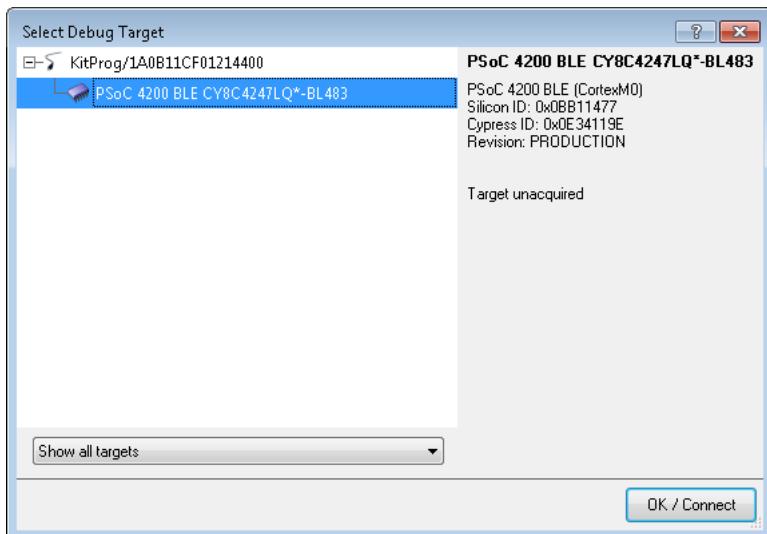


Note To be compatible, the target device must come from the same family and must have all (or more) of the resources as the project's device.

To Select a Device:

Devices can be connected via the MiniProg3 or kits and used within PSoC Creator. Under each MiniProg3 and kit will be a list of devices attached to that MiniProg3 or kit.

Click on the appropriate device from the list and click **OK/Connect** to use that device when programming or debugging.



The debugger will remember this selection for all subsequent debug sessions while PSoC Creator is opened. If you want to select a different debug target at any time, re-open the Select Debug Target dialog.

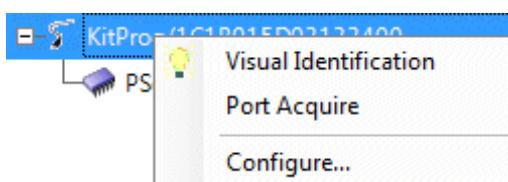
Note The **OK/Connect** command locks the selected device for exclusive use by this session of PSoC Creator.

If the chip was programmed without debugging enabled, clicking **Port Acquire** may be necessary for PSoC Creator to properly read the silicon revision. Doing so will reset the device and place it in a temporary debug state. While in this debug state code will not execute.

If after running **Port Acquire**, PSoC Creator still has trouble recognizing the silicon revision, PSoC Creator may be too old to handle this version of silicon, or the board may be incorrectly wired for the current settings in PSoC Creator. Verify the wiring of the board and the options selected under the [Programmer/Debugger Options dialog](#). For example, if the options for the MiniProg3 are set to use Reset for the Programming Mode, then the MiniProg3's XRES pin must be connected to the PSoC's XRES pin.

Context Menus:

Each node in the displayed tree may have commands accessible via the right-click menu. You can use these commands to configure the communication channel settings, for example.



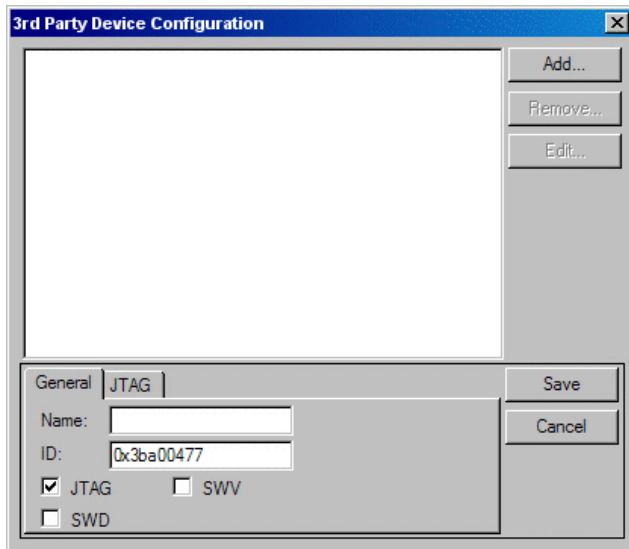
See Also:

- [Using the Debugger](#)

- [Device Configuration](#)
- [MiniProg3](#)
- [QuickProgrammer](#)
- [Programmer/Debugger Options](#)

Device Configuration

The Device Configuration dialog is used to configure PSoC Creator to identify a 3rd party device and report the provided name instead of just the silicon ID.



This is done so the [Select Debug Target dialog](#) can list correct information about devices that are attached to a computer.

Note While this configuration allows PSoC Creator to recognize 3rd party devices, these devices cannot be selected for debugging. The primary use of the Device Configuration dialog is to configure the size of the Instruction Register and Data Register for 3rd party devices attached in a JTAG chain.

To View Device Configuration:

Device Configuration can be viewed in two ways:

- From the **Tools** menu, select the [Options > Program/Debug > Device Recognition](#).
- From the [Select Debug Target dialog](#), right-click on the device node and select **Configure**.

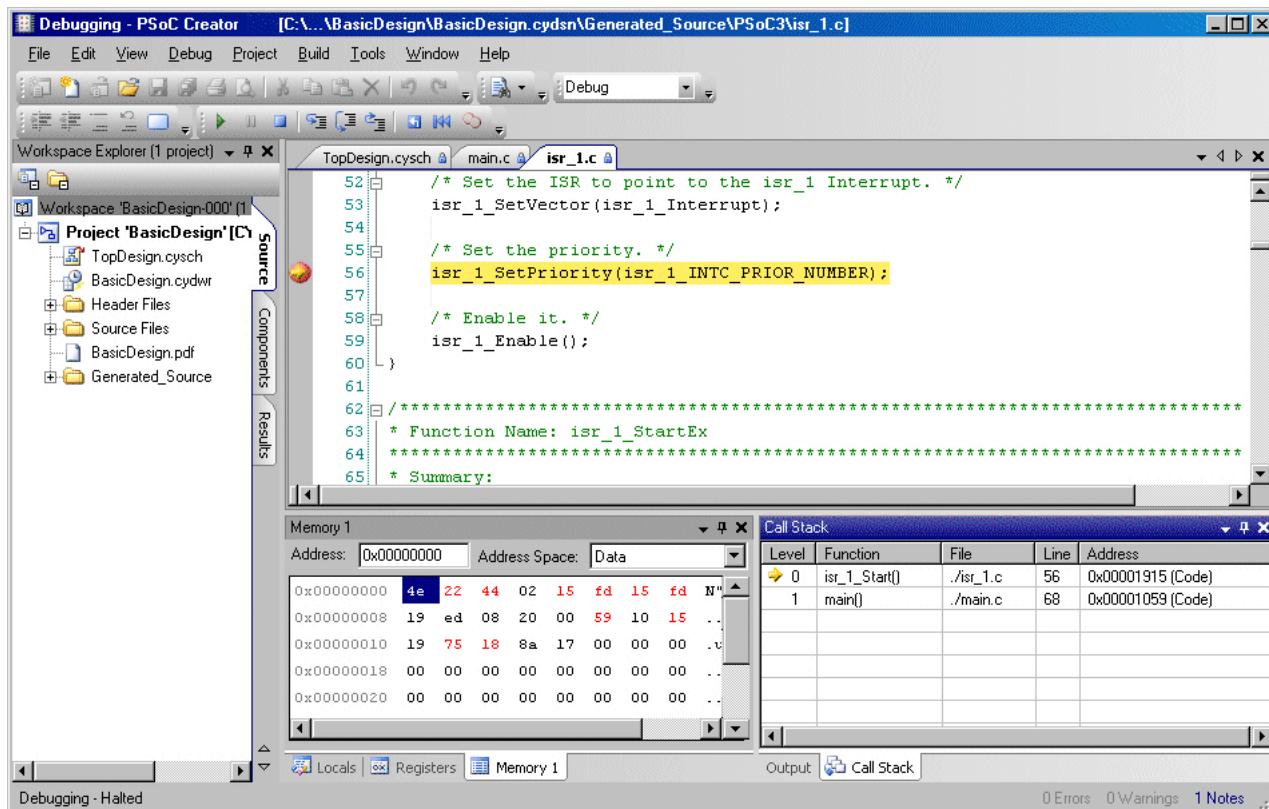
See Also:

- [Using the Debugger](#)
- [Select Debug Target](#)
- [MiniProg3](#)

- [Debugging Options](#)

Using the Debugger

The PSoC Creator debugger allows you to observe the run-time behavior of your program and determine the location of semantic errors.



The debugger understands features that are built into programming languages and their associated libraries. With the debugger, you can break (suspend) execution of your program to examine your code, evaluate and edit variables in your program, view registers, see the instructions created from your source code, and view the memory space used by your application.

Using a debugger, you can examine the content of variables in your program without having to insert additional calls to output the values. You can insert a breakpoint in your code to halt execution at the point you are interested in.

When your program is halted (in break mode), you can examine local variables and other relevant data using facilities, such as the Watch window and the Memory window. For more information, see [Watch Window](#) or [Memory Window](#). Not only can you view the contents while in break mode, you can edit or change the contents, if you desire. In most cases, you will set your breakpoint in a source file, where you write and edit your code. Sometimes, you may choose to set the breakpoint in the debugger's Disassembly window instead. The Disassembly window shows you the instructions created from your source code. For more information, see the [Disassembly Window](#). Unlike **printf** or **MsgBox**, setting a breakpoint does not add an additional functional call to your source code. Therefore, setting a breakpoint is unlikely to change the behavior of the program you are trying to debug.

Note Using the Debugger, you can program your device through PSoC Creator without launching PSoC Programmer. Also, the device will automatically be reprogrammed when you start a debug session.

The PSoC Creator debugger provides menu items, windows, and dialog boxes to access all its tools. You can obtain help on any window, dialog box, or control by selecting the item and pressing [F1]. You can also drag and drop to move debugging information between windows. The following lists the main sub-sections for the debugger:

- [Debugger Commands](#)
- [Debugger Menus](#)
- [Text Editor Context Menu Commands](#)
- [Debugger Windows](#)
- [Error Handling](#)

Supported Debuggers:

PSoC Creator supports the 8051 and Arm Cortex-M0, -M0+, -M3, and -M4 microprocessors. The supported debugger is GDB.

Debugger Toolbar Commands

The Debugger toolbar contains many of the common commands you will use while debugging your programs:



The following table describes the toolbar commands:

Command	Icon	Shortcut	Description
Execute Code/ Resume Execution		[F5]	Used to start/continue the debugger. <ul style="list-style-type: none"> • Automatically starts a build if the project is out-of-date • Updates the status bar's message to indicate that the debugger is starting • Programs the selected target with the latest version of the project • Starts the debug session
Halt Execution		[Ctrl]+[Alt]+[Break]	Halts the debug target in the middle of whatever it is currently doing. Especially useful if the chip is not behaving as expected and you want to know what it is actually doing.
Stop Debugging		[Shift]+[F5]	Terminates the debug session and places PSoC Creator back in the standard perspective. Use this if you are finished debugging the code and ready to make changes or do something else.
Step Into		[F11]	Executes a single line of code. If the line is a function call, the debugger will break at the first instruction in the function. If the line is not a function call, the debugger will break at the following line of code. Use this to verify that a line of code is doing what is expected. This function temporarily allows the processor to run until it finishes processing the instructions that make up the current line of code.
Step Over		[F10]	Executes a single line of code. The debugger will break at the following line of code. If the current line of code is a function call, the function will be executed without stopping. The debugger will then stop on the next line after the function call. Use this to verify that a line of code is doing what is expected. This function temporarily allows the processor to run until it finishes processing the instructions that make up the current line of code.

Command	Icon	Shortcut	Description
Step Out		[Shift]+[F11]	Finishes executing the current function. The processor is allowed to run until the current function has finished. It will halt again at the first instruction after the function call. Use this to exit the current function and return to the calling method. This function temporarily allows the processor to run until it finishes processing the instructions that make up the current function.
Rebuild and Run		[Ctrl]+[Shift]+[F5]	Halts the current debug session, recompiles the project, programs the device with the updated code, and starts the debugger again. This allows for testing changes much faster than having to start and stop the debugger each time a change is made.
Reset		[Shift]+[Alt]+[F5]	Resets the Program Counter (PC) to zero, and puts the processor into a run state. This allows you to start over running the program without going through the whole build and program sequence.
Enable/Disable All Breakpoints			Alternately enables and disables all breakpoints in the workspace. Use this to quickly change the state of all breakpoints instead of having to change each one individually. This is useful if there are multiple breakpoints set but you just want the processor to run.
Enable/Disable Global Interrupt			Alternately enables and disables global interrupts. Disabling global interrupts allows the processor to step through Main code without the occurrence of Interrupts.

See Also:

- [Using the Debugger](#)
- [Debugger Menu Commands](#)
- [Text Editor Context Menu Commands](#)

Debugger Menu Commands

The Debug menus provide access to all of the functionality and information available to the current debugger. There are two modes for the menu: inactive and active.

Inactive Mode Debug Menu:

When the debugger is not running, the Debug menu will be in the inactive state. In this state, only the functions that make sense are available on the menu. For instance, the Halt function is unnecessary in the inactive state. In the inactive mode, the Debug Windows submenu provides access to a limited subset of the available debug windows.

Command	Icon	Shortcut	Description
Windows >			Provides access to the various debugger windows. See Debugger Windows .
Breakpoints		[Ctrl] + [D], [B]	Opens the Breakpoints window to display all of the breakpoints that have been set in the workspace.
Output		[Ctrl] + [D], [O]	Opens the Output window .
Program		[Ctrl] + [F5]	Provides a one click means of programming the selected debug target with the code generated from the selected project. <ul style="list-style-type: none"> Automatically starts a build if the project is out-of-date Updates the status bar's message to indicate that programming is taking place. Launches PSoC Programmer behind the scenes to perform the programming.
Select Debug Target			Opens the Select Debug Target dialog to manually select the debug target to use.
Debug		[F5]	Starts the debugger.
Debug without Programming		[Alt] + [F5]	Starts the debugger without programming the device.
Attach to Running Target			Opens the Attach to Target dialog to connect to an already programmed target device. Applicable to PSoC 3 and PSoC 5LP only.
Toggle Breakpoint		[F9]	Alternately inserts and removes a breakpoint in the current line of code. This is the same as clicking in the Indicator Margin of the Code Editor . You can use the [F9] key as a shortcut for this command.
New Breakpoint >			Provides access to the following breakpoint windows. See Breakpoints Window .
Address Breakpoint		[Ctrl] + [D], [A]	Opens the Address Breakpoint window.
File Line Breakpoint		[Ctrl] + [D], [F]	Opens the File/Line Breakpoint window.
Function Breakpoint		[Ctrl] + [D], [U]	Opens the Function Breakpoint window.
Variable Watchpoint		[Ctrl] + [D], [V]	Opens the Variable Watchpoints window.
Memory Watchpoint		[Ctrl] + [D], [E]	Opens the Memory Watchpoint window.
Delete All Breakpoints		[Ctrl] + [Shift] + [F9]	Deletes all breakpoints in the workspace instead of having to remove each one individually. This is useful if there are multiple breakpoints set but you just want the processor to run.
Enable All Breakpoints			Enables all breakpoints

Active Debug Windows Menu:

The active debug mode indicates that you have started a debug session; that the active PSoC Creator subsystem is the debugger. In the active mode, the debugger is running and you have the full range of available functions; the Debug Windows submenu provides access to all of the available debug windows.

Command	Icon	Shortcut	Description
Windows >			Provides access to the various debugger windows. See Debugger Windows .
Breakpoints		[Ctrl] + [D], [B]	Opens the Breakpoints window to display all of the breakpoints that have been set in the workspace.
Output		[Ctrl] + [D], [O]	Opens the Output window .
Watch >			
1		[Ctrl] + [D], [W]	Opens one of four Watch windows to evaluate and display variables, registers, or expressions.
2		[Ctrl] + [Alt] + [W], [2]	
3		[Ctrl] + [Alt] + [W], [3]	
4		[Ctrl] + [Alt] + [W], [4]	
Locals		[Ctrl] + [D], [L]	Opens the Locals window to view and modify all of the local variables in the current debug frame.
Components		[Ctrl] + [D], [P]	Opens the Component Debug Window to view debug information about Components in your design.
Call Stack		[Ctrl] + [D], [C]	Opens the Call Stack window to track the order that different functions are called by the target program.
Memory >			
1		[Ctrl] + [D], [M]	Opens one of four Memory windows to display the values stored in the memory of the processor.
2		[Ctrl] + [Alt] + [M], [2]	
3		[Ctrl] + [Alt] + [M], [3]	
4		[Ctrl] + [Alt] + [M], [4]	
Disassembly		[Ctrl] + [Alt] + [D]	Opens the Disassembly window to display the basic instructions created for your source code.
Registers		[Ctrl] + [R], [R]	Opens the Registers window to display the core CPU registers and their values
Program		[Ctrl] + [F5]	<p>Provides a one click means of programming the selected debug target with the code generated from the selected project.</p> <ul style="list-style-type: none"> • Automatically starts a build if the project is out-of-date • Updates the status bar's message to indicate that programming is taking place. • Launches PSoC Programmer behind the scenes to perform the programming.
Select Debug Target			Opens the Select Debug Target dialog to manually select the debug target to use.
Show Current Line			Displays the line of code that is or will be executed.

Command	Icon	Shortcut	Description
Resume Execution		[F5]	Continues the debugger. Starts the debug target running again after a Halt or a breakpoint. Use this function to have the program continue running to the next breakpoint.
Halt Execution		[Ctrl] + [Alt] + [Break]	Pauses the debugger.
Stop Debugging		[Shift] + [F5]	Stops the debugging session.
Rebuild and Run		[Ctrl] + [Shift] + [F5]	Rebuilds the program and restarts the debugger.
Reset		[Ctrl] + [Alt] + [F5]	Resets the debugger.
Step Into		[F11]	Executes a single line of code. If the line is a function call, the debugger will break at the first instruction in the function. If the line is not a function call, the debugger will break at the following line of code. Use this to verify that a line of code is doing what is expected. This function temporarily allows the processor to run until it finishes processing the instructions that make up the current line of code.
Step Over		[F10]	Executes a single line of code. The debugger will break at the following line of code. If the current line of code is a function call, the function will be executed without stopping. The debugger will then stop on the next line after the function call. Use this to verify that a line of code is doing what is expected. This function temporarily allows the processor to run until it finishes processing the instructions that make up the current line of code.
Step Out		[Shift] + [F11]	Finishes executing the current function. The processor is allowed to run until the current function has finished. It will halt again at the first instruction after the function call. Use this to exit the current function and return to the calling method. This function temporarily allows the processor to run until it finishes processing the instructions that make up the current function.
Toggle Breakpoint		[F9]	Alternately inserts and removes a breakpoint in the current line of code. This is the same as clicking in the Indicator Margin of the Code Editor . You can use the [F9] key as a shortcut for this command.
New Breakpoint >			Provides access to the following breakpoint windows. See Breakpoints Window .
Address Breakpoint		[Ctrl] + [D], [A]	Opens the Address Breakpoint window.
File Line Breakpoint		[Ctrl] + [D], [F]	Opens the File/Line Breakpoint window.
Function Breakpoint		[Ctrl] + [D], [U]	Opens the Function Breakpoint window.
Variable Watchpoint		[Ctrl] + [D], [V]	Opens the Variable Watchpoints window.
Memory Watchpoint		[Ctrl] + [D], [E]	Opens the Memory Watchpoint window.
Delete All Breakpoints		[Ctrl] + [Shift] + [F9]	Deletes all breakpoints in the workspace instead of having to remove each one individually. This is useful if there are multiple breakpoints set but you just want the processor to run.
Enable All Breakpoints			Enables all breakpoints.
Refresh			Refreshes the debugger.
Enable/Disable Global Interrupt			Enable/Disable (depending on current state) Global Interrupt

See Also:

- [Using the Debugger](#)
- [Debugger Toolbar Commands](#)
- [Text Editor Context Menu Commands](#)

Debugger Indicators

This topic explains the various indicator icons that appear in the debugger:

Breakpoints:

The following table shows the different breakpoint symbols you may encounter:

	Hardware
Permanent	
Temporary	

There are six main icons as follows:

- Breakpoint enabled
- Breakpoint disabled
- Breakpoint enabled with condition hit count
- Breakpoint disabled with condition hit count
- Breakpoint disabled due to error setting. View the error in the [Notice List Window](#).
- Breakpoint with condition hit count disabled due to error setting. View the error in the [Notice List Window](#).

There are four main categories: hardware, software, permanent, and temporary. Red means hardware; green means software. The "1" in the top right corner means temporary.

Watchpoints: 

A watchpoint will halt the program when the specified memory location is read, written, or accessed by the CPU. There are three watchpoint icons, as follows:

- Read Watchpoint
- Write Watchpoint
- Access Watchpoint

Current Line Indicator: 

This indicates the current line of code being executed. It is an indication of the command that will be executed when the program is resumed.

Active Stack Element:

This indicates the current stack element that is active, if not the topmost item in the call stack. It is an indication that the debugger is not focused on the top most element of the stack.

Stack Element:

This is grey highlighting surrounding a line of code indicating it is part of the call stack. This is an indication of how code is being executed.

See Also:

- [Using the Debugger](#)
- [Breakpoints Window](#)
- [Variable Watchpoints](#)
- [Call Stack Window](#)
- [Notice List Window](#)

Debugger Status Messages

The PSoC Creator Status Bar will display various messages at different times when using the debugger. The following sections describe the debugger messages you may see:

- **Starting Debugger** – Indicates that you requested to start debugging a project. PSoC Creator is in the process of configuring the selected device for debugging.
- **Checking Build** – Indicates what is actually happening in the process of starting up the debugger. It takes a while to check if a build is necessary and, if so, to start the build process. This task uses a great deal of CPU power.
- **Programming** – Indicates that the selected debug target is being programmed. The programming process takes about 3-5 seconds to complete. This message provides information that something is actually happening.
- **Debugging - Halted** – Indicates that the target device is halted and the user can interact with it. The debugger has two states, halted and running. The options available to the user are significantly different between the states. This message provides an at-a-glance indication of which state the debugger is in.
- **Debugging - Running** – Indicates that the target device is running and the user cannot interact with it.

See Also:

- [Using the Debugger](#)

Debugger Windows

The PSoC Creator debugger offers a set of windows that give useful information for the debug environment. The displays of these windows are all optional, although some of them will be displayed by default.

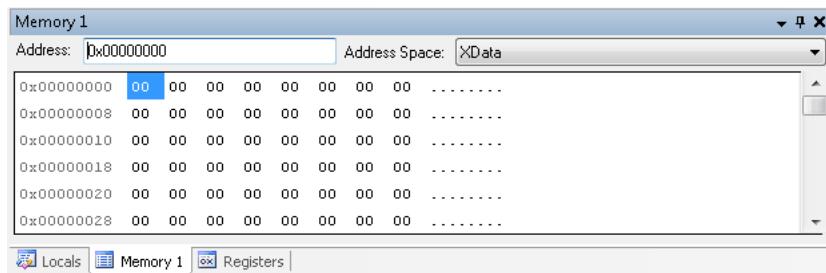
All of these windows are individually dockable or they can be free floating. If a window is free floating, it can be individually resized. If a window is docked, resizing it will affect the nearby docked windows. If more than one window is docked in the same location, a set of tabs will be placed at the bottom of the window listing all of the available windows. Persistence of the default window settings and the user interface to select the default windows is controlled by the PSoC Creator framework. See [Customizing the Framework](#) for more information about arranging and resizing windows.

The following are the debugger windows available:

- [Memory Window](#)
- [Watch Window](#)
- [Component Debug Window](#)
- [Breakpoints Window](#)
- [Registers Window](#)
- [Call Stack Window](#)
- [Locals Window](#)
- [Disassembly Window](#)

Memory Window

The Memory window displays the values stored in the memory of the processor.



The memory is broken up into the different regions defined by the target architecture. The window displays a list of addresses with the associated bytes along with an ASCII string that represents the series of displayed bytes. This window is updated every time a halt event occurs.

Several memory windows can exist at one time, with each window focusing on a different area in memory. This tool window is useful for viewing the raw data on the chip without any special filters or formatting. It is designed to allow quick viewing of large blocks of data for validation purposes. In addition, this may be the only means of viewing some regions of the chip that are not exposed via other windows.

To Display the Memory Window:

The debugger must be running or in break mode.

From the **Debug** menu, choose **Windows > Memory**, and click on **1, 2, 3, or 4**.

Context Menus:

The following options are available if you right-click in the window:

- **Columns** – Changes the number of data columns displayed. Default is 8.
- **Data View** – Changes the formatting of data. Default is 1-byte int.
- **Data Style** – Changes the display radix. Default is Hexadecimal.
- **ASCII Display** – Toggles whether the ASCII column is displayed
- **Copy** – Copies the selected item to the clipboard.
- **Add Watchpoint** – Adds a new watchpoint variable on the selected variable.

To Jump to a Specific Address:

Type the address in the Address field and press [**Enter**].

You can also jump to a specific function or variable address.

To Change Address Space:

Select the appropriate address space from the pull down menu.

To Edit Contents of Memory:

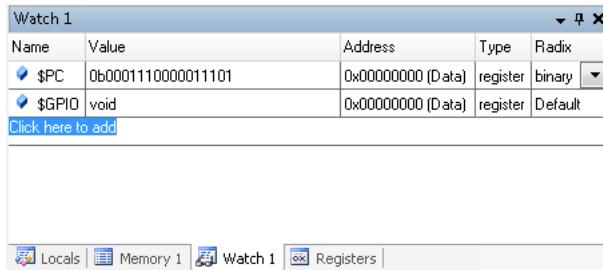
Click an entry to enable edit mode, type the desired value, and press [**Enter**].

See Also:

- [Using the Debugger](#)

Watch Window

The Watch window is used to evaluate and display variables, registers, or expressions. It will only display items that you specifically requested.



You can add items to the watch window by selecting a block of text in the [Text Editor](#), or by typing the expression directly into the **Name** column of the Watch window.

Use the Watch window to evaluate a wide range of expressions:

- **Simple variables** – Enter just the variable name. For example, if the program contains 'uint16 foo = 3;', enter 'foo'.
- **Array variables** – Enter just variable name with brackets around the index value. For example, if the program contains 'uint8[10] foo;', enter 'foo[2]'.
- **Struct variables** – Enter just the variable name with a period separating the different items. For example, if the program contains 'struct foo { uint16 a; uint8 b }; struct foo temp;', enter 'foo' to view the whole struct, or 'foo.a' to view just the 'a' member of the struct.
- **Registers** – Enter a '\$' sign in front of the register name. For example, to view the Program Counter, enter '\$PC'. All register names can be viewed by opening the [Registers window](#).
- **Expressions** – Enter the expression to evaluate. For example, to add a number to a variable, enter 'foo+3'.
- **Assignments** – Enter the assignment statement to evaluate. For example, to assign 3 to the variable foo, enter 'foo=3'.

The Watch window is automatically updated with the latest values at every halt event. While halted, you can modify the values of items to view and update any aspect of the design that might interest you. For numeric values, there is also an option to display the values in binary, octal, decimal, or hexadecimal. Several watch windows can exist at one time, with each window focusing on different aspects of the program. Each Watch window contains the following columns:

- **Name** – In this column you can type any valid expression recognized by the debugger.
- **Value** – The debugger evaluates the expression displayed in the **Name** column and places the result in this column.
 - If the expression is a variable or register name, you can edit the value in this column to change the contents of the variable or register.
 - You cannot edit the value of const variables.
 - You can edit and display register values for native-code applications only.
 - You can change the numeric format of the value to hexadecimal by right-clicking the Watch window and choosing the Hexadecimal Display option from the shortcut menu.
- **Address** – Shows the location of the variable in memory.
- **Type** – This column displays the data type of the variable or expression.
- **Radix** – Indicates how the **Value** is displayed.

To Display the Watch Window:

The debugger must be running or in break mode.

From the **Debug** menu, choose **Windows > Watch**, and click on **1, 2, 3, or 4**.

Context Menus:

The following options are available if you right-click in the window:

- **Copy** – Copies the selected item to the clipboard.

- **Paste** – Creates a new entry from the contents of the clipboard.
- **Edit Value** – Edits the entry's value, same as double clicking on the Value field.
- **Add Watchpoint** – Adds a new watchpoint variable on the selected variable.
- **Delete Watch** – Deletes the selected entry from the list.
- **Select All** – Selects all entries in the window.
- **Clear All** – Clears all entries in the window.
- **Radix** – Changes the default radix used by the debugger.
- **Expand Children** – Expands the entry's children items.
- **Collapse Parent** – Collapses the entry's parent item so child nodes are hidden.

To Enter a Watch Item:

1. Click in the **Name** column where it states "Click here to add."
2. Type the desired watch item and press [**Enter**].

To Modify a Watch Item:

Click an entry's **Value** field to change the value of the item, if not read-only.

To Change the Radix Display:

To change all rows, right-click and choose **Radix** and click the desired display type.

To change a single row, double-click the row to change, and select the display type from the pull-down menu.

To Delete a Watch Item:

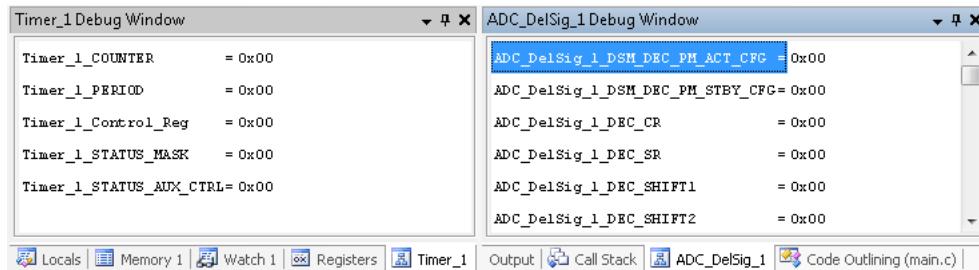
Right-click and select **Delete Watch**.

See Also:

- [Using the Debugger](#)
- [Text Editor](#)
- [Registers window](#)

Component Debug Window

The Component Debug Window is used to view debug information about Components in your design.



It provides a means of easily seeing what data is used to make up the Component. This can be helpful in tracking down issues with custom Components or figuring out why a Component is not behaving as expected.

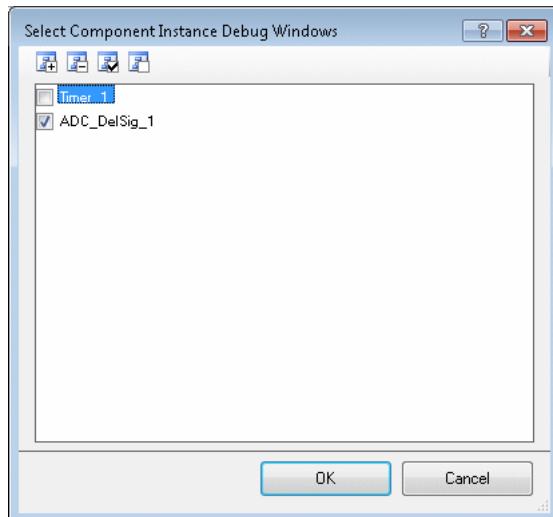
This window is generated for each Component instance in a project for which you selected using the [Select Component Instance Debug Windows](#). Each Component window lists the memory and registers for the instance, as well as any sub-Components that are used by it. Since not all Components have all three of these items, only the resources actually used will be displayed. This means some windows may only have a memory or register view. The memory and registers windows behave exactly like the main [Memory window](#) and [Registers window](#). The list of sub-Components are links to provide easy viewing access.

To Open this Window:

The debugger must be running or in break mode.

- From the **Debug** menu, choose **Windows > Components...**

The Component Window Selector dialog displays.



- Select the Component instances to view in the Components window and click **OK**.

The selected Component Debug Window(s) will open within the debugger framework. You can re-arrange how these windows are displayed within the framework to suit your needs.

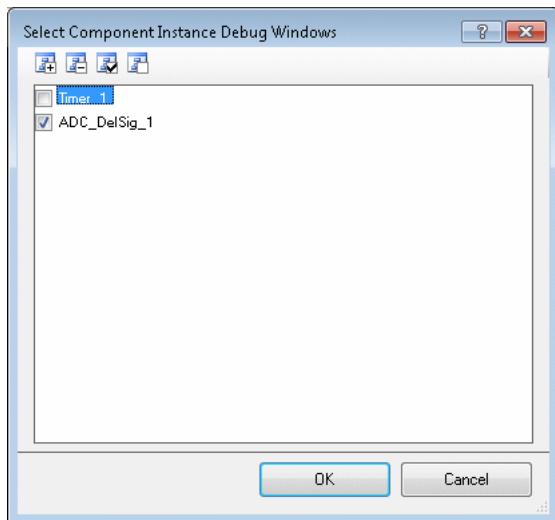
See Also:

- [Select Component Instance Debug Windows](#)

- [Memory Window](#)
- [Registers Window](#)

Select Component Instance Debug Windows

The Select Component Instance Debug Windows dialog allows you to select from a list of all Component instances that contain debug information.



Each instance you select will display in a different [Components Window](#) during a debug session. It presents each of the available Components in a simple tree view, which preserves the Components' nesting hierarchy. It also provides a clear indication of which Components are made up of other Components.

To Select/Deselect a Component Window:

Check/uncheck a check box next to each entry to cause the window to be displayed (checked) or closed (unchecked).

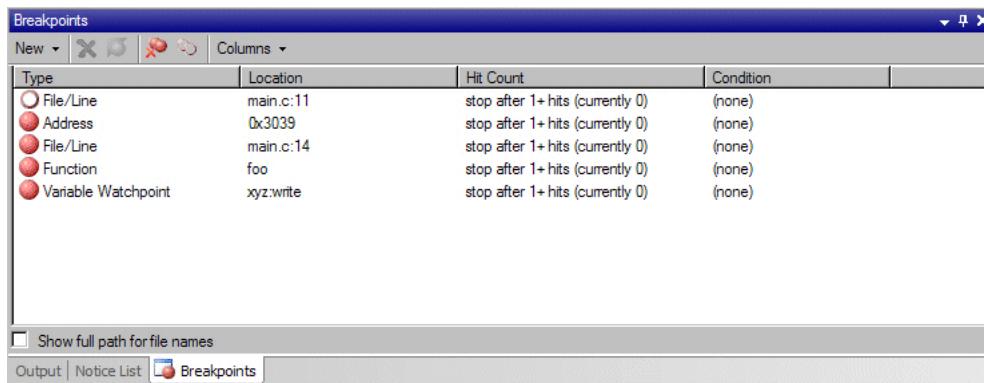
You can also use **Select All** or **Deselect All**, as desired.

See Also:

- [Components Window](#)

Breakpoints Window

The Breakpoints window displays all of the breakpoints that have been set in the workspace, whether they are enabled or not. The window displays the list of breakpoints by type and location. If the processor (debug tool chain) supports it, this window also provides additional information about the breakpoint such as the number of times it has been hit and any condition that must be true for it to be triggered.



A breakpoint is used to cause the debugger to halt the next time its location is reached. Breakpoints can be set, listed, disabled, or removed. Breakpoints can be set up in a number of different ways, such as specifying the line number of a file on which a breakpoint should be set, a function in which a breakpoint must exist, or an address when accessed.

If you set a hardware breakpoint, a red dot will appear in the **Type** column, as well as in the left margin of the source/disassembly corresponding to the line in which the breakpoint is assigned. A disabled breakpoint has its associated red dot replaced with a red circle. When a breakpoint is removed its corresponding dot/circle is removed from the margin. For a list and description of all debugger indicators, see [Debugger Indicators](#).

The number of breakpoints available is limited by the PSoC device. Refer to the appropriate device datasheet for the number of breakpoints available. PSoC Creator reserves a single breakpoint to perform other operations such as step, jump, and run-to-cursor. If the number of hardware breakpoints has been exhausted, and if you attempt to add another, a message will display in the [Output Window](#).

Note Breakpoints are hit twice when interrupts are enabled. This happens because the breakpoint gets hit, but before the line of code is actually executed an interrupt takes over and gets processed. When the interrupt has completed, the processor returns to the original line of code. This causes the breakpoint to be hit again.

To Open the Breakpoints Window:

Click **Debug > Windows > Breakpoints**.

Commands:

The Breakpoints window contains the following toolbar commands:

Icon	Command	Description
	New	Pull down menu to create different types of breakpoints.
	Delete	Deletes the selected breakpoint.
	Enable/Disable Breakpoint	Alternately enables and disables the selected breakpoint.
	Delete All Breakpoints	Deletes all breakpoints in the workspace instead of having to remove each one individually. This is useful if there are multiple breakpoints set but you just want the processor to run.

Icon	Command	Description
	Enable/Disable All Breakpoints	Alternately enables and disables all breakpoints in the workspace. Use this to quickly change the state of all breakpoints instead of having to change each one individually. This is useful if there are multiple breakpoints set but you just want the processor to run.
	Columns	Pull down menu to show/hide the columns in the window.
	Show full path for file names	Alternately shows/hides the full path name for file/line breakpoints.

To Set a Breakpoint:

1. Open the file you want to debug.
2. Click at targeted points in the left margin of the open file to set breakpoints; click again to unset them.

To Modify an Existing Breakpoint:

You can modify the breakpoint by right-clicking on it to access the following commands:

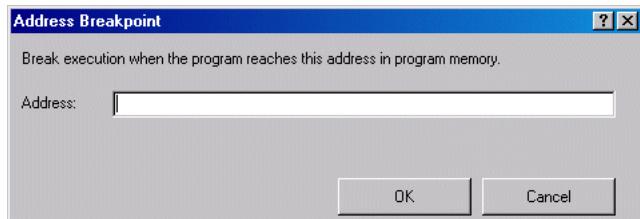
- **Delete** – Deletes the selected breakpoint.
- **Enable/Disable** – Alternately enables and disables the selected breakpoint.
- **Temporary** – Sets the breakpoint as temporary.
- **Location** – Opens the selected type of breakpoint dialog to modify the location.
- **Condition** – Opens the [Breakpoint Condition](#) dialog.
- **Hit Count** – Opens the [Breakpoint Hit Count](#) dialog.

See Also:

- [Address Breakpoint](#)
- [File/Line Breakpoint](#)
- [Function Breakpoint](#)
- [Variable Watchpoints](#)
- [Memory Watchpoint](#)
- [Breakpoint Condition](#)
- [Breakpoint Hit Count](#)
- [Output Window](#)

Address Breakpoint

The Address Breakpoint dialog is used to create a new breakpoint (or modify an existing one) at a specific address.



This breakpoint will be hit each time the Program Counter (PC) matches the address. It does not require any information about the source code that was compiled to generate the program data.

When not in debug mode, you will not see address breakpoints displayed in the margin indicator of the source editor. These types of breakpoints are only shown in the margin while in debug mode. You can see these breakpoints in the [Breakpoints window](#).

To Open the Dialog:

New Breakpoint

For a new breakpoint:

- From the PSoC Creator Debug menu, select New Breakpoint > Address Breakpoint...
- In the Breakpoints window **New** menu, select **Address...**

Existing Breakpoint

For an existing breakpoint, in the Breakpoints window, right-click on the breakpoint and select **Location...**

To Create a New Breakpoint:

Enter either a decimal or hexadecimal address to break on and click **OK**.

To Modify an Existing Breakpoint:

You can modify the breakpoint by right-clicking on it in the Breakpoints window to access the following menus:

- **Temporary** - Sets the breakpoint as temporary.
- **Location** - Opens the Address Breakpoint dialog to modify the address.
- **Condition** - Opens the [Breakpoint Condition](#) dialog.
- **Hit Count** - Opens the [Breakpoint Hit Count](#) dialog.

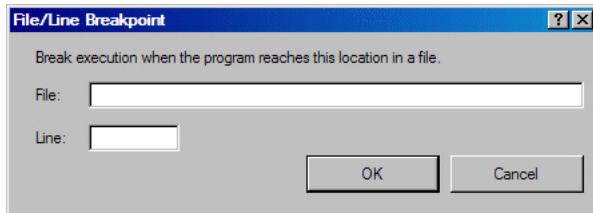
See Also:

- [Breakpoints Window](#)
- [File/Line Breakpoint](#)
- [Function Breakpoint](#)
- [Variable Watchpoints](#)
- [Breakpoint Condition](#)

- [Breakpoint Hit Count](#)

File/Line Breakpoint

The File/Line Breakpoint dialog is used to create a new breakpoint (or modify an existing one) at a specific file line of source code.



This breakpoint will be hit each time the line of code is being executed. This is the most convenient, but least precise, means of adding a breakpoint.

To Open the Dialog:

New Breakpoint

For a new breakpoint, from the PSoC Creator **Debug** menu, select **New Breakpoint > File/Line Breakpoint...**

Existing Breakpoint

For an existing breakpoint, in the Breakpoints window, right-click on the breakpoint and select **Location...**

To Create a New Breakpoint:

Click in the **Indicator Margin** of the [Code Editor](#) at the line in which you want to create a breakpoint.

Note You can also use the dialog to enter the line number and full name of the file in which to put the breakpoint and click **OK**.

To Modify an Existing Breakpoint:

You can modify the breakpoint by right-clicking on it in the Breakpoints window to access the following menus:

- **Temporary** - Sets the breakpoint as temporary.
- **Location** - Opens the Address Breakpoint dialog to modify the address.
- **Condition** - Opens the [Breakpoint Condition](#) dialog.
- **Hit Count** - Opens the [Breakpoint Hit Count](#) dialog.

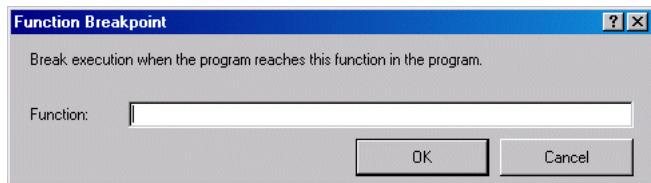
See Also:

- [Breakpoints Window](#)
- [Address Breakpoint](#)
- [Function Breakpoint](#)
- [Variable Watchpoints](#)
- [Breakpoint Condition](#)

- [Breakpoint Hit Count](#)

Function Breakpoint

The Function Breakpoint dialog is used to create a new breakpoint (or modify an existing one) at a specific function in the program.



Use this to create a breakpoint that will be hit when the function is called, independent of how the code inside and around the function is changed.

When not in debug mode, you will not see function breakpoints displayed in the margin indicator of the source editor. These types of breakpoints are only shown in the margin while in debug mode. You can see these breakpoints in the [Breakpoints window](#).

Note A function breakpoint does not get displayed in the margin indicator. When the function has arguments, Keil expects '_' in front of the function name. Otherwise the function breakpoint does not work. To address this, prefix the function name with '_'.

To Open the Dialog:

New Breakpoint

For a new breakpoint:

- From the PSoC Creator Debug menu, select New Breakpoint > Function Breakpoint...
- In the Breakpoints window **New** menu, select **Function...**

Existing Breakpoint

For an existing breakpoint, in the Breakpoints window, right-click on the breakpoint and select **Location...**

To Create a New Breakpoint:

Enter the name of a function to break on and click **OK**.

To Modify an Existing Breakpoint:

You can modify the breakpoint by right-clicking on it in the Breakpoints window to access the following menus:

- **Temporary** - Sets the breakpoint as temporary.
- **Location** - Opens the Function Breakpoint dialog to modify the function.
- **Condition** - Opens the [Breakpoint Condition](#) dialog.
- **Hit Count** - Opens the [Breakpoint Hit Count](#) dialog.

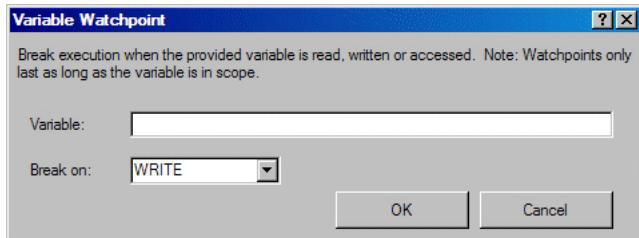
See Also:

- [Breakpoints Window](#)

- [Address Breakpoint](#)
- [File/Line Breakpoint](#)
- [Variable Watchpoints](#)
- [Breakpoint Condition](#)
- [Breakpoint Hit Count](#)

Variable Watchpoints

The Variable Watchpoint dialog is used to create a watchpoint (or modify an existing one) on a specific variable.



A watchpoint is used to set a breakpoint on a place in data memory as opposed to the standard breakpoint, which is used for program memory. Instead of setting the breakpoint on a line of code, you set it on a variable in the code.

This watchpoint will be hit each time the address at which the variable is located is read, written, or accessed based on your selection for **Break on**. Use it to track when specific variables/addresses are read from or written to. This can help track down why a specific memory location does not have the expected value. Just like a standard breakpoint, when hit, a watchpoint will cause the program to stop executing and an indicator will be displayed showing what instruction triggered the watchpoint.

Unlike standard breakpoints that are associated with a specific instruction in code, watchpoints can be hit at any number of instructions. It all depends on the data that is accessed by the instruction. Because of this, watchpoints do not show any indicator in the left hand margin of the text editor. However, they are still listed in the [Breakpoint Window](#).

Note The number of watchpoints available depends on the device being debugged. Refer to the applicable device datasheet for more information.

To Open the Dialog:

New Watchpoint

For a new watchpoint:

- From the PSoC Creator Debug menu, select New Breakpoint > Variable Watchpoint...
- In the Breakpoints window **New** menu, select **Watch Variable...**

Existing Watchpoint

For an existing watchpoint, in the Breakpoints window, right-click on the watchpoint and select **Location...**

To Create a New Watchpoint:

Enter the name of the variable to break on and click **OK**.

To Modify an Existing Breakpoint:

You can modify the breakpoint by right-clicking on it in the Breakpoints window to access the following menus:

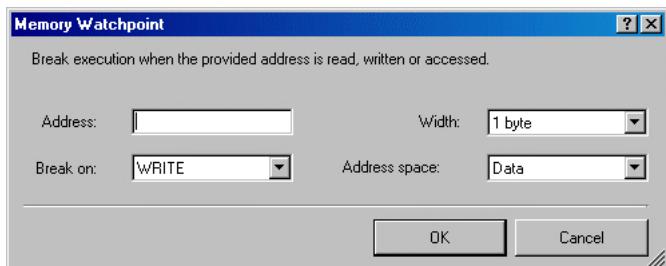
- **Location** - Opens the Variable Watchpoint dialog to modify the address.
- **Condition** - Opens the [Breakpoint Condition](#) dialog.
- **Hit Count** - Opens the [Breakpoint Hit Count](#) dialog.

See Also:

- [Breakpoints Window](#)
- [Address Breakpoint](#)
- [File/Line Breakpoint](#)
- [Function Breakpoint](#)
- [Memory Watchpoint](#)
- [Breakpoint Condition](#)
- [Breakpoint Hit Count](#)

Memory Watchpoint

The Memory Watchpoint dialog allows you to select the address, width, address space, and break type.



Memory watchpoints are similar to [Variable Watchpoints](#), but as they can be set on a particular memory address instead of a variable. Memory watchpoints are used by the [Analog Device Editor](#) to detect when switches are opened or closed. However, memory watchpoints can be used for other things as well, such as monitoring for stack or data corruption.

The dialog contains the following fields:

- **Address** – The address to set the breakpoint on. (**Note** The bottom n-bits of the address will be ignored based on the specified Width.)
- **Width** – The number of bytes to set the breakpoint on. This is implemented by ignoring the bottom n-bits in the address.
- **Break On** – The type of memory access for which the watchpoint should be triggered.
- **Address Space** – If the device has multiple memory spaces, this selects which one.

To Open the Dialog:

New Watchpoint

For a new watchpoint:

- From the PSoC Creator Debug menu, select New Breakpoint > Memory Watchpoint...
- In the [Breakpoints window](#) New menu, select **Watch Memory...**

Existing Watchpoint

For an existing watchpoint, in the Breakpoints window, right-click on the watchpoint and select **Location...**

To Create a New Watchpoint:

Enter a decimal or hexadecimal address to break on and click **OK**.

To Modify an Existing Breakpoint:

You can modify the breakpoint by right-clicking on it in the Breakpoints window to access the following menus:

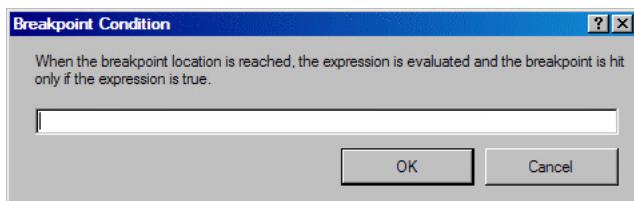
- **Location** - Opens the Memory Watchpoint dialog to modify the address.
- **Condition** - Opens the [Breakpoint Condition](#) dialog.
- **Hit Count** - Opens the [Breakpoint Hit Count](#) dialog.

See Also:

- [Breakpoints Window](#)
- [Address Breakpoint](#)
- [File/Line Breakpoint](#)
- [Function Breakpoint](#)
- [Variable Watchpoints](#)
- [Breakpoint Condition](#)
- [Breakpoint Hit Count](#)

Breakpoint Condition

The Breakpoint Condition dialog is used to add/modify a condition on a breakpoint.



This allows you to control when the debugger will actually report that the breakpoint was hit. Use it to allow code to be executed an arbitrary number of times while the condition is not true. This provides a quicker means of identifying problems than breaking every time and then manually evaluating the condition.

To Open the Dialog:

In the Breakpoints window, right-click on the breakpoint and select **Condition...**

To Set a Breakpoint Condition:

Enter a condition that must evaluate to true before the code is halted, and click **OK**.

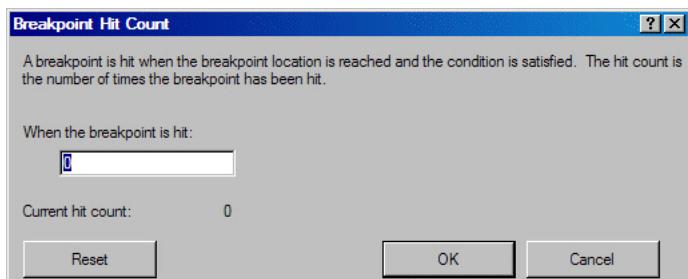
For example: `i>10 or a==b`

See Also:

- [Breakpoints Window](#)
- [Address Breakpoint](#)
- [File/Line Breakpoint](#)
- [Function Breakpoint](#)
- [Variable Watchpoints](#)
- [Breakpoint Hit Count](#)

Breakpoint Hit Count

The Breakpoint Hit Count dialog is used for setting the number of times the breakpoint must have been encountered before actually stopping.



A breakpoint is hit when the breakpoint location is reached and the condition is satisfied. The hit count is the number of times the breakpoint has been hit. This allows you to control when the debugger will actually report that the breakpoint was hit. Use it to allow the code to be executed a few times before actually triggering the breakpoint for user intervention. This is useful for quickly checking boundary conditions and finding out if code is executing more than expected.

To Open the Dialog:

In the Breakpoints window, right-click on the breakpoint and select **Hit Count...**

To Set the Breakpoint Hit Count:

Enter the number of times the breakpoint can be hit before halting click **OK**.

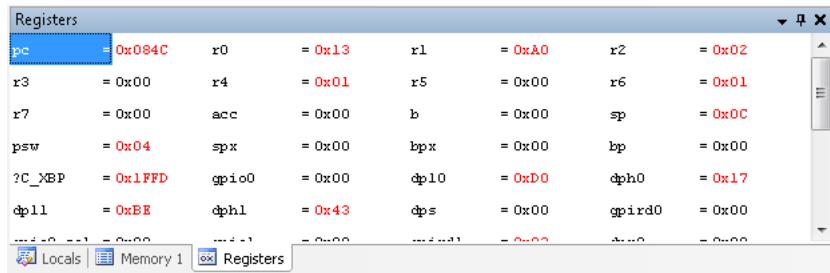
Click **Reset** to reset the number of times the breakpoint has already been hit.

See Also:

- [Breakpoints Window](#)
- [Address Breakpoint](#)
- [File/Line Breakpoint](#)
- [Function Breakpoint](#)
- [Variable Watchpoints](#)
- [Breakpoint Condition](#)

Registers Window

The Registers window displays the core CPU registers and their values.



It provides a means of quickly viewing what is happening in core of the processor. It also allows for modifying the value of any of the registers as you see fit to be sure the processor is doing what it is expected to do.

This window will be updated at every halt event. The values displayed in this window are displayed in hexadecimal format by default.

To Display the Registers Window:

The debugger must be running or in break mode.

From the **Debug** menu, choose **Windows > Registers**.

Context Menus:

The following options are available if you right-click in the window:

- **Copy** – Copies the selected item to the clipboard.
- **Select All** – Selects all registers in the display.
- **Details...** – Opens the [Register Details](#) window.
- **Radix** – Changes the default radix used by the debugger.

To Edit a Register Value:

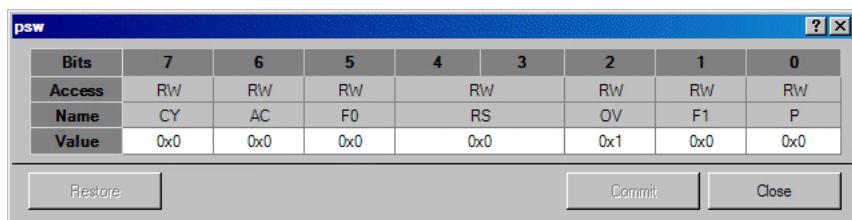
Click an entry to enable edit mode, type the desired value, and press [**Enter**].

See Also:

- [Using the Debugger](#)

Register Details

The Register Details dialog allows you to see more detailed information about a specific register. It displays each of the fields within the register, their access restrictions, and the value that is set for the field.



Additionally, hovering over a field brings up a tool tip describing the field and, in some cases, what specific settings mean.

To Open the Dialog:

Double click on a register in the [Registers Window](#), or right click on the register and select **Details...**

To Modify a Value:

Click on the value cell for the field you wish to modify, set the new value and clicking the **Commit** button.

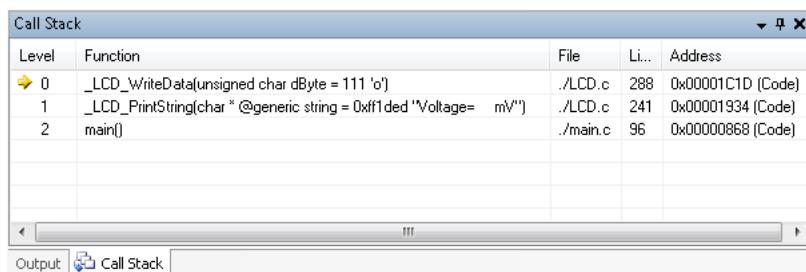
Click the **Restore** button to restore the value of the register that is currently on the chip.

See Also:

- [Registers Window](#)
- [Debugger Windows](#)

Call Stack Window

The Call Stack window is used to track the order that different functions are called by the target program.



Each function call that has not yet completed gets placed on the top of the stack. When the function is completed it will get popped off. This tool is useful in seeing how code is being executed and to make sure things are happening in the correct order. It provides an easy means of tracing exactly how the program got to executing the current line of code.

This window displays the name of each function and may be accompanied by optional information, such as line number, byte offset, etc. The display of this optional information can be turned on or off.

To Display the Call Stack Window:

The debugger must be running or in break mode.

From the **Debug** menu, choose **Windows** and click **Call Stack**.

Context Menus:

The following options are available if you right-click in the window:

- **Copy** – Copies the selected item to the clipboard.
- **Select All** – Selects all registers in the display.
- **Switch to Frame** – Changes the active item, same as double clicking an item
- **Run to Frame** – Causes code to execute until the selected frame is reached
- **Show Module Name** – Shows the file that the function exists in
- **Show Parameter Type** – Shows the type of each function parameter
- **Show Parameter Name** – Shows the name of each function parameter
- **Show Parameter Value** – Shows the value of each function parameter
- **Show Line Number** – Shows the line in the function that performed the call
- **Show Byte Offset** – Shows the instruction address that the call was made from

To Change the Optional Information Displayed:

Right-click the Call Stack window and then click the desired information to show or hide from the shortcut menu.

To Change the Active Call Stack Item/Frame:

Double-click and entry or right-click and select **Switch to Frame**.

This will shift the focus on the line of code in the editor that corresponds to the clicked frame. It will also cause the debugger to change the active frame. This causes the [Locals window](#) to update with the local variables from the selected frame.

See Also:

- [Using the Debugger](#)
- [Locals Window](#)

Locals Window

The Locals window allows you to view and modify all of the local variables in the current debug frame.

Locals				
Name	Value	Address	Type	Radix
i	0x00 '\0'	0x00000104 (XData)	char	Default
voltageMv	0x0000	0x00000105 (XData)	int	Default
sampleAverage	0x00000000	0x00000107 (XData)	long	Default

Locals | Memory 1 | Registers

You can see the value, type, and address of any variable. This allows you to quickly see if a function is behaving as expected and updating its variable items appropriately.

This window contains the following columns:

- **Name** contains the names of all local variables in the current scope. Structure and array variables have a tree control that you can use to display or hide the elements.
- **Value** shows the value contained by each variable. By default, integer variables are represented in hexadecimal form. You can change the representation to decimal by right-clicking the Locals window and choosing the **Decimal Display** option from the shortcut menu.
- **Address** shows the location of the variable in memory.
- **Type** identifies the data type of each variable listed in the Name column.
- **Radix** indicates how the Value is displayed.

To Display the Locals Window:

The debugger must be running or in break mode.

From the **Debug** menu, choose **Windows** and click **Locals**.

The default context is the function containing the current execution location.

Context Menus:

The following options are available if you right-click in the window:

- **Copy** – Copies the selected item to the clipboard.
- **Edit Value** – Edits the entry's value, same as double clicking on the Value field.
- **Add Watchpoint** – Adds a new variable watchpoint on the selected variable.
- **Select All** – Selects all entries in the window.
- **Radix** – Changes the default radix used by the debugger.
- **Expand Children** – Expands the entry's children items.
- **Collapse Parent** – Collapses the entry's parent item so child nodes are hidden.

To Modify a Variable:

Double-click an entry's **Value** field to change the value of the item, if not read-only.

To Change the Radix Display:

To change all rows, right-click and choose **Radix** and click the desired display type.

To change a single row, double-click the row to change, and select the display type from the pull-down menu.

See Also:

- [Using the Debugger](#)
- [Locals Window](#)

Disassembly Window

The Disassembly window displays the basic instructions created for your source code.



```

TopDesign.cysch main.c Disassembly
283: void LCD_WriteData(uint8 dByte)
0x000001C18 mov dptr, #laf
0x000001C1B mov a, r7
0x000001C1C movx @dptr, a
284: {
285:     uint8 nibble;
287:     LCD_IsReady();
0x000001C1D lcall 000fb6 <LCD_IsReady>
288:     nibble = dByte >> LCD_NIBBLE_SHIFT;
0x000001C20 mov dptr, #laf
0x000001C23 movx a, @dptr
0x000001C24 mov r7, a
0x000001C25 mov a, r7
0x000001C26 swap a
0x000001C27 anl a, #f
0x000001C29 mov r7, a

```

Rather than forcing you to read instruction codes in binary or hexadecimal format, the instructions are disassembled into assembly-language format. It allows you to view and enter breakpoints from combination of user source code and compiled assembly instructions in order to see at a lower level exactly what the code is doing.

Using the Disassembly window, you can step through the assembly code, set breakpoints and interact with the code using all the same capabilities you can use when debugging the C source code. Mixed mode disassembly, for example interspersed 'C' code, may also be displayed.

Assembly-language code consists of mnemonics, which are abbreviations for instruction names, and symbols that represent variables, registers, and constants. Each machine-language instruction is represented by one assembly-language mnemonic, usually followed by one or more variables, registers, or constants. Because assembly code relies heavily on processor registers (or, in the case of managed code, common language runtime registers), you will often find it useful to use the Disassembly window in conjunction with the [Registers window](#), which allows you to examine register contents.

To Display the Disassembly Window:

The debugger must be running or in break mode.

From the **Debug** menu, choose **Windows > Disassembly**.

Context Menus:

The following options are available if you right-click in the window:

- **Insert Breakpoint** – Inserts an address breakpoint at the current address.
- **Break Here Once** – Inserts a temporary address breakpoint at the address.
- **Run to Instruction** – Run the processor until the current address is reached.
- **Go to PC** – Sets the focus of the window to the current PC address.
- **Go to Address** – Sets the focus of the window on the provided address.
- **Show Source Code** – Intermixes user source and assembly code.
- **Show Line Numbers** – Shows/hides line numbers.
- **Hide Empty Lines** – Hides blank lines.
- **Show Code Bytes** – Shows the code data that represents the instructions.
- **Copy** – Copies the selected item to the clipboard.

To Insert a Breakpoint:

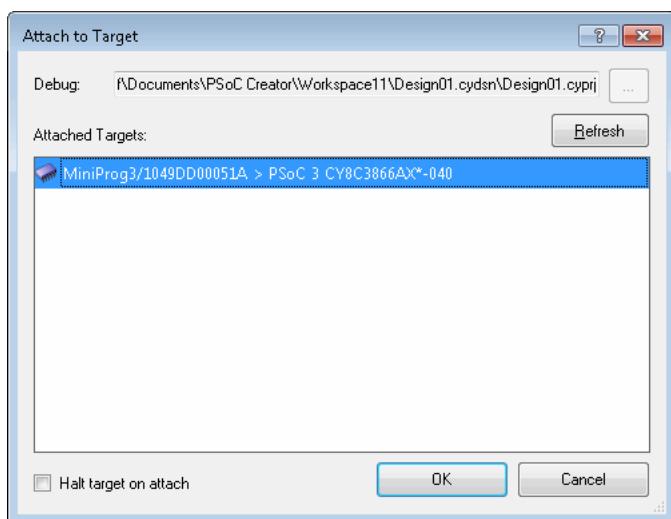
Click in the margin next to a line of code. Click again to remove it.

See Also:

- [Using the Debugger](#)
- [Registers Window](#)

Attach to Target

The Attach to Target dialog allows you to connect to an already programmed target device.



Use this dialog to debug a design that has been running for a while and has stopped working as expected. You can also use it for debugging a design for which no source code is available. Unlike the standard debugging flow, this command will not reprogram the device before opening the debugger.

Note Use of this feature will cause a momentary interruption in the code running on the PSoC device.

You can select a project file, a workspace file, or an elf file to use as the debug file. If you select a Project or Workspace, it will open that item in PSoC Creator. If a project is already open in PSoC Creator, this field will be automatically populated with the active project. This field is not required to attach the debugger to the target, but specifying a file does allow for more options while debugging.

Note For PSoC 6 devices, if you attach to a running Cm4 target, it will wake up the Cm0+ core.

To Open the Dialog:

Select **Attach to Running Target...** from the **Debug** menu.

To Select a File:

Click [...] and navigate to the Project, the Workspace, or the ELF file to select.

If a project is already open in PSoC Creator, this field will be automatically populated with the active project, and the browse button will be disabled. You can only select a file if there is no active project.

This field is not required to attach the debugger to the target, but specifying it does allow for more options while debugging.

To Select a Device:

Click on one of the devices listed that are attached to the computer. You may choose any item from this list to attach to.

To Halt Target on Attach:

Check this box to cause the debugger to halt the target device and show you where it halted.

See Also:

- [Debugger Menu Commands](#)
- [Using the Debugger](#)

Error Handling

This section details how the debugging module will handle various errors, such as the USB cable being unplugged.

Multiple Debugger Instances:

You can run multiple instances of PSoC Creator on a single PC. A USB port and/or a JTAG DUT lock mechanism and lock detection can be used to inform you when a PSoC Creator debugger instance has exclusive use of a USB device and or a JTAG DUT.

Hardware Target is Halted or Reset Externally:

It is possible for the hardware to be reset or halted independently of PSoC Creator. When this occurs, PSoC Creator will inform you via a break point indicator in the source or disassembly and a status bar message reporting the hardware has been reset/halted.

If the hardware is halted externally, the debugger will inform you as though you pressed **Break**. If reset, it will behave as though it were disconnected.

Hardware is Disconnected:

If the hardware is disconnected from the PC during a debugging session, the IDE will inform you through a dialog box the next time you attempt to communicate with the chip.

8 Completing the Project



There are several steps to complete your PSoC Creator project, including:

- [Review Device Datasheet](#)
- [Optimize Compiler Settings](#)
- [Archive the Project](#)
- [Set Build Configuration](#)
- [Select Programming Protocol](#)
- [Enable Device Protection](#)
- [Select Optional Reset Line](#)
- [Select Flash Security Protection](#)
- [Enable Write Once Latch Flash Protection](#)
- [Evaluate General Programming Options](#)

Review Device Datasheet

It is critical when going to production to finalize application pin selections, PCB layout, and hardware design against the recommended layout detailed in the respective PSoC device datasheets. You can navigate to the correct PSoC device datasheet available on www.cypress.com using the **Documentation** tab on the [Workspace Explorer](#).

The device datasheet provides a summary of the features, electrical characteristics, pin-outs, device-level specifications and peripheral electrical specifications. Common errors include layout error on the programming interface, reset lines, power connections and associated voltage connections. Refer also to [AN61290: PSoC 3 and PSoC 5LP Hardware Design Considerations](#).

You could also experience Firmware and Hardware misalignments when you develop designs using development kits that support the full-featured devices, and then move the design to a smaller PSoC package. The application may target pins or features that are not available on the smaller PSoC device. Please verify the pin-out and feature sets if the application is designed on a different PSoC package.

See Also:

- [Completing the Project](#)

- [Workspace Explorer](#)
- [AN61290](#)

Optimize Compiler Settings

As you migrate from development to production, project designs may require advanced code compression to fit large applications into the PSoC flash space. You will need to review the compiler code compression documentation provided through the compiler installations. Depending on your compiler selection, navigate to one of the following directories:

- <INSTALL_DIR>\PSoC Creator\import\gnu_cs\arm\<version>\share\doc\
- <INSTALL_DIR>\PSoC Creator\import\keil\pk51\<version>\C51\hlp

Note These documents are also available directly from the PSoC Creator **Help** menu.

See Also:

- [Completing the Project](#)
- [Help Menu](#)

Download and Archive Development Tools

As you move to production with a design, Cypress recommends that you store a backup copy of the PSoC Creator and PSoC Programmer development environments. This will be useful if you must return to a design environment in the future to make modifications or updates to the production project. This can be accomplished by navigating to the PSoC Creator and PSoC Programmer web pages and downloading the ISO images available in the downloads tables. These ISO images can be used to burn CDs/DVDs with the archived software contents.

- <http://www.cypress.com/go/psoccreator>
- <http://www.cypress.com/go/psocprogrammer>

See Also:

- [Completing the Project](#)

Archive the Project

When the development project has moved to a production state, you may be able to archive your project using the [PSoC Creator Workspace/Project Archiver](#). The Archiver tool supports a number of options on file density, scope of the archive, and compress options.

This is a valuable feature to store a development environment or place a project under [source control](#).

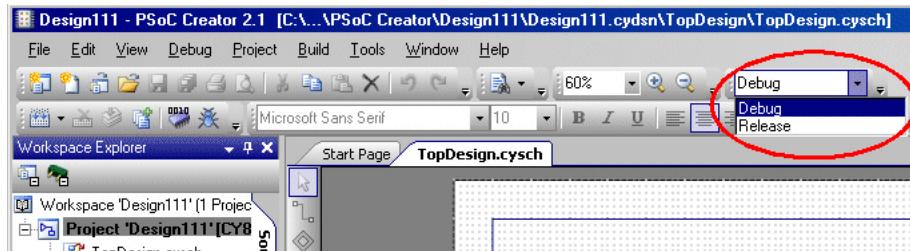
See Also:

- [Completing the Project](#)
- [Archiving a Workspace/Project](#)
- [Source Code Control](#)

Set Build Configuration

PSoC Creator provides **Debug** and **Release** build configurations for the compiler tool chains. Changing between build configurations can help when developing and testing a design. For example, while first developing code, it is easiest to use the **Debug** configuration, which typically has fewer optimizations and produces more debug information. As you move to a production state, the code becomes stable and is getting ready for release. Thus, using the **Release** configuration becomes preferable as additional optimizations are often desired. These optimizations help cut down the size of the program and allow it to run faster.

The **Build Configuration** toolbar is available by default:



If the toolbar is not available, refer to the [Customize](#) dialog topic for how to add it.

See Also:

- [Completing the Project](#)
- [Customize](#)

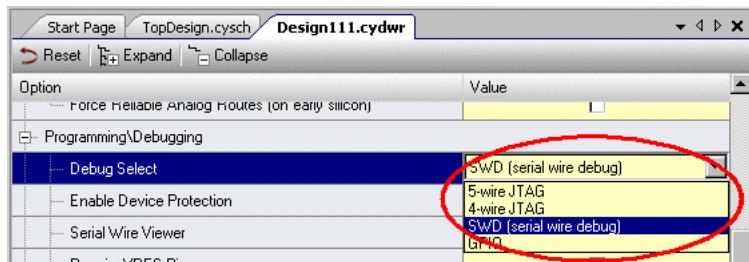
Select Programming Protocol

PSoC Creator supports configuration of the programming port, in the [System Editor](#). This is a critical selection when going into production. Make sure that this selection matches your system configuration and expectation. This applies to not only the initial programming event but also to subsequent programming events, if necessary.

To select this feature:

Open the design-wide resources (<project>.cydwr) file from the [Workspace Explorer](#). Then, select the **System** tab.

Under the **Programming\Debugging** options, use the **Debug Select** pull-down menu to select the correct programming protocol.



See Also:

- [Completing the Project](#)
- [System Editor](#)
- [Workspace Explorer](#)

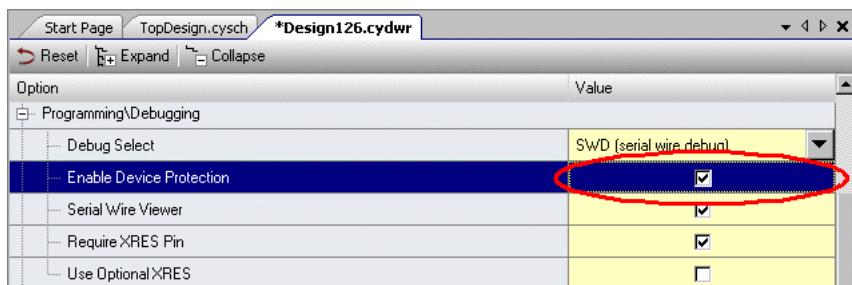
Enable Device Protection

When completing a production design, you should select the **Enable Device Protection** feature. Enabling this feature causes the part to implement flash protection settings and to disable debugging at run-time. It is still possible to connect a programmer, but debugging will be disabled.

To select this feature:

Open the design-wide resources (<project>.cydwr) file from the [Workspace Explorer](#). Then, select the **System** tab.

Under the **Programming\Debugging** options, select the **Enable Device Protection** check box.



See Also:

- [Completing the Project](#)
- [System Editor](#)
- [Workspace Explorer](#)

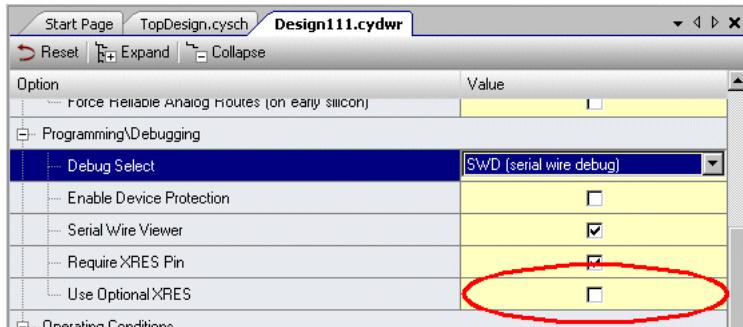
Select Optional Reset Line

PSoC Creator supports optional reset (XRES) line selections, in the [System Editor](#). It is important to ensure that this selection has been either enabled or disabled depending on system designs.

To select this feature:

Open the design-wide resources (<project>.cydwr) file from the [Workspace Explorer](#). Then, select the **System** tab.

Under the **Programming\Debugging** options, select or deselect the **Use Optional XRES** pull-down check box as needed..



Note If you have selected the optional reset line and require in-system programming for your target system, you will need to ensure that your programming solution (distributor, contract manufacturer, or third-party programming vendor) supports power cycle programming modes.

See Also:

- [Completing the Project](#)
- [System Editor](#)
- [Workspace Explorer](#)

Select Flash Security Protection

At the close of developing a production design, you may require flash protection of user code on the PSoC. You will want to ensure that the PSoC application code will not be accessible by any outside entity. PSoC Creator supports flash security selections from the design-wide resources **Flash Security** tab.

To Use this Feature:

Open the design-wide resources (<project>.cydwr) file from the [Workspace Explorer](#). Then, select the **Flash Security** tab.

Set up the flash security options as needed. Refer to [Flash Security Editor](#) for more information.

See Also:

- [Completing the Project](#)
- [Workspace Explorer](#)
- [Flash Security Editor](#)

Enable Write Once Latch Flash Protection

When completing a production level design, you may want to consider additional flash protection settings beyond the Flash protections described in [Select Flash Security Protection](#). PSoC products do support the write-once-latch (WOL) programming protection. With this option enabled, you can modify the NVL settings to enable the WOL option.

Once a device is programmed with the correct NVL configuration, it will be permanently locked. This feature should be used carefully as the device will not be recoverable once the WOL is enabled. This option is available in the PSoC Programmer tool through the Programming Options menu.

To enable a production HEX file to have the WOL option enabled, will need to include the protection code in the NVL configuration data. To learn how to do this, refer to the appropriate PSoC device programming specifications available online:

http://www.cypress.com/go/p3_p5_trm

See Also:

- [Completing the Project](#)
- [Select Flash Security Protection](#)

Evaluate General Programming Options

As your design is moving to completion, you will need to consider programming options and solutions. Cypress currently provides a general programming web page that discusses many critical programming topics. Specifically, you should review the available manufacturing programming solutions from your contract manufacturer, distributor - through a value added program - , or directly through a qualified third-party programming vendor.

You can navigate to this programming web page to find details on critical programming topics such as schematic requirements, programming specifications, available development programmers, and qualified third-party production programming vendors.

<http://www.cypress.com/go/programming>

See Also:

- [Completing the Project](#)

9 Reference Material



The following various documents are also included with this release

- Provided by Cypress
 - System Reference Guide
 - [Component Author Guide](#)
 - Customizer API Reference Guide
 - [Tuner API Reference Guide](#)
 - Warp Verilog Reference Guide
- [Provided by Third Parties](#)

The installed documentation is listed under **Documentation** on the PSoC Creator [Help menu](#).

Component Author Guide

The [Component Author Guide \(CAG\)](#) provides instructions and information that will help you create Components for PSoC Creator. The CAG is intended for advanced users to create sophisticated Components for end users to interact with PSoC Creator. However, there are some basic principles in the CAG that will also benefit novice users who may wish to create their own content.

The CAG is installed as part of the Component Development Kit, which also includes other documents and tools including:

- Datapath Configure Tool and [User Guide](#)
- [UDB Editor](#) and [User Guide](#)
- Customizer API Reference Guide
- Warp Verilog Reference Guide

You can find all these tools and documents on the Windows **Start** menu or PSoC Creator menus.

Tuner API Reference Guide

The Tuner API Reference Guide provides API reference information for tuner used to customize Components that can be tuned. This guide is generated from the tuner source code and comments. It is used in conjunction with the [Component Author Guide](#) for advanced users to create sophisticated Components for end users to interact with PSoC Creator.

This API reference guide is installed as part of the Component Development Kit. You can find it on the Windows **Start** menu or the PSoC Creator **Help** menu.

Third Party References

PSoC Creator includes third party compilers, as well as the documentation provided with them.

This documentation is located in the following default installation directories:

- **GNU Documentation** - <INSTALL_DIR>\import\gnu_cs\arm\4.4.1\share\doc\arm-arm-none-eabi\pdf
- **Keil Compiler Documentation** - <INSTALL_DIR>\import\keil\IC51\hlp

You can access these documents from the PSoC Creator [Help menu](#).

10 Contact Us



Thank you for contacting us. We value your suggestions and comments to help us improve PSoC Creator.

Use any of the following methods to contact us:

- For live help, call 1-800-541-4736 and select 3.
- For Technical Support, visit the [PSoC Software Community](#). This forum is monitored by Cypress applications engineers.

Please include as much of the following as possible.

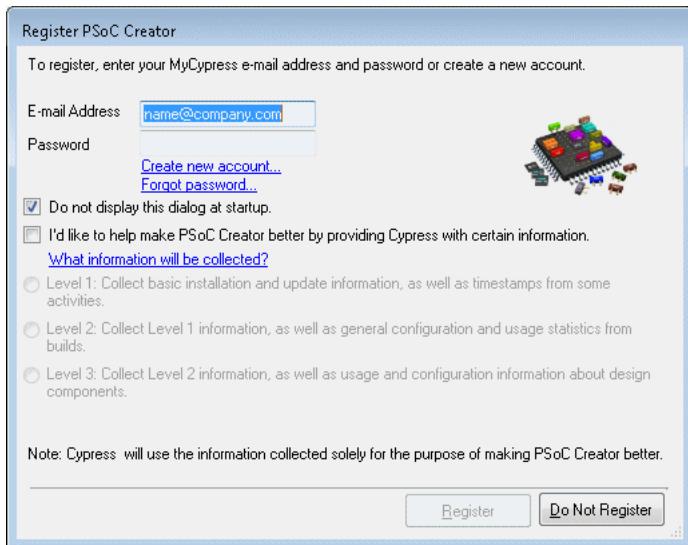
- Short problem description
- Build Version [Select **About** from the PSoC Creator **Help** menu.]
- O/S
- Error Message, if any
- What you were doing right before the problem occurred
- Severity of the problem (low, medium, high)

If possible, also include a screen shot.

11 Register PSoC Creator



The Register PSoC Creator dialog displays automatically at startup, unless you have already registered PSoC Creator or indicated not to display the dialog.



You can also open this dialog from the **Help** menu by selecting **Register...**

To Register PSoC Creator:

Enter your MyCypress email address and password and click **Register**. Click **Register Later** to close the dialog without registering.

- If the registration process fails, an error message will display on the dialog.
- If you cancel the registration process or if the network was unreachable after three attempts, the dialog will display a check box that allows you to never register the product.

To Create a New Account:

If you do not already have a cypress.com user account, click the **Create new account** link. This will open your web browser to the cypress.com account creation page.

If You Forgot your Password:

If you've forgotten your password, click the **Forgot Password** link. This will open your default web browser to the cypress.com forgot password page.

Information Gathered:

The registration process will transmit the following configuration information to Cypress and associate it with your cypress.com account.

- Cypress software product name and version
- Version of Windows and .NET
- CPU type and speed
- Memory size

If you are concerned about how we use the information gathered by our products, click the **How Cypress will use my information** link to open your web browser to a page that documents the kinds of information Cypress gathers during registration, the kinds of information gathered by the data collection system, and link to the Cypress privacy policy page.

Information Levels:

Please select one of the following levels of information to be sent to Cypress to help improve PSoC Creator:

- Level 1 – basic information
- Level 2 – level 1, plus configuration information from builds
- Level 3 – level 2, plus usage and configuration information from design Components

12 Index



3	
3rd Party Bootloader Support	506
3rd Party IDE	378, 379, 425, 443
A	
Add/Configure Components	8
Add/Edit Design	233
Address Breakpoint	538
Analog Device Editor	213, 217, 223
Analog Device Editor Context Menus	217
Analog Device Editor Debugging	223
Archive Development Tools	554
Archive the Project	555
Archiving	64, 555
Assembler Build Settings	321
Assigning A Core in a Multi	61
Attach to Target	550
Attribute	331
Auto	154
Auto Complete	185
Autocomplete	185
B	
Basic Design	19
Basic Hierarchical Design	39
Bootloader Bootloadable Project Export	507, 510, 511
Breakpoint Condition	543
Breakpoint Hit Count	544
Breakpoints Window	536
Build Menu	92, 310
Build Settings	311, 317, 318, 328
Build Toolbar Commands	309
Building	308
Built	164
Buses	172
Drawing	172
C	
Call Stack Window	546
Catalog Placement	269
Certain FM Devices Only	237
Clock Editor	225
Clocks	225
CMSIS-Pack	413
Code	63
Code Editor	182, 183, 184
Code Editor Context Menu Commands	184
Code Editor Toolbar	183
Code Example	15, 116
Code Explorer Window	186
Code Generation	311
Common Design Entry Toolbars	294
Compiler Build Settings	323

Completing	553	New Schematic	160
Component.....	102, 103, 162, 264, 267, 275, 534, 560	Parameter Validators	273
Exporting	275	Symbol	259
Component Author Guide	560	Symbol Parameters.....	271
Component Catalog.....	162	CSAttribute.....	331
Component Debug Window.....	534	Customize Commands.....	106
Component Installer.....	102	Customizer Build Settings	316
Component Item	264	Customizing	150
Adding	264	Framework	150
Component Terminals	267	CyElfTool Command Line Tool	374
Component Update.....	103	CyHexTool Command Line Tool	372
Component/Instance.....	50	CyPrjMgr Command Line Tool.....	362
Concepts	48	D	
Configure Component Parameters	164	Debug Build Settings.....	314
Configure Dialog Descriptions	279	Debug Select.....	247
Configure Local Clock.....	227	Debugger	92, 473, 475, 522, 523, 525, 528, 529, 530
Configure System Clocks	229	Using	522
Connecting Terminal.....	166	Debugger Indicators	528
Connectors	177	Debugger Menu Commands	92, 525
Contact Us.....	562	Debugger Status Messages	529
Control File	330, 332, 334	Debugger Toolbar Commands	523
Control File Format.....	332	Debugger Windows	530
Control File Pattern Matching	334	Debugging	26, 501
Copying	69	Debugging in Eclipse	452
a 69		Debugging using µVision	501
Project	69	Default Compiler	69
Core Design.....	61	Defining	269
Create New.....	445	Dependencies	107
Creating	52, 70, 72, 160, 259, 271, 273, 413	Description	67
µVision.....	413	Generating	67
Folders.....	72	Design	7, 26, 61, 128, 296, 304, 379, 381, 423, 424, 425, 426, 429, 431, 435, 440, 443, 472, 478, 489, 492
New File.....	70	Debugging	26
New Project	52	Export.....	425, 426, 429

Design Elements Palette	296	Eclipse Installation Configuration	443
Design Entry Options.....	128	Edit Menu.....	88
Design Entry Reserved Words	304	EEPROM Editor	256
Design to Makefile	489	Enable Device Protection.....	556
Design Tutorials.....	7	Enable Write Once Latch Flash Protection	558
Designs in Eclipse	381	Enumeration Types	113
Designs in Makefile.....	472, 492	Environment Options.....	138
Design-Wide Resources	207	Error Handling.....	551
Device Configuration	521	Evaluate General Programming Options.....	559
Device Selector.....	109	Existing File.....	71
Device Update Installer.....	112	Opening	71
Directives.....	336	Existing Project	57, 60
Directives Editor.....	252	Opening	57
disable schematic	179	Existing Project Item	60
Disabled Code	190	Export.....	275, 425, 426, 429, 431, 435, 440, 478, 484, 489, 510, 511
Disabling/Enabling Schematic Pages	179	Component.....	275
Disassembly Window.....	549	Design	425, 426, 429
DMA Editor	240	Export New	431, 435
DMA Wizard	241, 243, 244, 246	Exporting a Component	275
DMA Wizard Generated Code	246	Exporting a Design to	425, 426, 429
DMA Wizard Global Settings	243	Exporting a Design to Generated CMSIS.....	435
DMA Wizard Transaction Descriptors.....	244	Exporting a Design to Makefile.....	440
Dock Tool Windows	151	Exporting a FMO	489
Document Windows	77	Exporting a PSoC 3 Design to.....	478
Download.....	554	Exporting a PSoC 4.....	431
Download and.....	554	Expression Editor	293
Dragging Pin.....	209	F	
Draw Multiple	301	Family Migration Information	115
Drawing	172	File Menu	88
Buses.....	172	File/Line Breakpoint	539
E		Files	67, 88, 141, 379
Eclipse Bootloader Support	507	Reloading.....	141
Eclipse IDE	426	Find All References.....	188

Find and Replace.....	131
Find Code Example	116
Find in Files	194
Find Replace	191
Find Results.....	204
Fixed Attribute	331
Flash Programming	501
Flash Security Editor.....	254
Flashing	452
and Debugging in Eclipse	452
Flashing PSoC 5LP	452
Float Tool Windows	150
FM0	489, 492
Opening	492
Folders.....	72
Creating	72
For PSoC 6.....	413
Format Shape.....	292
Framework.....	73, 80, 150
Customizing	150
Framework Description.....	73
Framework Interface Components.....	80
Function Breakpoint.....	540
FX2LP Drivers	500

G

GCC Settings in µVision	496
General Tasks	51
Generate Verilog.....	118
Generated CMSIS	466
Opening	466
Generated Files	341
Generating.....	66, 67
Description.....	67

Project Datasheet.....	66
Getting	7, 241
Started	7
Go	206

H

Hello World Blinky	8
Help Menu.....	96
Hide Tool Windows	154
How To.....	46

I

IAR Bootloader Export Support.....	510
IAR IDE	429
IAR Project.....	406, 456
IDE Export Files	494
Import Component	119
Import into Eclipse.....	445
Inline Code Diagnostics	189
Input	372
Integrating	378
Interrupt Editor	238
Interrupts.....	238, 285

J

J-Link	473
--------------	-----

K

Keil µVision IDE	431, 478, 493
Keil µVision IDE Notes	493
Keil Compiler.....	375
Key.....	494
Keyboard Shortcuts.....	98
KitProg Drivers	500

L

Language Support Options	129
Library Component Project	30

Library Generation Build Settings	329
Library project.....	484
Lines	301, 372, 374
Linker Build Settings	326
Locals Window.....	548
Locked Route Cleanup	224
M	
Manual Placement	220
Mapper	329
Memory Watchpoint.....	542
Memory Window	530
Merge Dialog	120
MFT Editor	237
MiniProg3	500, 501, 517
Registering.....	500
Miscellaneous Export Notes	503
Modified Files	121
Move Tool Windows	153
Moveshape	302
Multi	381
My First Design.....	8
My Templates	17
N	
Names	175
New File.....	70
Creating	70
New Project	52
Creating	52
New Schematic.....	160
Creating	160
New Workspace.....	58
Adding	58
Notice Details	122

Notice List Window.....	84
O	
Obsolete Device.....	123
Ohm Meter	219
Opening	57, 71, 466, 492
Existing File.....	71
Existing Project	57
FMO	492
Generated CMSIS.....	466
Optimize Compiler Settings.....	554
Options Dialog.....	124
Output Window.....	83
P	
Pack	435
Pack Projects	466
Parameter Validators	273
Creating	273
Parameters	164
Peripheral Driver Library	317
Pin Editor	209
Pins	209, 235
Placer.....	329
Pre-Build/Post-Build Script.....	466
Print Preview	139
Probe	452
Program/Debug Options	135
Programmer	516
Project As	65
Saving	65
Project Build	381
Project Datasheet.....	66
Generating	66
Project Item	58

Project Management.....	124	Reloading.....	141
Project Menu	91	Files	141
Project Types.....	49	Replace in Files.....	196
Projects 49, 58, 64, 65, 66, 69, 91, 413, 466, 484, 494, 553, 555		Resource Meter.....	87
Archive.....	555	Review Device Datasheet	553
Completing.....	553	Route Editing.....	222
Copying	69	Router	329
Properties	140	Rubber-Banding	170
PSoC . 48, 73, 308, 335, 376, 379, 381, 406, 413, 423, 424, 431, 443, 456, 472, 473, 478, 484, 499			
PSoC 3 Projects for μVision IDE	484	S	
PSoC 5LP	456	Saving	65
PSoC 5LP Design to.....	431	Project As.....	65
PSoC 5LP Designs.....	424	Schematic	157, 158, 180, 289
PSoC 5LP Designs with.....	443	schematic comment	179
PSoC 6 Designs in Makefile	423	Schematic Editor	157, 158
PSoC Creator	48, 73, 308, 335, 472, 499	Schematic Editor Context Menu Commands	158
Understanding	48	Schematic Macro Editor	289
PSoC Creator Framework	73	Schematic Terminals.....	180
PSoC Creator Project	308	schematic, disable.....	179
Building.....	308	Search Result.....	205
PSoC Creator Toolchain Settings	499	Select Component Instance Debug Windows	535
PSoC UDBs.....	335	Select Datasheet.....	143
R		Select Debug Target	518
Reentrant Code in PSoC 3	376	Select Flash Security Protection	558
Reference Material	560	Select Optional Reset Line.....	557
Reference Tooltips.....	190	Select Programming Protocol.....	556
Register Details	546	Select Sheet Template	143
Register PSoC Creator	563	Select Source Clock.....	234, 235
Registering	500	Selecting	69
MiniProg3	500	Default Compiler	69
Registers Window.....	545	Set Build Configuration	555
Regular Expressions.....	199	Setting	406, 456, 473, 475
		Steps	456
		Up for Segger J-Link	473

Up for ULink2.....	475	Third Party References	561
Setup Subsection.....	352	To Line	206
Shapes	302	Tool Windows.....	76
Sheet Template Editor	290	Toolchain Build Settings.....	320
Sheet Template Page Setup.....	144	Tools Menu	96
Signal Name	145	Trace Debugger for PSoC 5LP	473
Source Code Control	350	Tuner API Reference Guide.....	561
Standard Toolbar.....	97	Tuner Communication Setup	147
Started.....	7	Types	75
Getting	7		
Starter Designs.....	15	U	
Static Timing Analysis.....	352	UDB Control Register.....	282
Step7	478	UDB Count7	286
Steps	456	UDB Datapath.....	279
Setting	456	UDB Design Elements Palette	277
Symbol.....	258, 259, 260, 263, 271	UDB Editor	276
Creating	259	UDB Properties	278
Symbol Editor	258, 263	UDB State Machine.....	287
Symbol Editor Context Menus	263	UDB Status Interrupt Register.....	285
Symbol Parameters	271	UDB Status Register	283
Creating	271	UDB Status Register Interrupt.....	285
Symbol Wizard.....	260	ULink Debugger Probes.....	475
System Clock APIs	232	ULink Pro and Segger J.....	475
System Clocks	232	Understanding.....	48
System Editor	247	PSoC Creator.....	48
T		Use Tabbed Documents	152
Tab Groups.....	77	User Commands	328
Target	318, 550	Using	156, 177, 298, 381, 443, 501, 522
Attach	550	Debugger	522
Target IDEs	318	Using Design Entry Tools.....	156
Terminal Name	146	Using Multiple Pages	177
Text.....	131, 297	Using Multiple Pages and Connectors	177
Text Editor Options.....	131	Using Text Substitution	298
		Using the Debugger	522

uVision.....	413, 466	Wire Labels and Names.....	175
uVision Bootloader Export Support.....	511	Wires.....	166, 175
V		Working.....	166, 180, 267, 297, 301, 302
Variable Watchpoints.....	541	Workspace	64, 81
View Menu.....	90	Workspace Explorer.....	81
W		Workspace/Project	48
Watch Window.....	531	Writing	63
Welcome.....	5	Code	63
Wide Clock	233	X	
Wildcards.....	203	XTAL Configuration.....	230
Window Menu	96	Z	
Windows	75, 76, 96	Zooming	303
Wire Labels.....	175		