

PSoC Designer: Integrated Development Environment

Getting Started 25-Minute Tutorial
Revision 1.0

CMS10006A
Last Revised: July 3, 2001
Cypress MicroSystems, Inc.

This tutorial of *PSoC Designer: Integrated Development Environment* is designed to demonstrate the use of the tools in a hands-on application using PSoC Designer, the ICE, Pod, and PSoC Pup™ board. Note that this is a basic project with the sole purpose of giving you a taste for what can be done. The media presentation for this tutorial can be accessed by double-clicking xxx.exe. The time for the demonstration is approximately 25 minutes. (Note: this video demo is still being created as of this release of this document).

This project demonstrates the use of two User Modules, the **DAC6** and **Timer16**. DAC6 is the 6-bit voltage output DAC and the Timer16 is a 16-bit down timer with period and capture registers. Complete functional descriptions for all User Modules are found in the data sheets contained within PSoC Designer.

The software project that will be developed in assembly code cycles through the 63 possible values of the DAC6 using the Timer16 module interrupt routine as a delay. The current digital value is then supplied to the DAC and LEDs attached to port 2 of the PSoC Pup™ board. The analog output of the DAC is routed through the analog buffer for column 3 and connected to port 0 pin 2 that is available on the user header of the PSoC Pup board. Using a voltmeter, the output analog voltage can be observed.

If difficulties are encountered with this project, additional information can be found in *PSoC Designer: Integrated Development Environment User Guide* as well as in the new redesigned PSoC Designer help system. A fully documented example for this project is available in the ...\`Examples` folder of the PSoC Designer installation directory. It is called `Example_DAC_output_28pin`.

For comprehensive details on developing systems, compiling, and assembling, see:

- *PSoC Designer: Integrated Development Environment User Guide*
- *PSoC Designer: C Language Compiler User Guide*
- *PSoC Designer: Assembly Language User Guide*
- *8C20000 Family of Devices Data Sheet*

For specific technical application questions please refer to our web site (www.cypressmicro.com) under the technical support forums.

<u>Overview</u>	2
<u>Section 1. Getting Started</u>	4
<u>Section 2: Create the DAC6 Project with Timer16</u>	5
<u>Section 3: Configure your Project</u>	8
<u>Section 4: Generate Application Files</u>	14
<u>Section 5: Develop System Software</u>	15
<u>Section 6: Debug the Project with the ICE</u>	18
<u>Section 7: You Are Done 25 Minutes... Summary</u>	25
Figure 1: New Project Dialog Box	5
Figure 2: New Configuration Dialog Box	5
Figure 3: Parts Catalog Dialog Box	6
Figure 4: Device Editor Default Entry	7
Figure 5: User Module Selections	8
Figure 6: Place User Modules Mode	9
Figure 7: DAC6 User Module Parameters	10
Figure 8: Timer16 User Module Parameters	10
Figure 9: Global Resources	11
Figure 10: AnalogOutBuffer_3 Port_0_2	12
Figure 11: Specify Pin-out Mode	13
Figure 12: PortPin P0[2] Settings	13
Figure 13: Application Generation Status	14
Figure 14: main.asm Source Code for Tutorial	15
Figure 15: Timer16_1INT.asm Source Code for Tutorial	16
Figure 16: Status Window	16
Figure 17: Debugger Subsystem	20
Figure 18: Breakpoints Dialog Box	21
Figure 19: ASM Watch Variables	22
Figure 20: Debugger Subsystem Toolbar	22

Section 1. Getting Started

There are two ways to use this tutorial:

1. In conjunction with the video media presentation xxx.exe. (still being developed)
2. As a stand-alone document to create your own application.

If you are using this tutorial alongside the video/media presentation, get your coffee cup and schedule yourself about 25 minutes. Hit Go and you will be off and running.

If you are creating this project on your own workstation you must have the current release of PSoC Designer installed and the ICE hardware connected. (PSoC Designer can be downloaded from the web site at www.cypressmicro.com). If this is not the case, refer to sections 2 and 9 in the *PSoC Designer: Integrated Development Environment User Guide*.


The arrow \Rightarrow symbol signifies actions by the user. This project walks the user through most elements of PSoC Designer.

Section 2: Create the DAC6 Project with Timer16

Creating a new project is the first step in the tutorial. In this section you will:

- ⇒ **Create a New Project**
- ⇒ **Create Project Directory and Files**
- ⇒ **Select a Part**
- ⇒ **Select a Programming Language**

Create a New Project

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

Create Project Directory and Files

⇒ Inside New Project dialog box, click once on **Create a new Configuration**, type **Tutorial_DAC** in the Project name field, and either type or **Browse** to designate the location of your new project directory.



Figure 1: New Project Dialog Box

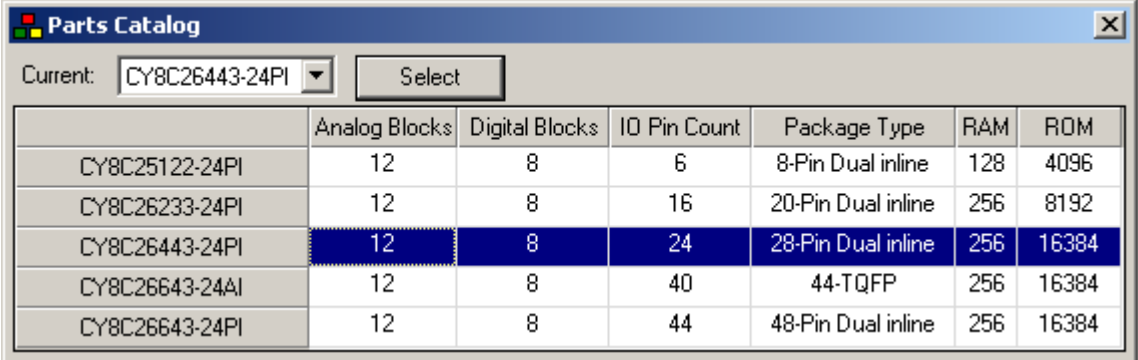
⇒ When finished, click **Next**. Once you click **Next**, you will see the New Configuration dialog box.



Figure 2: New Configuration Dialog Box

Select a Part

⇒ Click the drop-arrow in the Select Base Part field and select the **28-Pin Dual inline** part for this tutorial. Click **Select** to save your selection and exit the dialog box.



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

Figure 3: Parts Catalog Dialog Box

Select a Programming Language

Assembly language will be selected by default.

⇒ (Note that C will only be an option if the C Compiler has been enabled in your version of PSoC Designer. See *PSoC Designer: C Language Compiler Tutorial* for enabling instructions.)

⇒ Click **Finish**.

After clicking **Finish**, you will be in the Select User Module mode of the Device Editor subsystem (which is the default). You will see the ADCINC12 User Module and its data sheet. See the following figure.

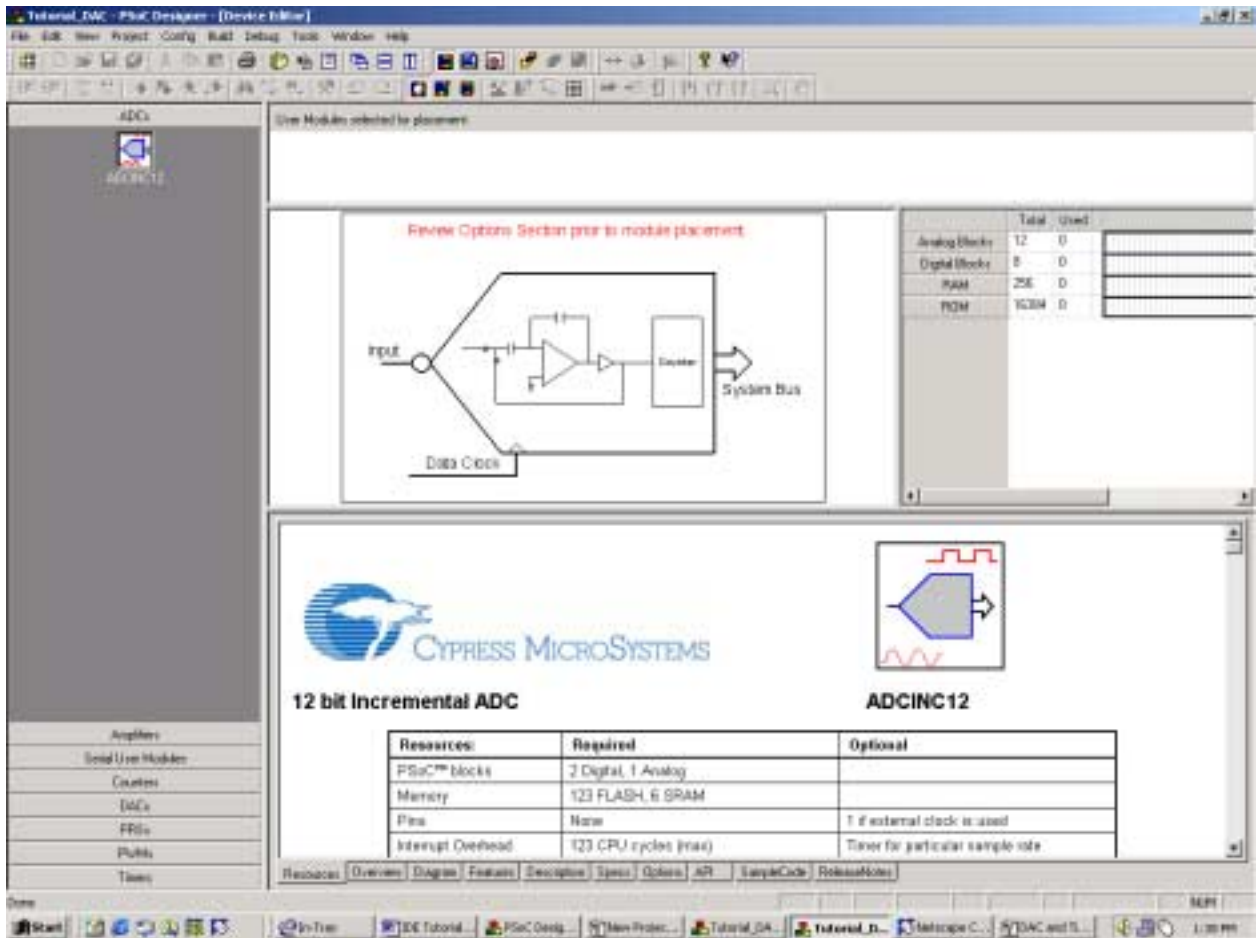


Figure 4: Device Editor Default Entry

In the left frame, you will see options of available User Modules grouped by type. To view the individual User Modules, click one of the group titles (ADCs, Amplifiers, Serial User Modules, Counters, DACs, PRSs, PWMs, Timers). The technical data sheet for the User Module you have clicked will be in the lower window. To quickly access specific information in the data sheet, click the different tab options (Resources, Overview, Diagram, Features, etc.).

To view the files and project folders that were created when you entered “Finish,” access **View >> Project**. The folders and files will all start with the project name, “**Tutorial_DAC.**” PSoC Designer will update these files and the interfaces between the User Modules upon application generation.

The Resource Manager window resides on the far upper right. With each module you add, the system updates the Analog Blocks, Digital Blocks, ROM, and RAM usage used by the current set of “selected” User Modules. If you attempt to select a User Module that requires more resources than are currently available, PSoC Designer will not allow the selection.

Tracking the available space and memory of configurations for your device is something you do intermittently during the whole process of configuring your target device.

Section 3: Configure your Project

Configuring your chosen device is the next step to programming your PSoC Microcontroller. It is a **six**-step process, all of which are performed in Device Editor:

1. **Select Applicable User Modules**
2. **Place User Modules**
3. **Configure User Module Parameters**
4. **Specify Global Resources**
5. **Make Interconnections**
6. **Specify Pin-out**

Step 1: Select Applicable User Modules

Here, we will be selecting the **DAC6** and **Timer16** User Modules. A User Module, as defined in PSoC Designer, is an accessible, pre-configured function that once placed and programmed will work as a peripheral on the target device.

- ⇒ Click the DAC title in the left frame then select the **DAC6** User Module.
- ⇒ Double-click the module. It will appear in the upper active window.
- ⇒ Click the Timers title and select **Timer16** by double-clicking.
- ⇒ View your selections in the upper active window.

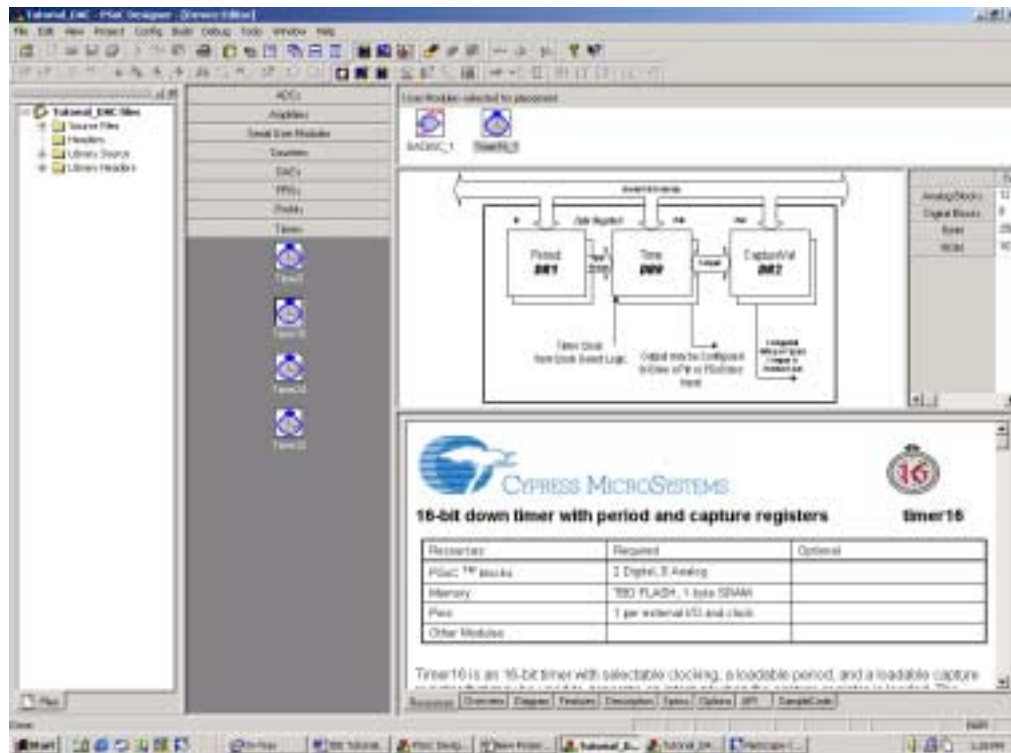



Figure 5: User Module Selections

Two additional features for configuring User Modules are remove and rename. At any time during device configuration you can add and remove User Modules to and from your device.

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

The development tool also allows the user to rename User Modules to company-specific naming conventions. To rename a User Module, right-click on the module, select Rename, and type a new name. All related files and routine calls will dynamically be renamed.

Step 2: Place User Modules.

Here, we will place the **DAC6** in analog block ASB13, and the **Timer16** in digital blocks DBA00 and DBA01. (Note: The 16-bit timer uses 2 digital blocks.)

⇒ To access the Place User Module mode of Device Editor, click the Place User Modules icon in the Device Editor toolbar. See Figure 6.



Figure 6: Place User Modules Mode

⇒ Single-click on the **DAC6** User Module icon.



Click the Next Position icon to advance the highlights to the next available location (identified with green cross-hatch background).

When you click the module, the first available location for the User Module on the device is highlighted. Note that if your User Module occupies a combination of blocks (both digital and analog), the active blocks (green cross-hatch) will advance as you click the Next Position icon and the inactive blocks (blue cross-hatch) will remain static.

⇒ Using the Next Position icon, advance placement position to analog block **ASB13** then click the Place User Module icon or right-click and select Place

Once you have placed the module, it will appear on the device, color-coded, bearing the module name and its position on the device.

⇒ Repeat the previous action to place the **Timer16** User Module, however, this time leave Timer16 in its default position, digital blocks **DBA00** and **DBA01**.

Step 3: Configure User Module Parameters

User Module parameters are the internal block specifications defined in the applicable data sheets. Each User Module must be configured individually. As you single-click a selected User Module you can view its parameters under User Module Parameters in the lower left window.

⇒ For this tutorial, the **DAC6** requires that the analog output bus be enabled. Under User Module Parameters, set A_Bus to **Enabled** and the ClockPhase to **Normal** for the **DAC6**. See Figure 7.



Figure 7: DAC6 User Module Parameters

Configure the **Timer16** User Module as follows:

⇒ Timer User Modules are quite flexible and provide greater configurability. For this tutorial, set the following User Module Parameters for the **Timer16** User Module:

Clock: Set to **24V2**. This will provide a 93.75 kHz clock. The value is the CPU_Clock 24 MHz calculated by divided by $24V1 = 16$ and then $24V2 = 16$.

Input: Set to **High**. We will not be using the Capture Val.

Interrupt_Type: Set to **Terminal_Count**. This will pick the rising edge of the terminal count. For further details, see section 8 in the Device Data Sheet (accessed at <http://www.cypressmicro.com/>).

LSB_Captureval: Set to **0**. This is the DR2 register described in the Timer16 User Module Data Sheet (accessed in PSoC Designer). Because we have set the **Input** to **High**, there is no rising edge, hence no capture.

MSB_Captureval: Set to **0**.

LSB_Period: Set to **0**.

MSB_Period: Set to **155**. This will set the terminal count to a rate of $615.4 / 256$ Hz.

Output: Set to **None**, as this project is completely interrupt driven.



Figure 8: Timer16 User Module Parameters

Step 4: Specify Global Resources

Global resources are hardware settings that determine the underlying operation of the entire part. Such settings include the CPU_Clock, Analog Power, and Reference Power. They are located in the upper left window.

To update the project global resources, click each option and make applicable selections.

⇒ Select the following parameters for this project:

CPU_Clock: Set to **3MHz**.

32K_Select: Set to **Internal**, as no external crystal is needed.

PLL_MODE: Set to **Disable**. PLL can only be enabled when 32K_Select is External.

Sleep_Timer: Set to the default value of **512_Hz**.

24V1= 24MHz/ N: Set to **16**. This signal is used for the DAC6.

24V2=24V1/N: Set to **16**. This sets the Timer16 clock.

Analog Power: Set to **ON**. This is required to power up all the analog sections.

Ref Power: Set to **LOW**. Ref_Power adjusts the power output of the references. It should be low for most projects. If you have several analog PSoC blocks you may want to use the **High** setting.

Ref Mux: Set to **Refs=AGND+/-Bandgap**, which is the default.

Op-Amp Bias: Set to **Low**, which is the default (lower power consumption and slower slew rate).

SC Power: Set to **ON**. This powers the Switch Capacitor Analog PSoC blocks. DAC6 is in a Switch Capacitor Analog PSoC block.

A_Buff_Bypass: Set to **Drive**, which is the default. A_Buff_Bypass bypasses the buffer and is used for characterization purposes only. It should be left in **Drive**.

A_Buff_Power: Set to **Low**, which is the default. A_Buff_Power selects the power level of the buffer and is a tradeoff between drive output power and power consumption. Low is adequate for most projects.

SwitchModePump: Set to **Off**.

VoltMonRange: Set to **3.3V**.

VoltMonThreshold: Set to **80%**.



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

Figure 9: Global Resources

Step 5: Make Interconnections

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

Connect the analog output buffer to Port 0_2, which will result in output to the “Pup.” The Pup is the hardware containing the LEDs. It is provided with the Software Development Kit.

⇒ Set interconnectivity parameters by left-clicking the lower right hand output bus “**pa3**” then double-clicking **Port_0_2**.

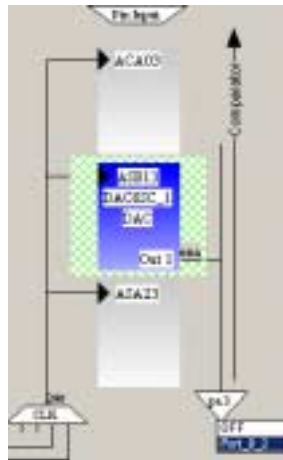


Figure 10: AnalogOutBuffer_3 Port_0_2

Step 6: Specify Pin-out

When you specify a PSoC block connection to a pin you are making a physical connection to the hardware of the M8C device. These configurations will later be emulated and debugged within the device simulation unit (In-Circuit Emulator).

⇒ Click the Specify Pin-out icon.



Figure 11: Specify Pin-out Mode

Enable AnalogOutBuffer_3 to connect to **Port_0_2** by right-clicking the Analog Bus and double-clicking **Port_0_2**.

⇒ At PortPin P0[2] (in left frame), select **AnalogOutBuf_3** and leave the drive at its default, **HighZ**. This will also update the Pin-out view of the physical Hardware, P0[2] should turn to dark green to reflect Analog Output.

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

Figure 12: PortPin P0[2] Settings

You have completed device configuration for the project. Now would be a good time to save your project!

⇒ To save the project, click File >> Save Project.


Section 4: Generate Application Files

Generating application files is the final step to configuring your target device.

When you generate application files, PSoC Designer takes all device configurations and updates existing assembly-source and C compiler code files (including the project library source, *PSoCConfig.asm*) and generates API (Application Program Interface) and ISR (Interrupt Service Routine) shells.

At this time, the system also creates a data sheet based on your part configurations that can be accessed in the View menu under Data Sheet. This data sheet is specific to your project.

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

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

Once the process is complete, a graphic dialog box will appear informing you that the application code has been generated successfully. Now, click Application Editor to begin source programming.

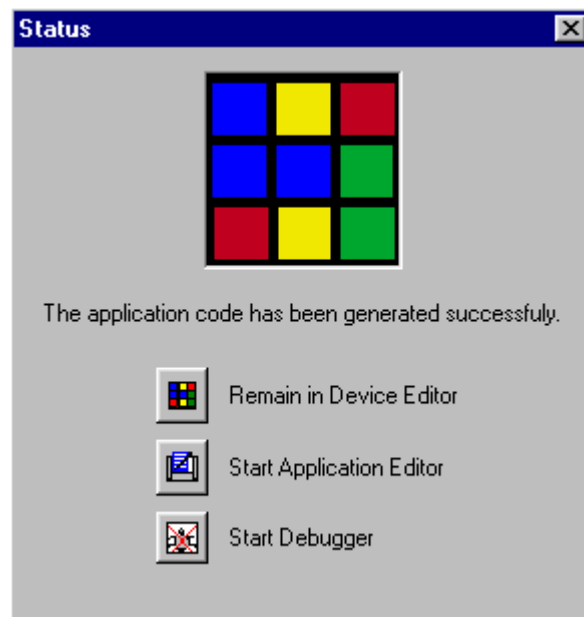


Figure 13: Application Generation Status

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

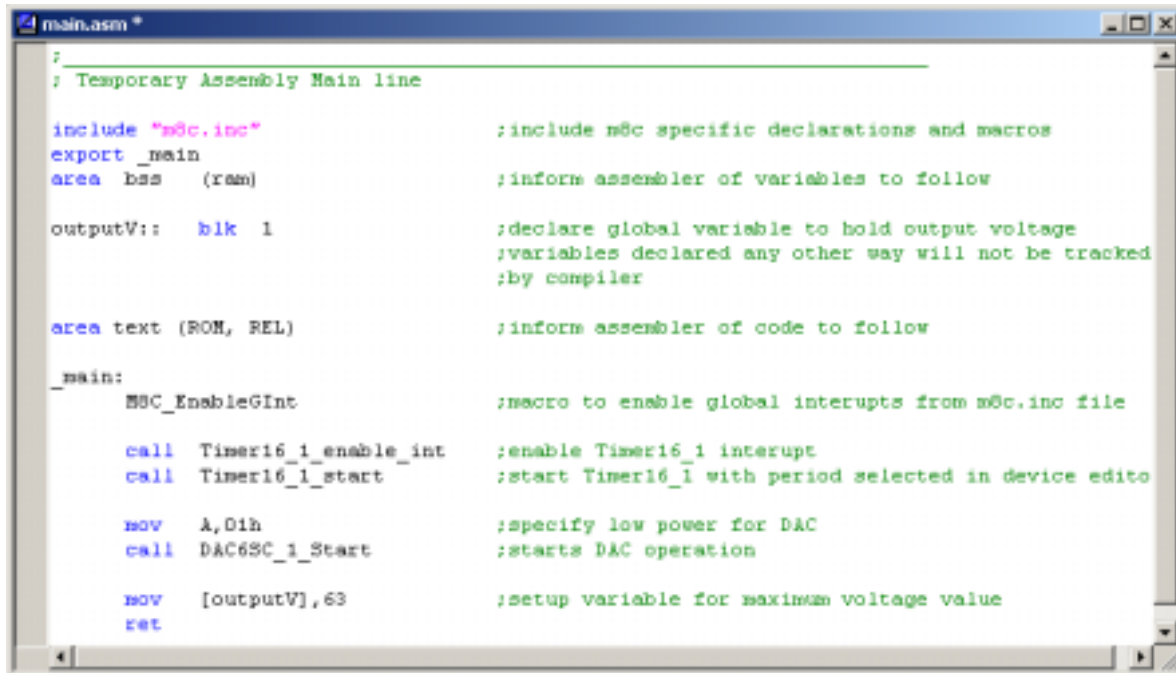
Source Programming

At this point in the project you are ready to develop your system software. The interfaces and interrupt tables (API and ISR files) have been created dynamically, based on your device configuration. Now you will:

- ⇒ **Develop Source Code**
- ⇒ **Build the Project**

Develop Source Code

⇒ In the source tree under Source Files double-click *main.asm* and type the following source code:



```
main.asm
;
; Temporary Assembly Main line

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

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

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

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

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

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

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

Figure 14: main.asm Source Code for Tutorial

⇒ In the source tree under Library Source double-click *Timer16_1INT.asm* and type the following source code:

```

Timer16_IINT.asm *
include "m8c.inc"           ;include m8c specific declarations
include "Timer16_1.inc"

export      Timer16_IINT
export      _Timer16_IINT

area bss(RAM)              ;declare variables here

area text(ROM,REL)

Timer16_IINT:
_Timer16_IINT:
    dec    [outputV]        ;Decrement voltage variable and place on port 2
    mov    A,[outputV]
    mov    reg[PRT2DR],A

    call   DAC6SC_i_WriteOffsetBinaryStall ;write new voltage variable to DAC6
                                                ;waiting for optimum write time

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

branch1:
    ret

```

Figure 15: Timer16_IINT.asm Source Code for Tutorial

This would be a great time to save your project!

Build the Project

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

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

⇒ To build the current project, click the Build icon  in the toolbar.

The status (or error-tracking) window in the bottom left-hand corner of Application Editor is where the status of file building resides. Each time you build your project, the status window is cleared and the current status entered as the process occurs.

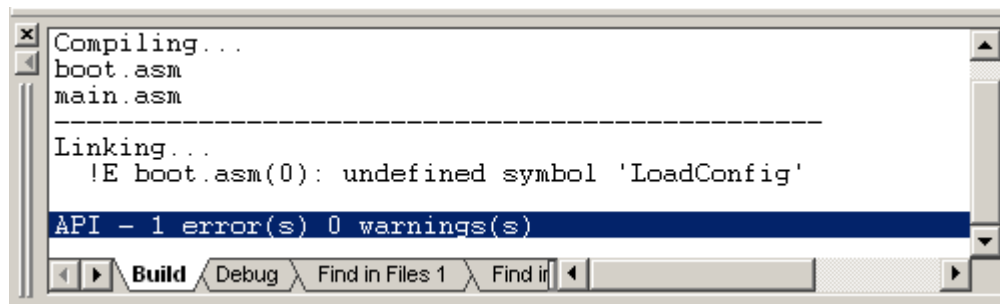



Figure 16: Status Window

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


If errors do occur, modify your files and re-compile using the Compile icon . After successful compilation, the project must be re-built.


Debug the Project

The PSoC Designer Debugger provides in-circuit emulation (ICE) that allows you to test the project in a hardware environment while viewing and debugging device activity in a software environment. In this section we will:


- ⇒ **Enter Debugger Subsystem and Connect to Test Pod**
- ⇒ **Employ Test Strategies to Monitor the Project**

Enter Debugger Subsystem and Connect to Test Pod

⇒ Access the Debugger subsystem by clicking the Debugger icon .

⇒ Physically connect your computer and PSoC Designer to the In-Circuit Emulator. The ICE must be connected before you can download and debug your project. Click the Connect icon .

Details for connecting the ICE can be found in section 9 of the *PSoC Designer: Integrated Development Environment User Guide*.

⇒ Download the project .rom file to the Pod by clicking the Download to Emulator icon .

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

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

An important general rule to remember before downloading is to make sure there is not a part (M8C) in the programming socket of the Pod. Otherwise, debug sessions will fail.

Employ Test Strategies to Monitor the Project

In this section we will briefly touch on the following system features:

- **Debug Strategy**
- **Trace File**
- **Breakpoints**
- **Watch Variables**
- **Execute the Program**
- **Monitor Program Output**
- **View Registers**
- **Dynamically Changing Accumulator Value**
- **Verify Program Functionality**
- **Dynamically Changing RAM to Demonstrate Code/Path Coverage**

Debug Strategy

Following is a summary of our debug strategy:

- The ICE provides the ability to select variables of interest (from the users program) that can be monitored real time. These variables are called Watch Variables. The Watch Variable from the code that we will view/monitor is the “**OutputV**” value. This value should range from 63 to 0 and then reset again.
- We will set breakpoints on both routines to see the decrementing of the **OutputV** value as well as the transfer to the Accumulator.
- Finally, we will see the output to Port 2 register.
- Global reference register variables that were initialized in the Device Editor will also be verified.

The debugging window has many useful features: Register Memory space, Watch Variable list, output files, source files, and CPU registers (see the device Data Sheet, section 2.0 for details).

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

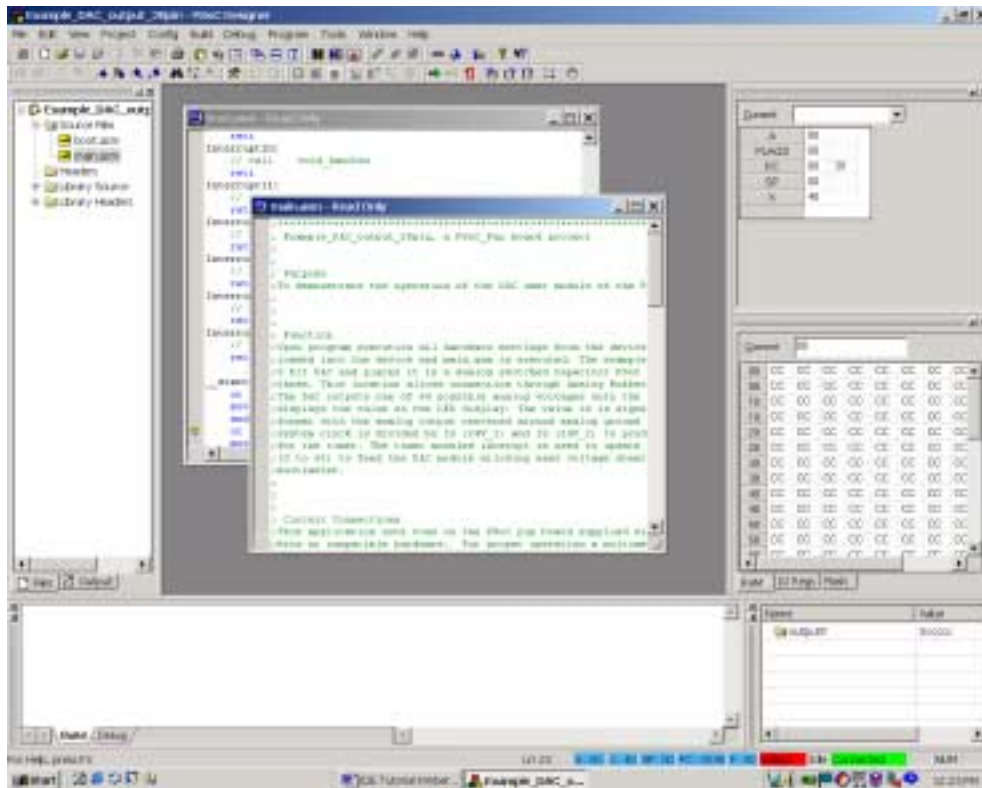



Figure 17: Debugger Subsystem

Trace File

⇒ To view the trace file, click the Trace icon . The file should be empty at this point.

The trace feature of PSoC Designer enables you to track and log device activity at either a high or detailed level. The contents can be selected to include register values, data memory, and time stamps. The Trace window displays a continuous listing of program addresses and operations from the last breakpoint. Each time program execution starts, the trace buffer is cleared. When the trace buffer becomes full it continues to operate and overwrite old data.

Breakpoints

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

For our example we will set two breakpoints:

- ⇒ Open the *main.asm* file by highlighting it in the source tree. (If the source tree is not showing, access View >> Project.)
- ⇒ Scroll down in the file to the statement: `M8C_EnableGInt.`
- ⇒ Go to the left margin next to this statement and left click. A red dot will signify that you have just set a breakpoint.

⇒ The second breakpoint we will set in the *Timer16_1Int.asm* file. Open *Timer16_1Inst.asm*.

- ⇒ Scroll down to the statement “`dec [outputV]`”
- ⇒ Click on the left margin next to this statement. A red dot will signify that you have set a breakpoint.

⇒ To view all the breakpoints you have set, access Debug >> Breakpoints.
(Note: the line number of the breakpoints (90 and 92) may be different)

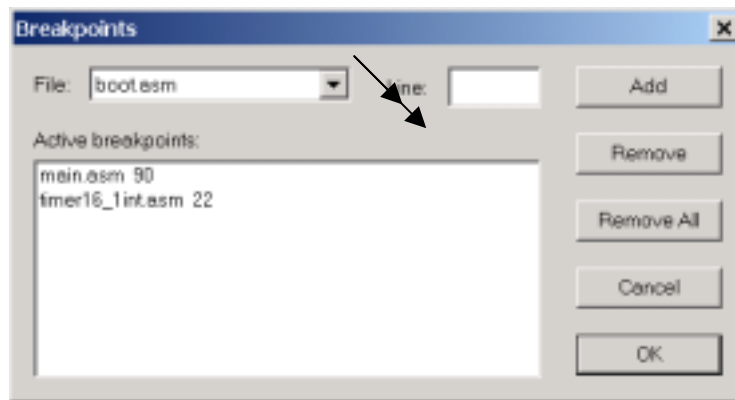


Figure 18: Breakpoints Dialog Box

⇒ Press OK to close the Breakpoint Dialog Box.

Watch Variables

We will set a watch on the variable “**OutputV**.” The address for this variable is 0Eh. This can be found in the Trace window or in the *output.mp* file.

⇒ To set **OutputV** as a Watch Variable, access Debug >> Watch Variables and fill in the following details:

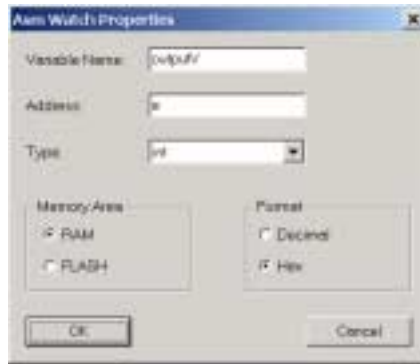





Figure 19: ASM Watch Variables

Execute the Program

We are now ready to start execution of the program. Our example should stop on the first breakpoint in *main.asm*. A yellow arrow will point to the `M8C_EnableGInt` line of code when this happens.

⇒ Click the Start/Go icon  to execute the program.

⇒ To view the trace buffer click the Trace icon . The buffer will contain all the code initialization and *boot.asm* executable codes.

⇒ Click the Start/Go icon  to execute the program. The program will stop on the second breakpoint in *Timer16_1INT.asm*, `dec [outputV]`

⇒ Use the “Step Into...” functions to execute the next several instructions. Watch what happens to `outputV`, the Accumulator and the LEDs on the PuP at each step.

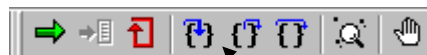


Figure 20: Debugger Subsystem Toolbar

Monitor Program Output

The **OutputV** value will be put into the Accumulator and then transferred through a call to the DAC routine.

The LED output should be equivalent to the output on Port 2. Note that the four LEDs closest to the header are not used.

View Registers

Bank Registers

At this point in our tutorial we can view the I/O registers. The results of the “**OutputV**” variable will be output onto Port 2 data line, which is located at I/O Address Register 008.

⇒ In the right frame of the Debugger subsystem, click the Flash (Bank0) tab and go to memory location 008. This is the Port 2 data line, which will be output to the Pod LED display. At this point, the LEDs should be lit representing this value.

View RAM Registers

⇒ View the **OutputV** variable in the RAM section of the Debugger. Click the RAM tab and scroll over to the address location 0E. The value is “**3E**” This is also shown in the Watch Variable window in the lower right corner of the subsystem.


Dynamically Changing Accumulator Value

Execute the following actions to see PSoC Designers’ ability to change the Accumulator (A):

⇒ Remove all Breakpoints (Debug>>Breakpoints and click Remove All, or click on the Breakpoints to remove them.)


⇒ Set a Breakpoint in *Timer16_1INT.asm* at the following line,

```
“mov      reg[PRT2DR], A”
```

⇒ Click the Start/Go icon  to execute the program.

⇒ Access View >> Debug Windows >> CPU Registers.


⇒ Double Click the data field for A, enter “**00.**”

⇒ Use the Step Into icon  to execute the next line of code.

Results: The Accumulator value is no longer the “**OutputV**” variable. All LEDs should now be off.


Verify Program Functionality

The program should be cycling through the **OutputV** value from 63-0 decimal. The LEDs should cycle through the output values sequentially.

⇒ To test functionality, click the Reset  icon.

⇒ To clear all Breakpoints, access Debug >> Breakpoints and hit the Remove All button.

⇒ Execute normal program operation by clicking the Start/Go icon .



⇒ After program executes through several cycles of LEDs, click the Halt  icon.

Verify the values for the LEDs (Port 2 output), Watch Variable “**OutputV**,” and the Accumulator (A). They should all be the same.

A good debugging strategy is to force path coverage through logical branches. Let’s see if the **OutputV** is reset to 63 in the *Timer16_1Int.asm* routine.

Dynamically Changing RAM to Demonstrate Code/Path Coverage

The ICE allows dynamic editing of memory. We will change the **OutputV** value to force path coverage through the “jnz” logical branch.

- ⇒ Set a Breakpoint in *Timer16_1Int.asm* at the command
“`mov A, [outputV]`” (after the call to the DAC6 routine).
- ⇒ Click the Start/Go icon  to execute the program.
- ⇒ Click the RAM tab and change the **OutputV** value to “00.”
- ⇒ Use the Step Into icon  to execute the next several lines of code.

The “jnz” command should fail and the “**OutputV**” value will be reset to “3fh.”

Side Note to Tutorial: You have now successfully built and tested your project. If you were developing a system you would be ready to program the part.

Programming the part stores the ROM data directly in the FLASH memory of the part. The Cypress MicroSystems device can be reprogrammed multiple times due to its FLASH Program Memory.

To program a part follow these steps:

- ⇒ Click the **Program Part** icon .
- ⇒ Place part to be programmed on Pod when prompted and select the .rom file from the ...\`output` folder of your project directory.

NOTE: The emulator socket cannot be connected during programming.

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

Section 7: You Are Done 25 Minutes... Summary

Results

You have just programmed and created your part. The expected output for this tutorial uses the LEDs on the Pup board. The LEDs should sequence from 0 to 63 continually. The analog output of the DAC is routed through the analog buffer for column 3 and connected to port 0 pin 2 that is available on the user header of the PSoC Pup board. Using a voltmeter, the output analog voltage can be observed.

Note that you have only used 2 digital blocks and 1 analog, leaving lots of room to spare.

For you advanced users, try the following changes to the project:

- Change the Timer16 User Module Timer8
- Change the position of the DAC6 user module
- Change the position of the Timer16 User Module