Customizing PSoC Designer™ User Modules

Doc. # 001-84020 Rev. *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

Copyrights

© Cypress Semiconductor Corporation, 2012-2013. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.

PSoC® is a registered trademark of Cypress Semiconductor Corporation. All products and company names mentioned in this document may be the trademarks of their respective holders.

Purchase of I2C components from Cypress or one of its sublicensed Associated Companies conveys a license under the Philips I2C Patent Rights to use these components in an I2C system, provided that the system conforms to the I2C Standard Specification as defined by Philips. As from October 1st, 2006 Philips Semiconductors has a new trade name - NXP Semiconductors.

Flash Code Protection

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

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

Contents

1. Introduction **5**

1.1 Overview 5
1.2 Acronyms Used in the Document 5
1.3 Documentation Conventions 5
1.4 Document Revision History 6

2. Creating a Custom User Module **7**

2.1 Step 1: Copy and Name the User Module 7
2.1.1 API Register List 11
2.1.2 Modifying the UM 12
2.2 Step 2: Add Interrupt 12
2.3 Step 3: Remove Interrupt Type Parameter 14
2.4 Step 4: Add Interrupts to SHAPE 15
2.5 Step 5: Add Interrupt Handler 18
2.6 Step 6: Change APIs 19
2.7 Step 7: Edit the .inc file 20
2.8 Step 8: Edit the .h File 21
2.9 Step 9: Edit Placement Files 22
2.10 Verify All Modifications 23
2.11 Distribution 23
2.11.1 Exporting the User Module 23
2.11.2 Importing the User Module 24

3. Extending the Custom User Module **27**

3.1 Add Wizard 28
3.1.1 Step 10: Create C# Package - General Rules 28
3.1.2 Step 11: Attach *.dll to UM 33
3.2 Verify Wizard Availability in Custom UM 35
3.2.1 Step 12: Update both xml and h files to manage time constant 35
3.2.2 Step 13: Implement Required Functionality in Wizard 38
3.3 Add Pre-Processing Javascript 41
3.3.1 Step 14: Create JS file 41
3.3.2 Step 15: Attach Pre-Processing Javascript to UM 41
3.3.3 Step 16: Implement Required Functionality in Pre-/Post-Processing JS 42
3.4 Add Placement Javascript 46
3.4.1 Step 17: Attach Placement Javascript to UM 46
3.4.2 Step #18: Implement Required Functionality in Placement JS 47
3.5 Add Code Generation Javascript 48
3.5.1 Step 19: Attach Code Generation Javascript to UM 48
3.5.2 Step 20: Implement Required Functionality in Code Generation JS 50
3.6 Add Design Rule Check Javascript 50
3.6.1 Step 21: Attach Design Rule Check Javascript to UM 50

3.6.2 Step 22: Implement Required Functionality in Design Rule Check JS 52
3.7 Add Un-placement Javascript 52
3.7.1 Step 23: Attach Un-placement Javascript to UM 52
3.7.2 Step 24: Implement Required Functionality in Un-placement JS 54

A. Appendices 55

A.1 Appendix A 55
A.1.1 Timer16xPackage\Timer16xPackage\Timer16xPackage.cs 55
A.1.2 Timer16xPackage\Timer16xPackage\WizardForm.Designer.cs 57
A.1.3 Timer16xPackage\Timer16xPackage\WizardForm.cs 61
A.2 Appendix B 66
A.2.1 Timer16X\Ver_2_6\scripts\EventHandlers.js 66
A.3 Appendix C 68
A.3.1 Timer16X\Ver_2_6\scripts\Timer16xRules.js 68

1. Introduction

1.1 Overview

The objective of this guide is to create an improved user module (Timer16X), using the old user module (Timer16) as a template, with the help of the “User Module Customization Wizard” available in PSoC Designer™ 5.4. This guide describes how to modify an existing user module to meet your needs. The “User Module Customization” feature allows you to create a copy of an existing user module and export it.

Each user module is a combination of information on the interconnections of PSoC resources and the software used to control them. It is possible to generate new UMs or customize existing UMs. Different UMs can be combined to produce a new UM. These new UMs can be similar to the old ones, with no hardware changes and only with API changes.

1.2 Acronyms Used in the Document

Table 1-1. List of Acronyms

Acronym	Description
LSB	Least significant bit
MSB	Most significant bit
CRET	Close, Reopen with Explorer to Test
ISR	Interrupt service routine
API	Application Programming Interface

1.3 Documentation Conventions

Table 1-2. Document Conventions for Guides

Convention	Usage
`Courier New`	Displays file locations, user entered text, and source code: `C:\ ...cd\icc\`
Italics	Displays file names and reference documentation: Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*.
[**Bracketed, Bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]

Table 1-2. Document Conventions for Guides *(continued)*

Convention	Usage
File > Open	Represents menu paths: File > Open > New Project
Bold	Displays commands, menu paths, and icon names in procedures: Click the **File** icon and then click **Open**.
Times New Roman	Displays an equation: 2 + 2 = 4
Text in gray boxes	Describes cautions or unique functionality of the product.

1.4 Document Revision History

Table 1-3. Revision History

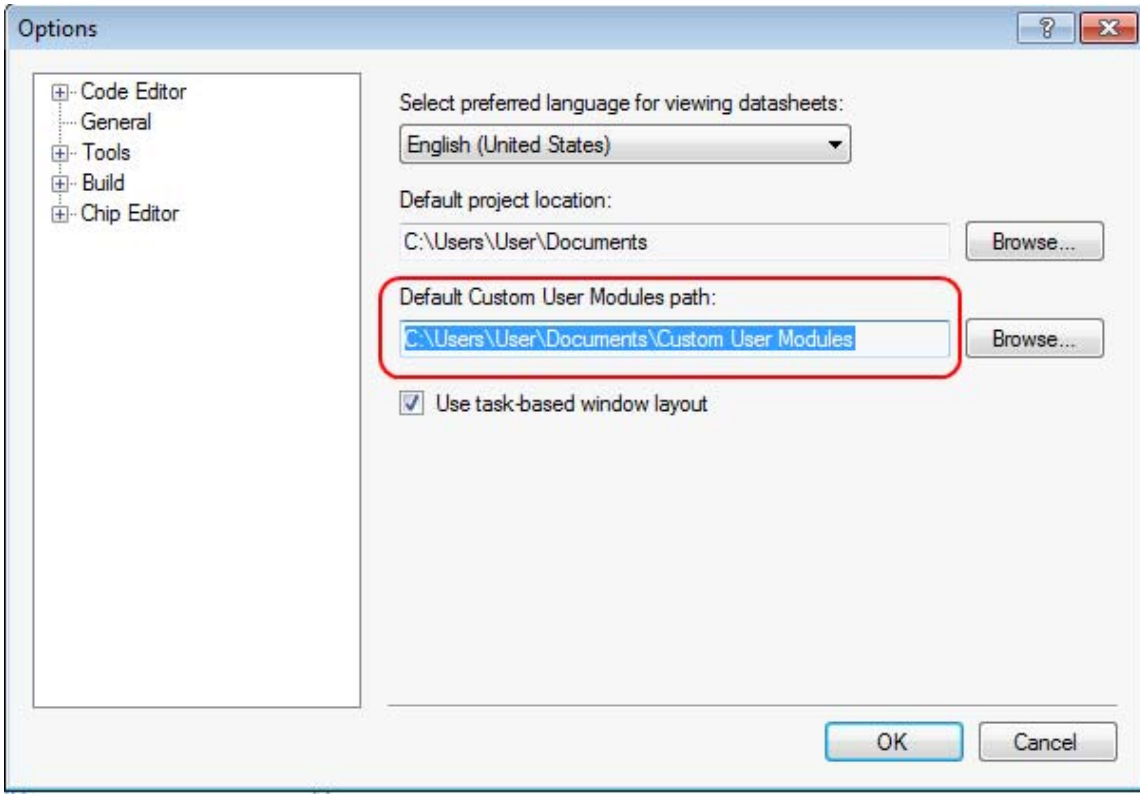
Revision	Date	Description of Change
**	12/21/2012	New specification.
*A	4/25/13	Added Extending section.

2. Creating a Custom User Module

The objective of this project is to build an improved Timer16 User Module, using the existing user module. The upper digital block of the present Timer16 has a parameter selection to generate interrupt on Capture, Terminal Count, and Compare True. The goal is to create a new UM (Timer16X) that generates a Terminal Count interrupt for the upper digital block and generates a Capture interrupt for the lower digital block.

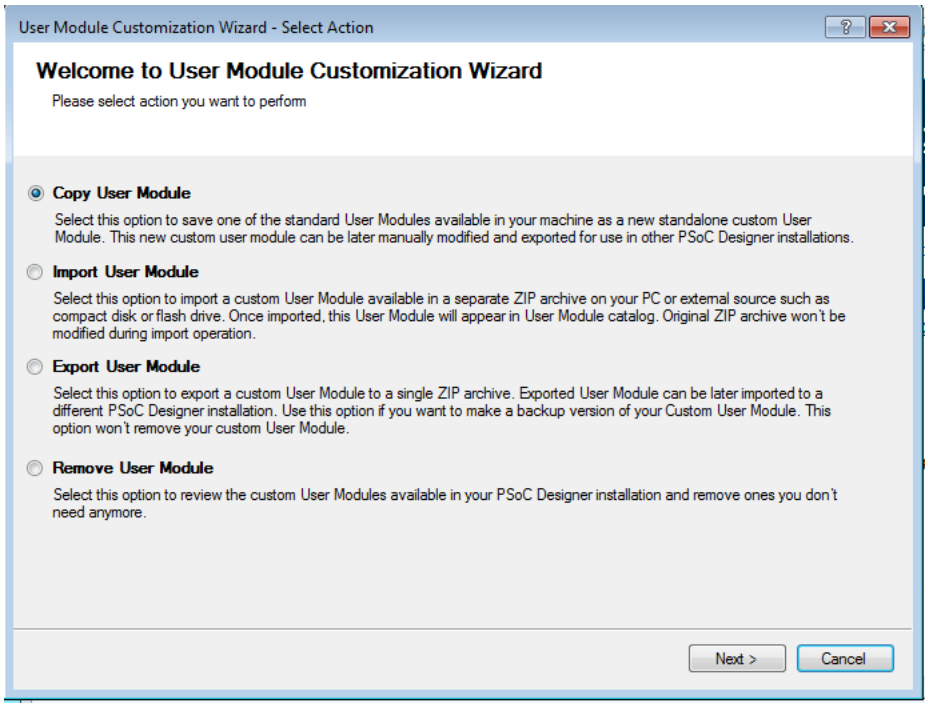
2.1 Step 1: Copy and Name the User Module

1. Launch PSoC Designer.
2. Select **Tools** > **Options** > **General** and set the default location in the 'Default Custom User Module path' tab to store and process custom UMs. There is only one folder with custom UMs in each PSoC Designer installation.

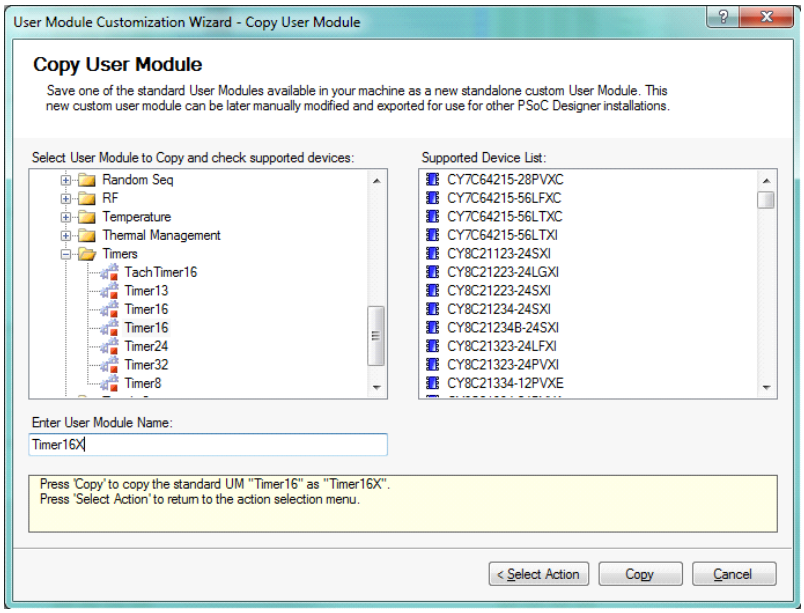


3. Go to **Tools** > **User Module Customization Wizard**.

4. Select the **Copy User Module** option and click **Next**.

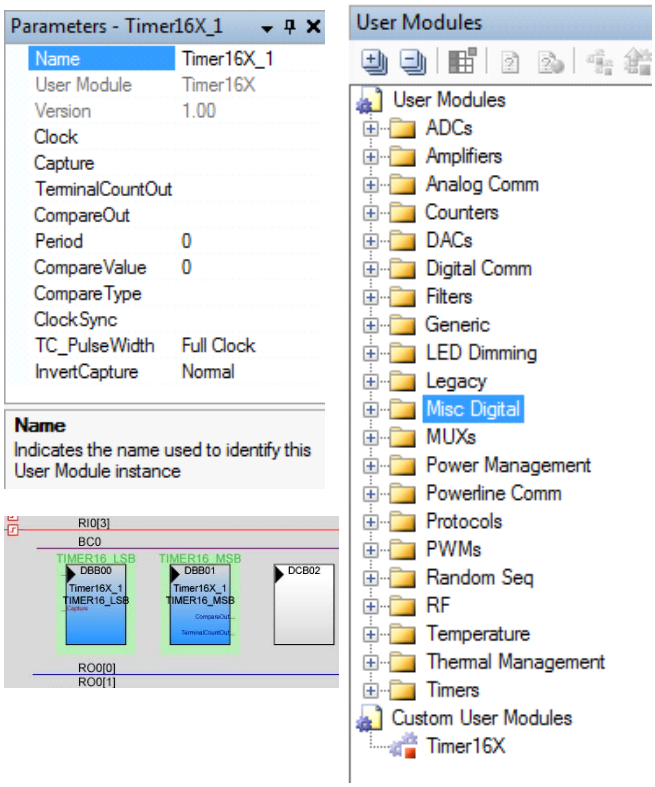


5. Choose ‘Timer16’ in the User Module tree.
6. Enter the name Timer16X.
7. Observe the list of devices that support the UM in the “Supported Device List” box.
8. Click **Copy** and then click **Close**.

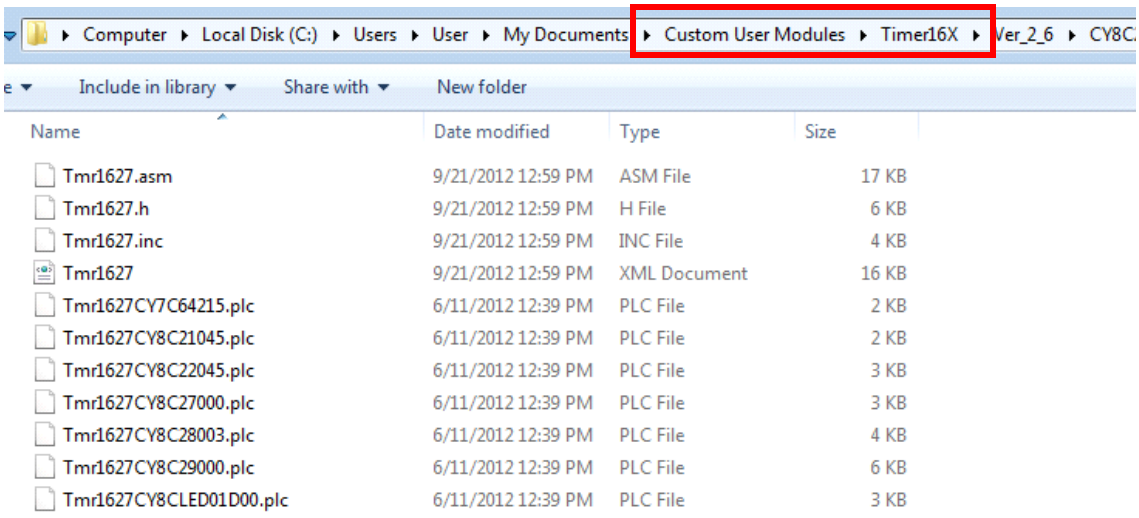


9. Create the project based on the CY8C29666-24LFXI part (or any other part from the ‘Supported Devices’ List).
10. Verify that the UM is present and you are able to place it.

11. For UMs that have significant changes in API but no hardware changes, this is your chance to change the APIs to the way that you think they should be, while leaving the base UM hardware configuration unchanged.



12. Close PSoC Designer.
13. Go to the Timer16X folder in the folder with custom UMs.



14. Open the CY8C27 UM sub-folder and open *the* Tmr1627.xml file with your preferred text editor
15. This file has four main sections:
    a. SHAPE
       For regs/bit fields that are set and do not change.

b. PARAMETER_LIST
For bit fields that are set by user selection.

c. API_REGISTER_ALIAS_LIST
Allows appropriate names of registers.

d. API_FILE LIST
Lists the API files to be generated

Tmr1627.xml

```
<!--
*******************************************************************************
*******************************************************************************
**  FILENAME: Tmr1627.xml
**   Version: 2.6, Updated on 2012/10/30 at 20:40:30
**
**  DESCRIPTION: 16-bit Timer Personalization/Parameterization specificati
**               for the 22/24/27/28xxx PSoC family of devices
** - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
**      Copyright (c) Cypress Semiconductor 2012. All Rights Reserved.
*******************************************************************************
*******************************************************************************
-->
<PSOC_DEVICE_DB>
<USER_MODULE_LIST>
<USER_MODULE NAME = "Tmr1627"
             TYPE = "PSOC_TIMER"
          VERSION = "2.6"
              ROM = "93"
              RAM = "0"
             HTML = "Timer.html"
              PDF = "CY8C27\Timer27.pdf"
             ICON = "Timer16.ico"
         METAFILE = "Timer_Block27.emf"
       API_PREFIX = "Tmr1627"
         API_PATH = "\CY8C27"
>
   <SHAPE SHAPE_TYPE = "BLOCKLIST">
     <BLOCK_LIST>
   </SHAPE>

   <PARAMETER_LIST>

   <API_REGISTER_ALIAS_LIST>

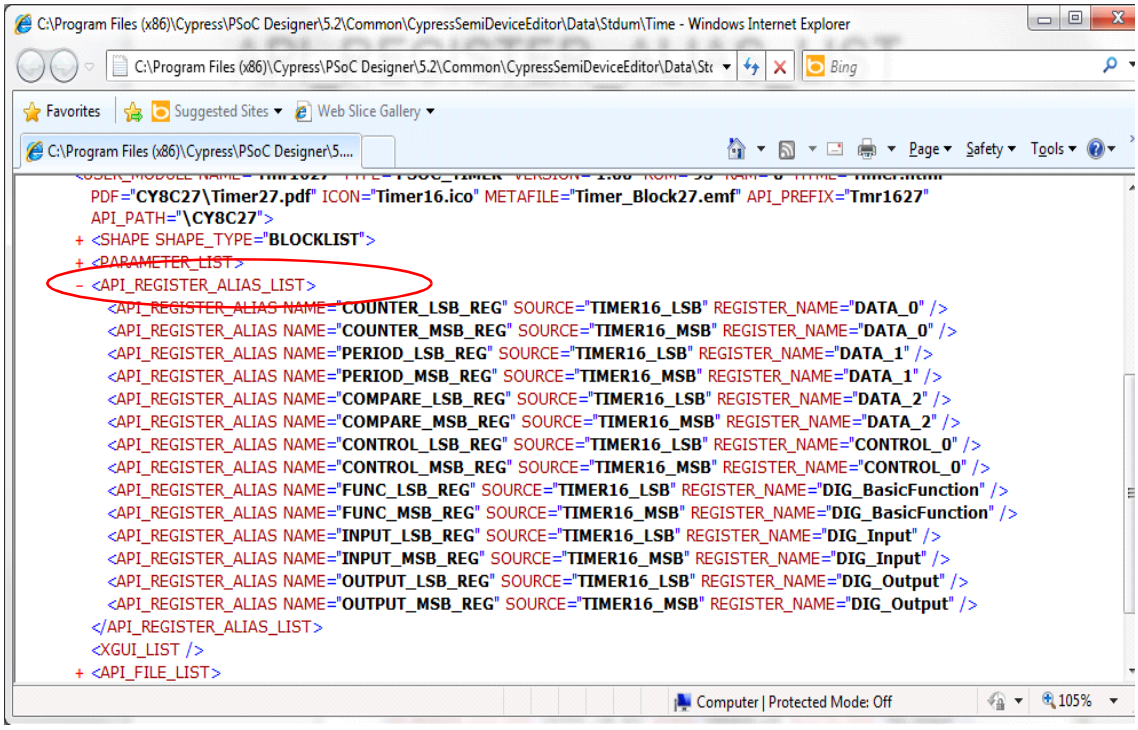
   <XGUI_LIST />

   <API_FILE_LIST>

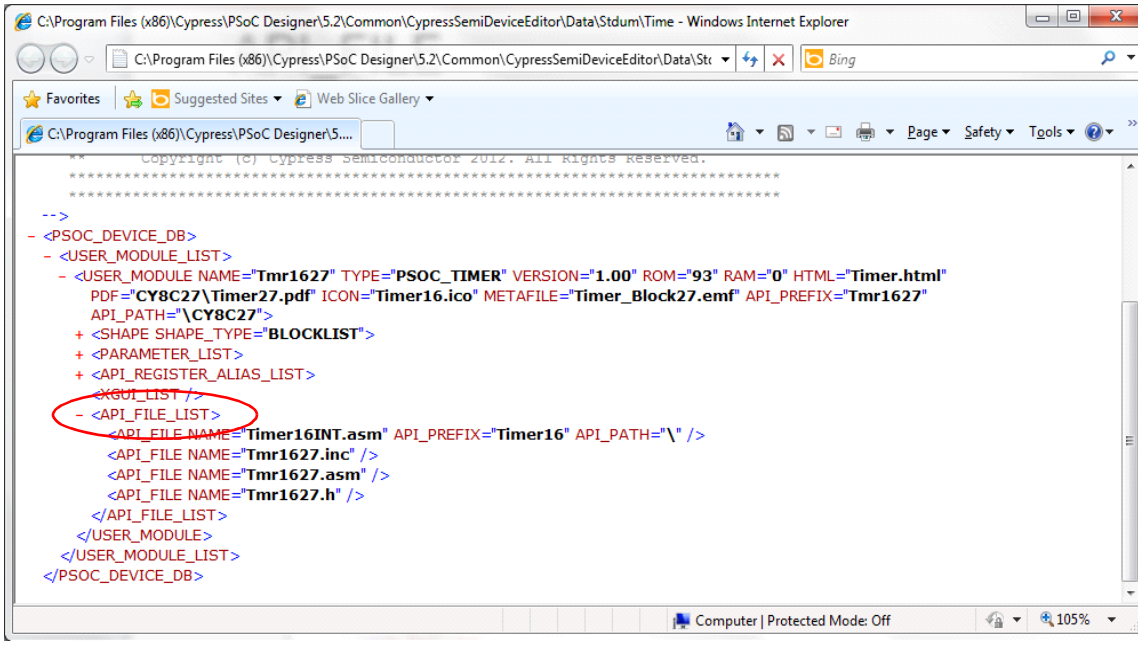
</USER_MODULE>
</USER_MODULE_LIST>
</PSOC_DEVICE_DB>
```

2.1.1 API Register List

- The API_REGISTER_ALIAS_LIST allows you to define register names.



- The API_FILE_LIST allows you to define the name and location of all API files. It is possible to add multiple assembly files and C routines.



2.1.2 Modifying the UM

To modify this UM, the LSB block must be set to be an interrupt source in the SHAPE section. The parameter used to select the interrupt type must be removed from the PARAMETER_LIST area. Bit values must be added to the SHAPE section to set the interrupt type to the following:

- Terminal Count for the MSB Block
- Capture for the LSB Block

2.2 Step 2: Add Interrupt

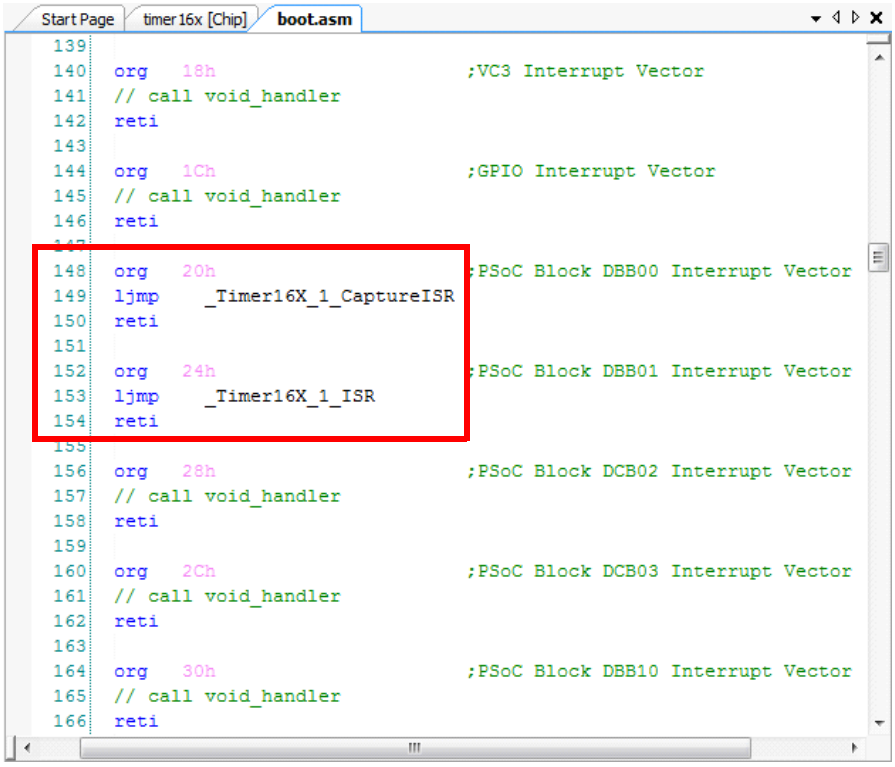
1. Open *Tmr1627.xml*.
2. Use the interrupt code from the TIMER16_MSB block to add an interrupt to the Timer16_LSB block. Name this new interrupt source _CaptureISR.
3. Close the file and reopen with Internet Explorer. This checking process is done frequently; you can name it ‘Close, Reopen with Explorer to Test’ (CRET).

Timer16X.xml | Timer16X.xml | Tmr1627.xml

```
<SHAPE SHAPE_TYPE = "BLOCKLIST">
    <BLOCK_LIST>
        <BLOCK              NAME = "TIMER16_LSB"
                            TYPE = "DIGITAL"
                  INTERRUPT_SOURCE = "_CaptureISR"
                    INTERRUPT_TYPE = "JUMP"
        >
            <REGISTER_LIST>
                <REGISTER NAME = "DIG_BasicFunction">
                    <BITFIELD_LIST>
                        <BITFIELD NAME = "Function"    VALUE = "Timer"  />
                        <BITFIELD NAME = "End"         VALUE = "NotEnd" />
                    </BITFIELD_LIST>
                </REGISTER>
            </REGISTER_LIST>
            <INPUT_LIST/>
        </BLOCK>
        <BLOCK              NAME = "TIMER16_MSB"
                            TYPE = "DIGITAL"
                  INTERRUPT_SOURCE = "_ISR"
                    INTERRUPT_TYPE = "JUMP"
        >
            <REGISTER_LIST>
                <REGISTER NAME = "DIG_BasicFunction">
                    <BITFIELD_LIST>
                        <BITFIELD NAME = "Function"    VALUE = "Timer" />
                        <BITFIELD NAME = "End"         VALUE = "IsEnd" />
                    </BITFIELD_LIST>
                </REGISTER>
            </REGISTER_LIST>
            <INPUT_LIST>
                <INPUT        SOURCE = "TIMER16_LSB"
                                TYPE = "BLOCK"
                       REGISTER_NAME = "DIG_Input"
                            BITFIELD = "DataSelect"
                />
            </INPUT_LIST>
        </BLOCK>
    </BLOCK_LIST>
</SHAPE>
```

4. Open a CY8C29666-24LFXI part-based project (or any other part from the Supported Devices List) with PSoC Designer.

5. Place Timer16X.
6. Generate the application.
7. Go to the application editor.
8. Open *boot.asm*.
9. Verify that the new interrupt vector has been added.



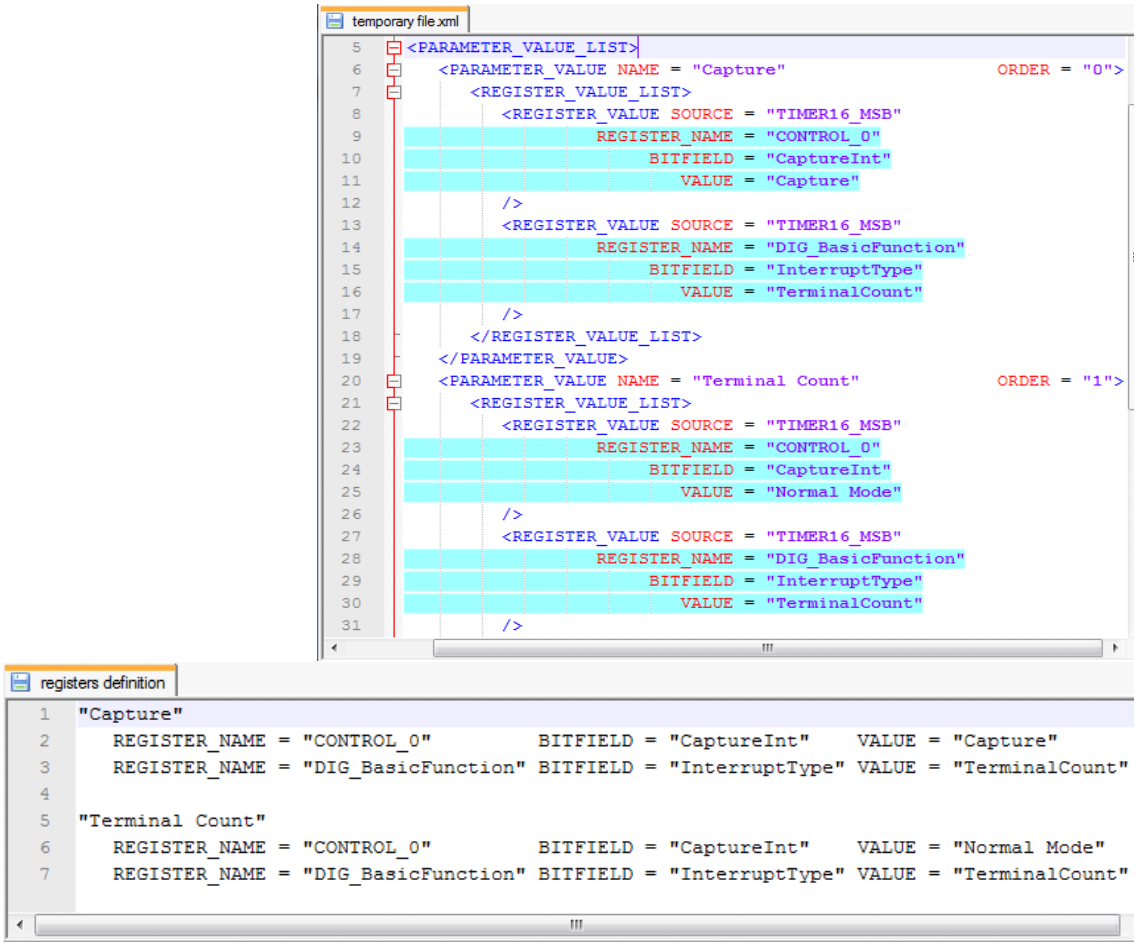
2.3 Step 3: Remove Interrupt Type Parameter

1. Close PSoC Designer.
2. Open Tmr1627.xml.
3. Remove the InterruptType parameter.
4. Save the removed portion in a temporary file.
5. For the ClockSync parameter, change the ORDER attribute from 8 to 7.
6. For the TC_Width parameter, change the ORDER attribute from 9 to 8.



2.4 Step 4: Add Interrupts to SHAPE

1. Open the temporary file where you pasted the InterruptType parameter.
2. The REGISTER_NAME, BITFIELD and VALUE attributes for each type of interrupt can be used to define the required registers.



3. Open Tmr1627.xml.
4. For TIMER16_LSB, add “InterruptType” bit field to the “DIG_BasicFunction” register. Set its value to “TerminalCount”.

5. Add the CONTROL_0 register. Add the “CaptureInt” bit field to it. Set its value to “Capture”.

Tmr1627.xml

```
<SHAPE SHAPE_TYPE = "BLOCKLIST">
  <BLOCK_LIST>
    <BLOCK              NAME = "TIMER16_LSB"
                        TYPE = "DIGITAL"
                        INTERRUPT_SOURCE = "_CaptureISR"
                        INTERRUPT_TYPE = "JUMP"
    >
      <REGISTER_LIST>
        <REGISTER NAME = "DIG_BasicFunction">
          <BITFIELD_LIST>
            <BITFIELD NAME = "Function"    VALUE = "Timer"  />
            <BITFIELD NAME = "End"         VALUE = "NotEnd" />
            <BITFIELD NAME = "InterruptType"         VALUE = "TerminalCount" /
          </BITFIELD_LIST>
        </REGISTER>
        <REGISTER NAME = "CONTROL_0">
          <BITFIELD_LIST>
            <BITFIELD NAME = "CaptureInt"    VALUE = "Capture"  />
          </BITFIELD_LIST>
        </REGISTER>
      </REGISTER_LIST>
      <INPUT_LIST/>
    </BLOCK>
    <BLOCK              NAME = "TIMER16_MSB"
                        TYPE = "DIGITAL"
                INTERRUPT_SOURCE = "_ISR"
                  INTERRUPT_TYPE = "JUMP"
    >
      <REGISTER_LIST>
        <REGISTER NAME = "DIG_BasicFunction">
          <BITFIELD_LIST>
            <BITFIELD NAME = "Function"    VALUE = "Timer" />
            <BITFIELD NAME = "End"         VALUE = "IsEnd" />
          </BITFIELD_LIST>
        </REGISTER>
      </REGISTER_LIST>
      <INPUT_LIST>
        <INPUT        SOURCE = "TIMER16_LSB"
                        TYPE = "BLOCK"
                REGISTER_NAME = "DIG_Input"
                     BITFIELD = "DataSelect"
        />
      </INPUT_LIST>
    </BLOCK>
  </BLOCK_LIST>
</SHAPE>
```

6. For TIMER16_MSB, add “InterruptType” bit field to the “DIG_BasicFunction” register. Set its value to “TerminalCount”.

7. Add the CONTROL_0 register. Add the “CaptureInt” bit field to it. Set its value to “Normal Mode”.

Tmr1627.xml

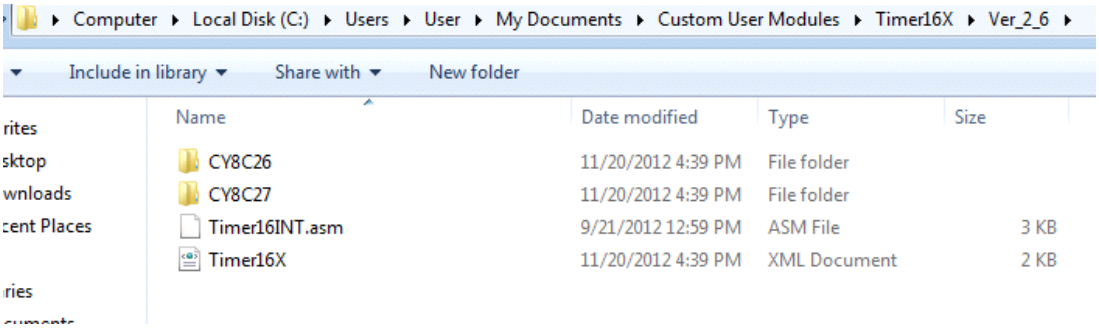
```
<SHAPE SHAPE_TYPE = "BLOCKLIST">
  <BLOCK_LIST>
    <BLOCK              NAME = "TIMER16_LSB"
                        TYPE = "DIGITAL"
                        INTERRUPT_SOURCE = "_CaptureISR"
                        INTERRUPT_TYPE = "JUMP"
    >
      <REGISTER_LIST>
        <REGISTER NAME = "DIG_BasicFunction">
          <BITFIELD_LIST>
            <BITFIELD NAME = "Function"    VALUE = "Timer"  />
            <BITFIELD NAME = "End"         VALUE = "NotEnd" />
            <BITFIELD NAME = "InterruptType" VALUE = "TerminalCount" />
          </BITFIELD_LIST>
        </REGISTER>
        <REGISTER NAME = "CONTROL_0">
          <BITFIELD_LIST>
            <BITFIELD NAME = "CaptureInt"    VALUE = "Capture"  />
          </BITFIELD_LIST>
        </REGISTER>
      </REGISTER_LIST>
      <INPUT_LIST/>
    </BLOCK>
    <BLOCK              NAME = "TIMER16_MSB"
                        TYPE = "DIGITAL"
                INTERRUPT_SOURCE = "_ISR"
                  INTERRUPT_TYPE = "JUMP"
    >
      <REGISTER_LIST>
        <REGISTER NAME = "DIG_BasicFunction">
          <BITFIELD_LIST>
            <BITFIELD NAME = "Function"    VALUE = "Timer" />
            <BITFIELD NAME = "End"         VALUE = "IsEnd" />
            <BITFIELD NAME = "InterruptType"         VALUE = "TerminalCount" />
          </BITFIELD_LIST>
        </REGISTER>
        <REGISTER NAME = "CONTROL_0">
          <BITFIELD_LIST>
            <BITFIELD NAME = "CaptureInt"    VALUE = "Normal Mode"  />
          </BITFIELD_LIST>
        </REGISTER>
      </REGISTER_LIST>
      <INPUT_LIST>
        <INPUT        SOURCE = "TIMER16_LSB"
                        TYPE = "BLOCK"
                REGISTER_NAME = "DIG_Input"
                    BITFIELD = "DataSelect"
        />
      </INPUT_LIST>
    </BLOCK>
  </BLOCK_LIST>
</SHAPE>
```

To modify firmware for this UM, you must do the following:

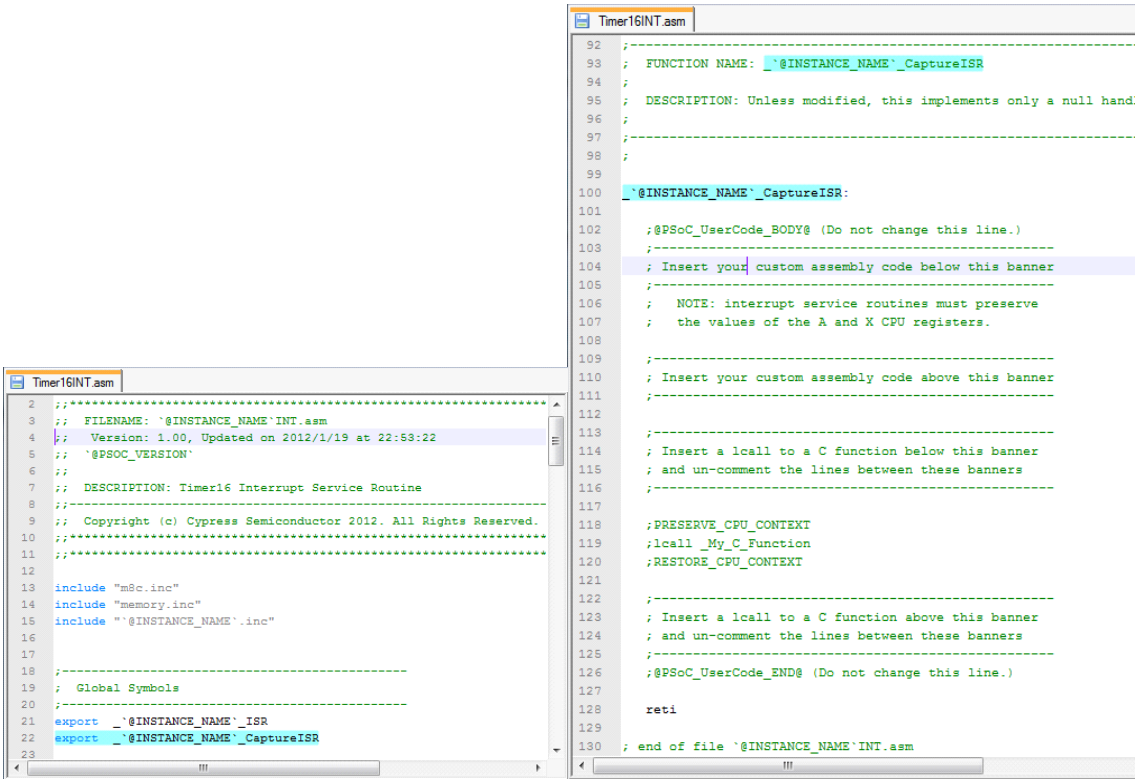
- Add a new interrupt handler for the new interrupt.
- Change the API to pass the intended interrupt to enable or disable type.
- Generate a new interrupt mask for this interrupt.
- For both C (.h) and assembly (.inc), change the placement file for the CY8C29000 parts so that both the interrupt masks are always in the same interrupt register.

2.5 Step 5: Add Interrupt Handler

1. Go to the Timer16X folder in the folder with custom UMs. Go to Timer16INT.asm file, which is located on the same level as the CY8C27 folder and open it.

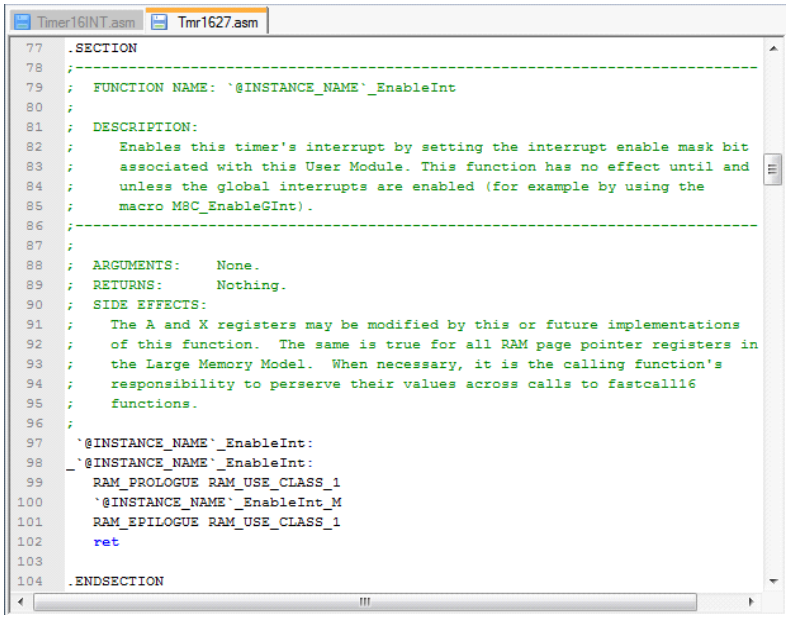


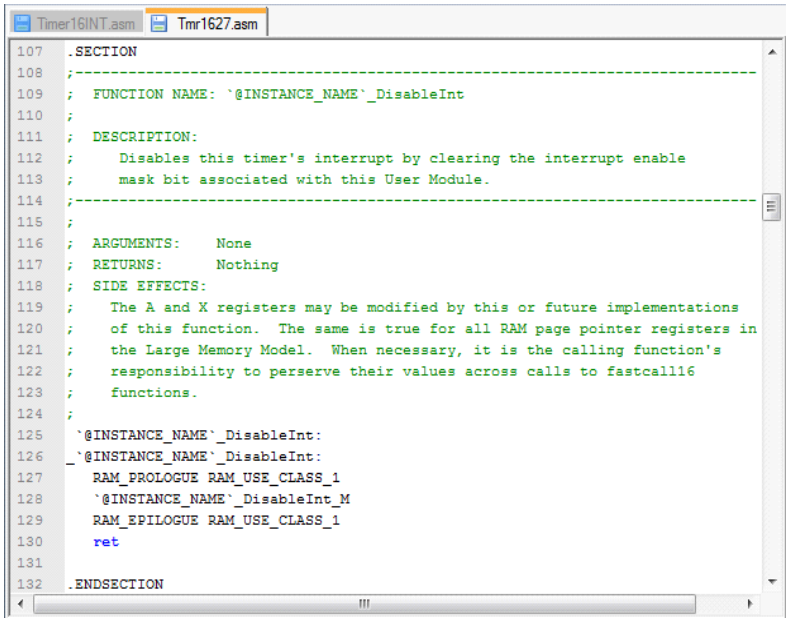
2. Create a duplicate _ISR function and rename it as _CaptureISR.
3. Add ‘export’ to the new label and save the file.



2.6 Step 6: Change APIs

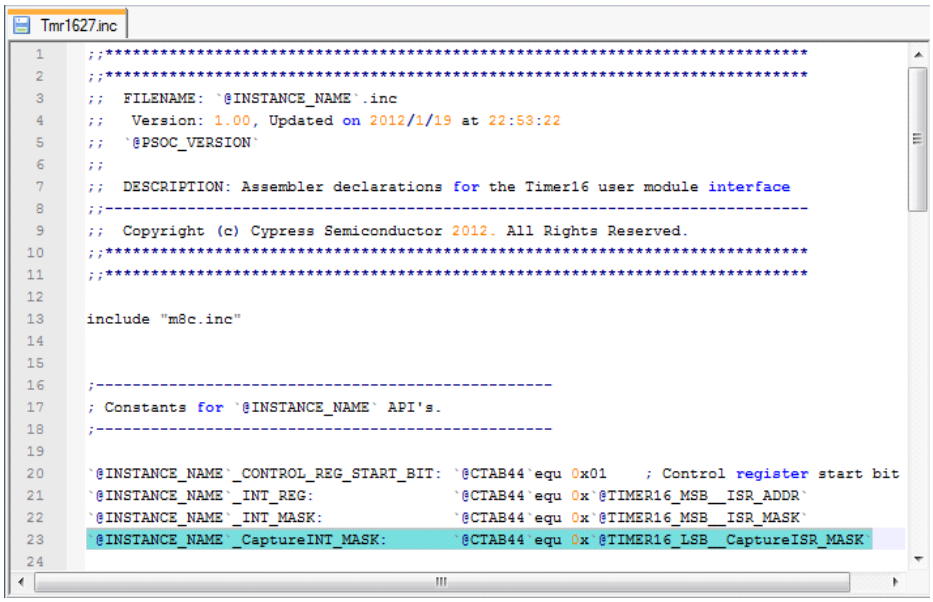
1. Open the CY8C27 folder and open Tmr1627.asm.
2. Both `@INSTANCE_NAME`_EnableInt and `@INSTANCE_NAME`_DisableInt APIs are macros. These macros are defined in Tmr1627.inc.
3. Although not needed for this particular example, this step is shown because many UMs you develop will require some assembly code alteration.





2.7 Step 7: Edit the .inc file

1. Open *Tmr1627.inc*.
2. Duplicate the _INT_ MASK declaration and change for new _CaptureINT_MASK declaration.
3. Change enable and disable interrupt macros to use ‘A’ as the enabling and disabling mask.




```
macro `@INSTANCE_NAME`_EnableInt_M
;M8C_EnableIntMask `@INSTANCE_NAME`_INT_REG, `@INSTANCE_NAME`_INT_MASK
mov  X, SP
push A
mov  A, reg[`@INSTANCE_NAME`_INT_REG]
or   A, [X]
mov  reg[`@INSTANCE_NAME`_INT_REG], A
pop  A
endm

macro `@INSTANCE_NAME`_DisableInt_M
;M8C_DisableIntMask `@INSTANCE_NAME`_INT_REG, `@INSTANCE_NAME`_INT_MASK
mov  X, SP
cpl  A
push A
mov  A, reg[`@INSTANCE_NAME`_INT_REG]
and  A, [X]
mov  reg[`@INSTANCE_NAME`_INT_REG], A
pop  A
endm
```

2.8 Step 8: Edit the .h File

1. Open *Tmr1627.h*.
2. Change the EnableInt and DisableInt function prototypes to require a BYTE argument.
3. Duplicate the _INT_ MASK #define and change the new _CaptureINT_MASK define.

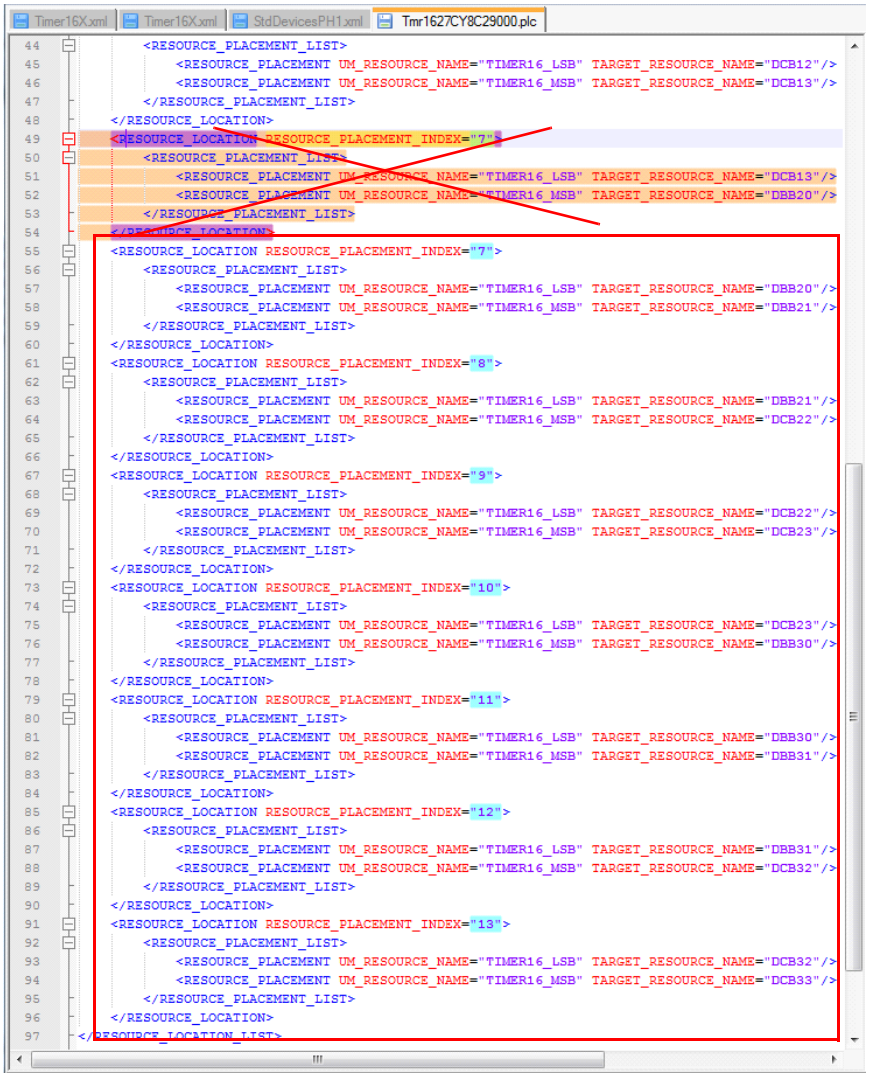
Tmr1627.h

```
37  //-------------------------------------------------
38  // Prototypes of the `@INSTANCE_NAME` API.
39  //-------------------------------------------------
40
41  //extern void `@INSTANCE_NAME`_EnableInt(void);                    // Proxy 1
42  extern void `@INSTANCE_NAME`_EnableInt(BYTE bMask);                // Proxy 1
43  //extern void `@INSTANCE_NAME`_DisableInt(void);                   // Proxy 1
44  extern void `@INSTANCE_NAME`_DisableInt(BYTE bMask);               // Proxy 1
45  extern void `@INSTANCE_NAME`_Start(void);                          // Proxy 1
46  extern void `@INSTANCE_NAME`_Stop(void);                           // Proxy 1
47  extern WORD `@INSTANCE_NAME`_wReadTimer(void);                     // Proxy 1
48  extern WORD `@INSTANCE_NAME`_wReadTimerSaveCV(void);               // Proxy 2
49  extern void `@INSTANCE_NAME`_WritePeriod(WORD wPeriod);            // Proxy 1
50  extern WORD `@INSTANCE_NAME`_wReadCompareValue(void);              // Proxy 1
51  extern void `@INSTANCE_NAME`_WriteCompareValue(WORD wCompareValue);// Proxy 1
52
53  // The following functions are deprecated.
54  // They may be omitted in future releases
55  //
56  extern WORD w`@INSTANCE_NAME`_ReadCompareValue(void);       // Deprecated
57  extern WORD w`@INSTANCE_NAME`_ReadTimerSaveCV(void);        // Deprecated
58  extern WORD w`@INSTANCE_NAME`_ReadCounter(void);            // Obsolete
59  extern WORD w`@INSTANCE_NAME`_ReadTimer(void);              // Deprecated
60  extern WORD w`@INSTANCE_NAME`_CaptureCounter(void);         // Obsolete
61
62
63  //-------------------------------------------------
64  // Constants for `@INSTANCE_NAME` API's.
65  //-------------------------------------------------
66
67  #define `@INSTANCE_NAME`_CONTROL_REG_START_BIT `@CTAB48`( 0x01 )
68  #define `@INSTANCE_NAME`_INT_REG_ADDR          `@CTAB48`( 0x`@TIMER16_MSB__ISR_ADDR` )
69  #define `@INSTANCE_NAME`_INT_MASK              `@CTAB48`( 0x`@TIMER16_MSB__ISR_MASK` )
70  #define `@INSTANCE_NAME`_CaptureINT_MASK       `@CTAB48`( 0x`@TIMER16_LSB__CaptureISR_MASK` )
71
```

2.9 Step 9: Edit Placement Files

Placement files list all legal placements for a particular chip family. Only the CY829000 family allows placement so that each block has a different interrupt register for the DCB13 DBB20 blocks.

Open *Tmr1627CY8C2900.plc*. Remove RESOURCE_PLACEMENT_INDEX="7" and renumber all the following indices (highlighted by a red box in the following graphic).



2.10 Verify All Modifications

1. Start a PSoC Designer project for the CY8C29666-24LFXI part.
2. Verify the correctness of all placement combinations, which are represented in the modified *Tmr1627CY8C2900.plc* placement file. The following is a list of the Timer16X legal placements:

#	TIMER16_LSB	TIMER16_MSB
1	DBB00	DBB01
2	DBB01	DCB02
3	DCB02	DCB03
4	DCB03	DBB10
5	DBB10	DBB11
6	DBB11	DCB12
7	DCB12	DCB13
8	DBB20	DBB21
9	DBB21	DCB22
10	DCB22	DCB23
11	DCB23	DBB30
12	DBB30	DBB31
13	DBB31	DCB32
14	DCB32	DCB33

3. Generate the application. Verify that the API compiles correctly.

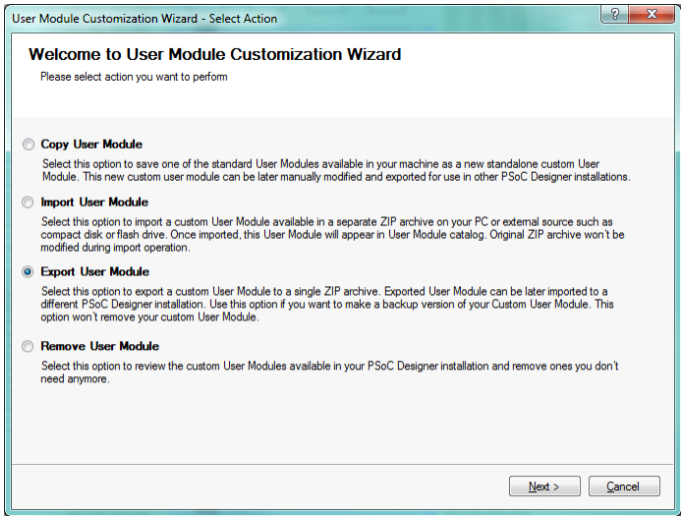
2.11 Distribution

After the new user module is code complete and tested, you can export Timer16X User Module to a single zip archive. The exported user module can be imported to a different PSoC Designer installation on another machine.

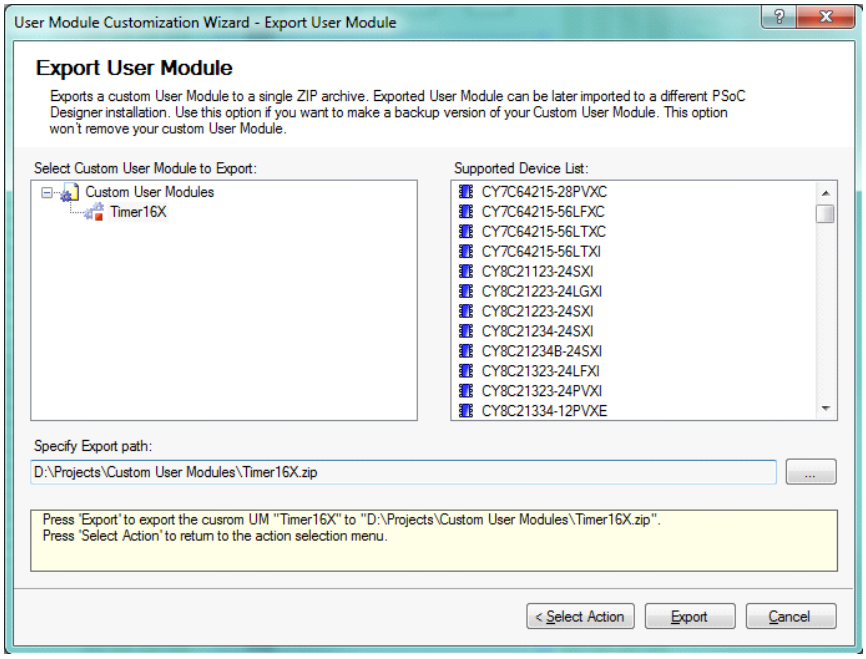
2.11.1 Exporting the User Module

1. Launch PSoC Designer.
2. Go to: **Tools > User Module Customization Wizard**.

3. Select **Export User Module** option and select **Next**.



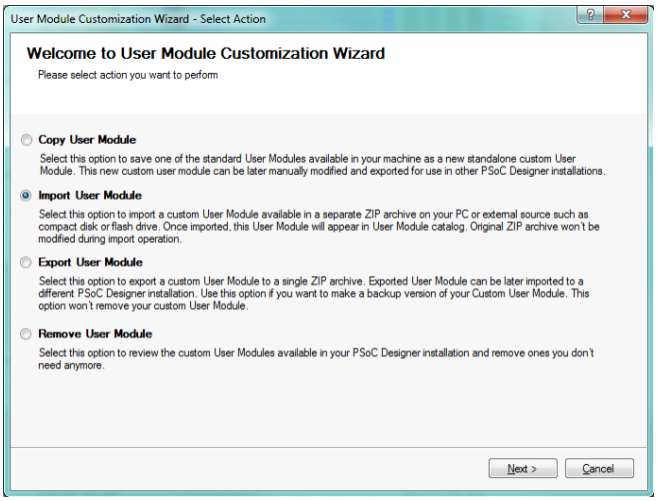
4. Select the **Timer16X** User Module.
5. Specify the path to which the user module will be exported.
6. Select **Export**.



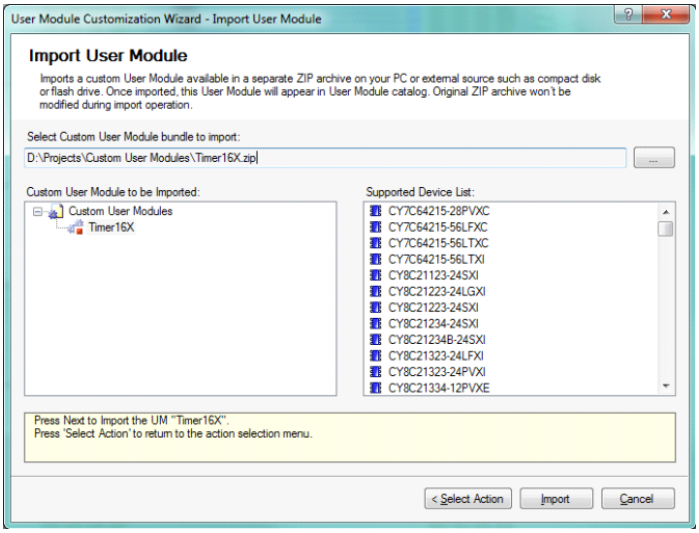
2.11.2 Importing the User Module

1. After the custom user module is exported, you can manually copy the Timer16X.zip archive to a specific location in another machine.
2. Launch PSoC Designer.
3. Go to **Tools > User Module Customization Wizard**.

4. Select the **Import User Module** option.



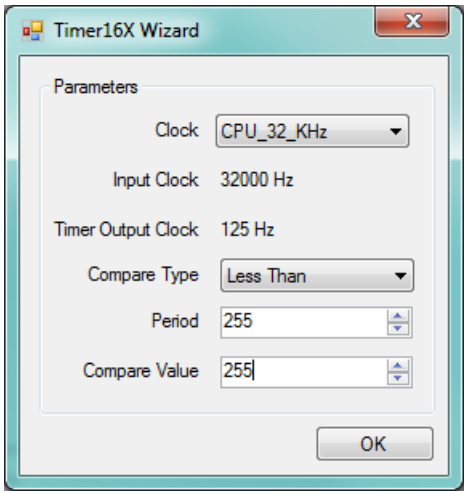
5. Select **Next**.
6. Select the Timer16X.zip archive.
7. Select **Import**.



3. Extending the Custom User Module

The objective of this project is to build an improved Timer16 User Module, using the existing user module. The upper digital block of the present Timer16 has a parameter selection to generate interrupt on Capture, Terminal Count, and Compare True. The goal of the extending is to add to our Timer16X UM a Graphical User Interface for configuring the parameters (Wizard) as well as pre-processing, code generation placement, un-placement and Design Rule Checker scripts as described below.

- Wizard which allows configuring UM parameters with the next view:



  '**Clock**', '**Period**' and '**Compare Value**', '**Compare Type**' are UM parameters we want to configure. Range of values for '**Clock**', '**Compare Type**' parameters as well as and minimum and maximum for '**Period**' and '**Compare Value**' to be loaded from UM architecture definition xml file. We will also display '**Input Clock**' and '**Timer Output Clock**' as read-only values.

  Timer16X Wizard determines and saves time of last Wizard modifying. This value is used in *.h file as a constant. 'OK' Wizard button stores the parameters in our project.

- The pre-processing javascript (JS) allows us to form value list of '**CompareOut**' UM parameter depending on the value selected for '**TerminalCountOut**' parameter and filter this way row connections that are already being used.
- Placement JS that works during UM placement sets value of '**TC_PulseWidth**' parameter to '**One-Half Clock**' if doubling the SysClk is enabled ('**SysClk*2 Disable**' global parameter is set to '**No**').
- Code generation JS which changes value of '**CompareValue**' parameter if it exceeds '**Period**' parameter value.
- Design Rule Check (DRC) JS displays a warning message in 'Output' window if calculated value of '**Timer Output Clock**' is greater than 8 MHz. '**Timer Output Clock**' parameter value depends on values of global parameters.

- Un-placement JS displays a message in separate dialog window to inform user that UM was deleted.

3.1 Add Wizard

Now we will add a Wizard to our UM. The Wizard configures '**Clock**', '**Period**' and '**Compare Value**', 'Compare Type' parameters. Minimum and maximum ranges of both '**Period**' and '**Compare Value**' parameters are loaded from UM architecture definition xml file. The Wizard has 'Input Clock' and 'Timer Output Clock' read-only parameters which are calculated corresponding to the next:

- 'Input Clock' determine timer clock depends on 'Clock' UM parameters and values of 'Power Setting [Vcc / SysClk freq]', 'VC1= SysClk/N', 'VC2= VC1/N', 'VC3 Source' and 'VC3 Divider' correspondingly, in Hz;
- 'Timer Output Clock' = 'Input Clock'/('Period'+1), in Hz

Also Wizard saves last time of its modifying. This value will be available in *.h file as a constant.

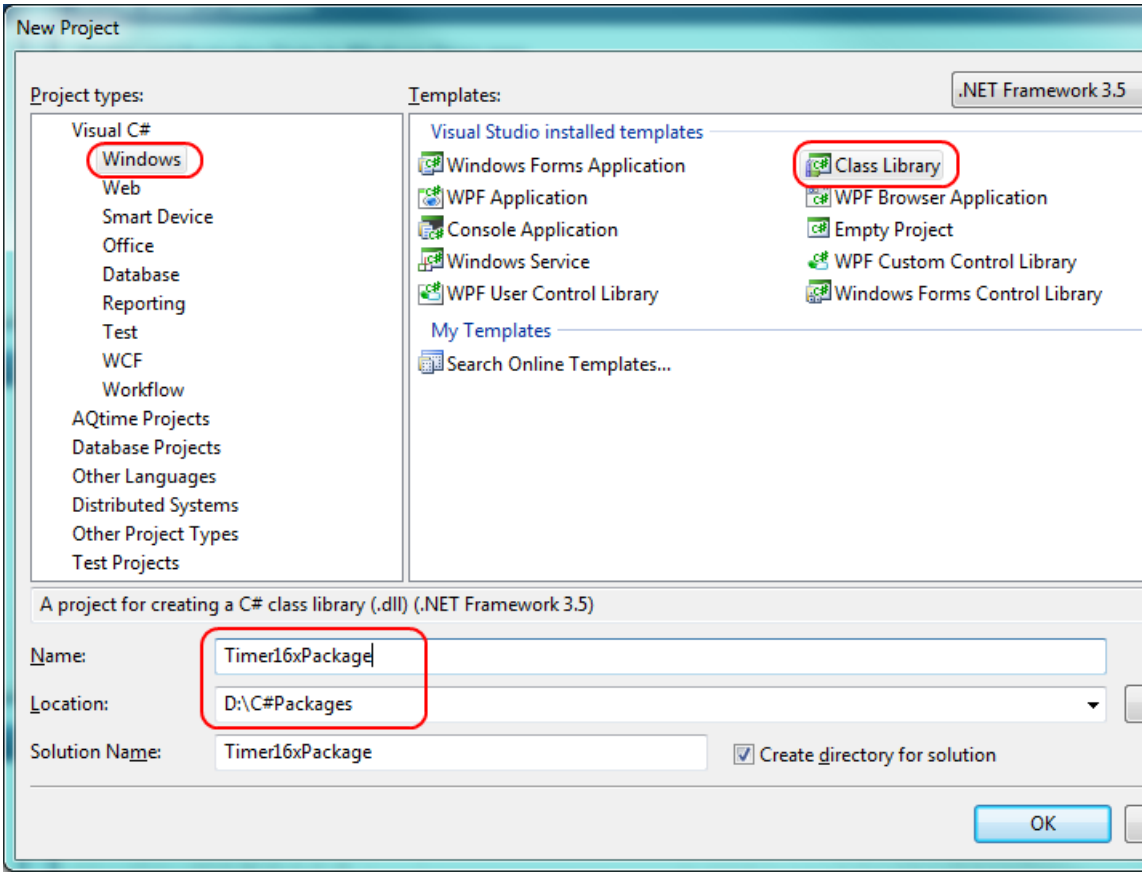
First, we will build a C# library that implements the Wizard GUI and behavior and then we will attach the Wizard to the UM, update UM xml description and *.h to manage constant.

3.1.1 Step 10: Create C# Package - General Rules

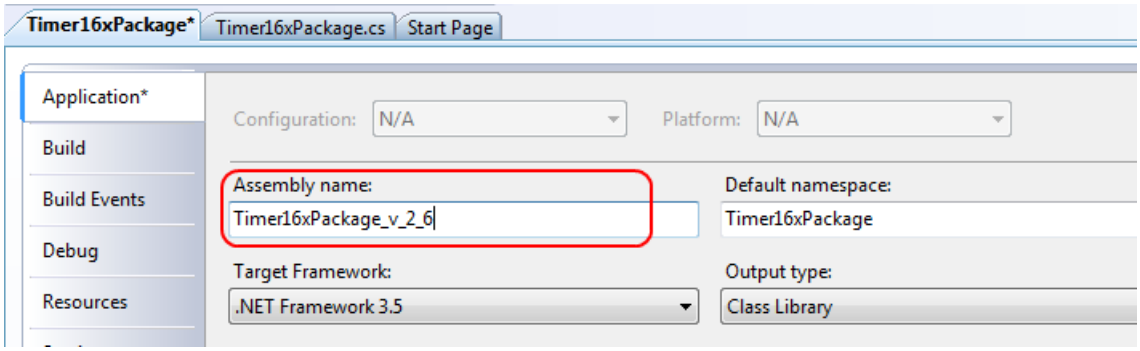
We will use Microsoft Visual Studio 2008 Service Pack 1 for Wizard creation. PSoC Designer is a .NET 2.0 applications so it is recommended to set 2.0 as the target framework for our library.

1. Launch Microsoft Visual Studio (VS).
2. Select **New > Project** option in **File** menu.
3. Select '**Windows**' project type and '**Class Library**' template in '**New Project**' dialog.
4. Set '**Timer16xPackage**' name in '**New Project**' dialog.

5. Set a location in '**New Project**' dialog to store created C# package.

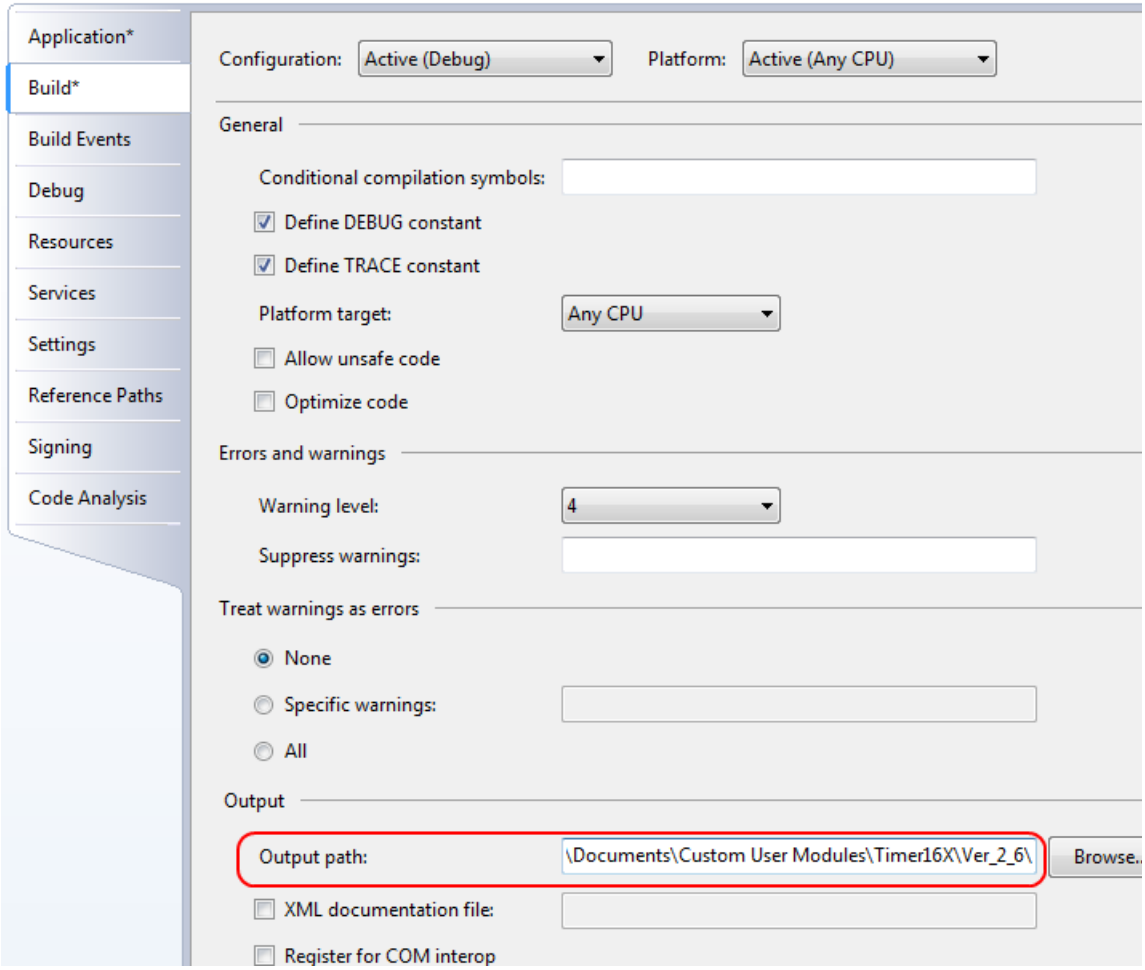


6. Rename '**Class1.cs**' in Solution Explorer to '**Timer16xPackage.cs**' (if Solution Explorer is not available by default, go to **View** menu and select '**Solution Explorer**' item).
7. Right-click on **Timer16xPackage** in Solution Explorer and select **Properties** in available menu.
8. Set '**Timer16xPackage_v_2_6**' assembly name in '**Application**' tab:



9. Set path to **Ver_2_6** folder of **Timer16X** custom UM in '**Build**' tab to pint to the folder with custom user module folder as described below. In our case the path to the Timer16X Custom UM is "Doc-

uments\ Custom User Modules\". Whenever you build your Wizard C# application, the DLL library file will appear updated in UM folder:

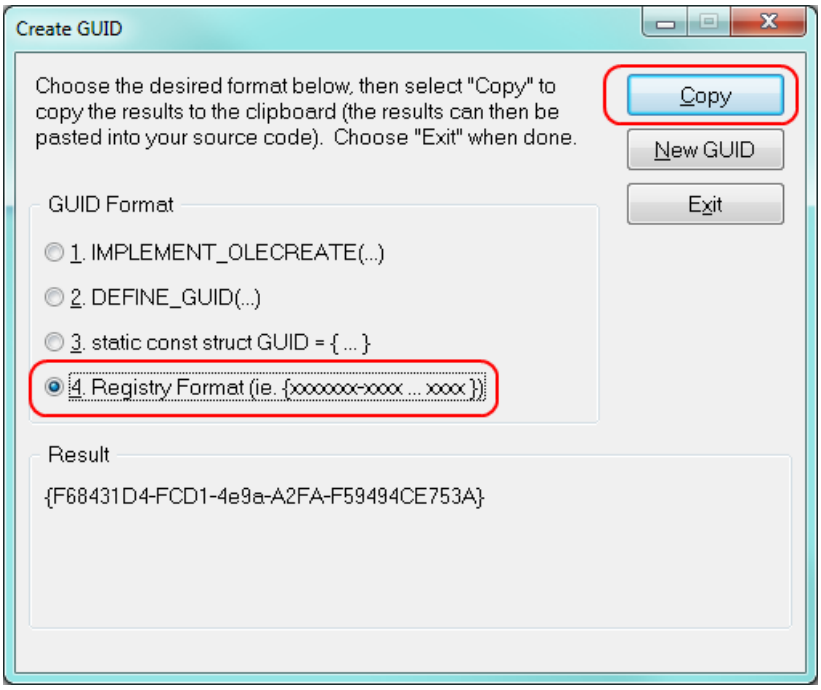


10. Save All.
11. Open *Timer16xPackage.cs* file in VS.
12. Set the next line after namespace Timer16xPackage:

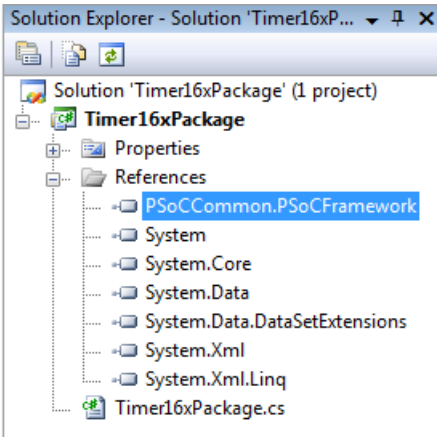
```
[Guid("XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"),
ClassInterface(ClassInterfaceType.None)]
```

13. Select **Tools > Create GUID**.

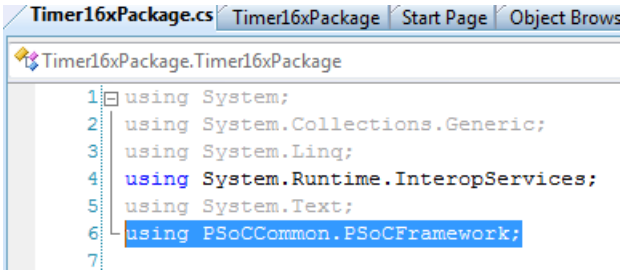
14. Select '**Registry Format**' in '**Create GUID**' dialog and select **Copy** button:



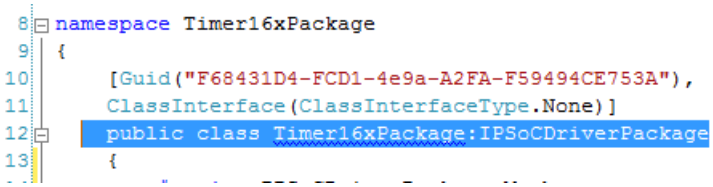
15. Exit from '**Create GUID**' dialog.
16. Replace `XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX` with `{F68431D4-FCD1-4e9a-A2FA-F59494CE753A}` in **Guid** string.
17. Remove '{' and '}' brackets in copied string.
18. Right-click on '**References**' in Solution Explorer and select '**Add Reference…**' item.
19. Go to '**Browse**' tab in '**Add Reference**' dialog.
20. Select '**PSoC Designer 5**' folder which is located in **PSoC Designer 5.4** installation folder.
21. Select '**PSoCCommon.PSoCFramework.dll**' file and close '**Add Reference**' dialog by pressing **OK**.
22. Check if **PSoCCommon.PSoCFramework** item is available under '**References**' in Solution Explorer:



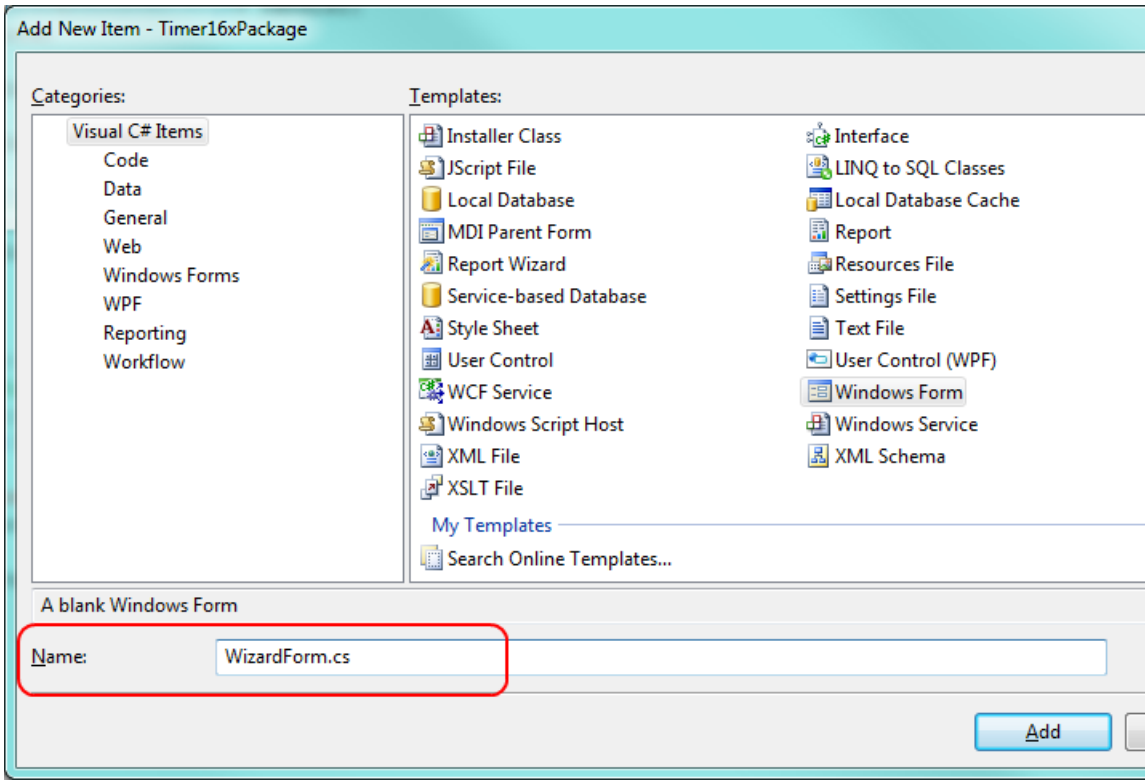
23. Now open *Timer16xPackage.cs* file and add **PSoCCommon.PsoCFramework** name space to *Timer16xPackage.cs* file:



24. Declare **IPSoCDriverPackage** interface in **Timer16xPackage** class definition:



25. Right-click on **IPSoCDriverPackage** interface and select **Interface > Implement Interface**.
26. Declare **_instanceName**, **_packageType**, **_cmxID**, **_lock** and **_installPath** private fields in **Timer16xPackage** class. See example of declaration in Appendix A on page 55.
27. Right-click on **Timer16xPackage** in Solution Explorer and select **Add > Windows Form…**
28. Type '**WizardForm**' name in '**Add New Item - Timer16xPackage**' dialog.
29. Press '**Add**' button in '**Add New Item - Timer16xPackage**' dialog:



30. Implement **DoWizard** method in **Timer16xPackage** class. See example of declaration in Appendix A on page 55.
31. Temporarily replace '`WizardForm wizardForm = new WizardForm(_cmxID, _instanceName, sOverlayName);`' string with: `WizardForm wizardForm = new WizardForm();` in **Timer16xPackage** class.
32. Implement **InstallPath**, **SetCmxID**, **SetInstanceName**, **SetPackageType** and **Start** methods in **Timer16xPackage** class. See example of declaration in Appendix A on page 55.
33. Remove '`throw new NotImplementedException();`' string in **OnMessage** and **OnQuit** methods.
34. Implement **OnPropertiesUpdated** method in **Timer16xPackage** class. See example of declaration in Appendix A on page 55.
35. Select **Build > Build Solution**. Check and correct any errors/warnings.

3.1.2 Step 11: Attach *.dll to UM

1. Go to the **Timer16X** folder in the folder with custom UMs.
2. Open the **CY8C27** UM sub-folder and open the *Tmr1627.xml* file
3. Replace **</XGUI_LIST>** element with the following text:

```
<XGUI_LIST>
<XGUI NAME="TIMER16X_WIZARD" DISPLAY_NAME="TIMER16X Wizard..."
SRC="..\Timer16xPackage_v_2_6.dll"/>
</XGUI_LIST>
```

Value of NAME attribute is free. The value of DISPLAY_NAME specifies text to be displayed when you right-click on UM in chip view The value of SRC attribute specifies relative path to the library that implements the wizard.

Finally, the UM xml will look like this:

Tmr1627.xml

```
115         <PARAMETER  NAME = "Period"           TYPE = "BLOCK"
116                    ORDER = "4"          VALUE_TYPE = "INT"
117                                             MIN = "0x0000"
118                                             MAX = "0xffff"
119                                             HIDDEN = "TRUE"
120         >
135         <PARAMETER  NAME = "CompareValue"     TYPE = "BLOCK"
136                    ORDER = "5"          VALUE_TYPE = "INT"
137                                             MIN = "0x0000"
138                                             MAX = "0xffff"
139                                             HIDDEN = "TRUE"
140         >
155         <PARAMETER  NAME = "CompareType"
156                    ORDER = "6"
157                     TYPE = "BLOCK"
158                     HIDDEN = "TRUE"
159         >
191         <PARAMETER  NAME = "ClockSync"
192                    ORDER = "7"
193                     TYPE = "BLOCK"
194         >
254         <PARAMETER  NAME = "TC_PulseWidth"    TYPE = "BLOCK"
255                    ORDER = "8"              SOURCE = "TIMER16_MSB"
256                                       REGISTER_NAME = "CONTROL_0"
257                                           BITFIELD = "TCPulseWide"
258                                              VALUE = "Full Clock"
259         />
260     </PARAMETER_LIST>
261
262     <API_REGISTER_ALIAS_LIST>
320
321       <XGUI_LIST>
322           <XGUI NAME="TIMER16X_WIZARD" DISPLAY_NAME="TIMER16X Wizard..." SRC="..\Time
323       </XGUI_LIST>
324
325     <API_FILE_LIST>
326         <API_FILE NAME = "Timer16INT.asm"
327             API_PREFIX = "Timer16"
328               API_PATH = "\"
329         />
```

3.2 Verify Wizard Availability in Custom UM

1. Launch PSoC Designer.
2. Create chip-level project based on CY8C29666-24LFXI part number (or on any other part from the Supported Devices List of our Timer16X UM) with PSoC Designer.
3. Place **Timer16X** UM.
4. Check if '**TIMER16X Wizard…**' appears in right-click menu of placed UM. If it does, press it. If there is no '**TIMER16X Wizard…**' present in right-click menu – close PD, go back and check the previous steps.
5. Observe empty wizard form appears upon press '**TIMER16X Wizard…**'

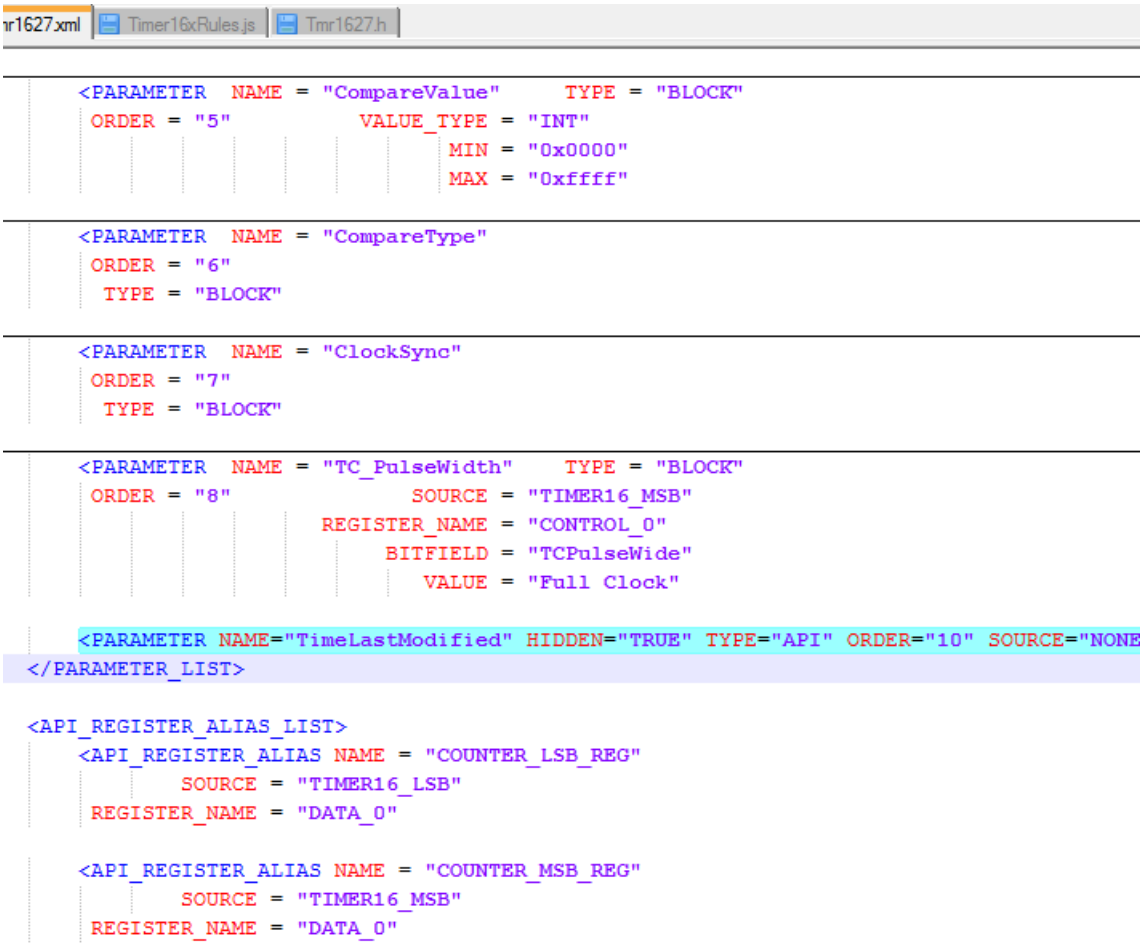
Now you are ready to implement the Wizard functionality. Whenever you add any code to the Wizard C# library, you can build and check to see if it works as expected. Remember to close PSoC Designer instances, so that the library can be updated properly in our UM folder.

3.2.1 Step 12: Update both xml and h files to manage time constant

1. Go to the **Timer16X** folder in the folder with custom UMs.
2. Open the **CY8C27** UM sub-folder and open the *Tmr1627.xml* file
3. Add the following after '**TC_PulseWidth**' parameter description:

```
<PARAMETER NAME="TimeLastModified" HIDDEN="TRUE" TYPE="API" ORDER="10"
SOURCE="NONE" VALUE_TYPE="STRING" VALUE=""/>
```

4. CRET.



5. Open *Tmr1627.h* file in **CY8C27** UM sub-folder.
6. Add the next constant declaration after **COMPARE_VALUE**:

```
#define `@INSTANCE_NAME`_TIME_LAST_MODIFIED `@CTAB48` "`@TimeLastModified`"
//Last time Wizard was modified at this time
```

EventHandlers.js | Tmr1627.xml | Timer16xRules.js | Tmr1627.h

```
55  //
56  extern WORD w`@INSTANCE_NAME`_ReadCompareValue(void);       // Deprecated
57  extern WORD w`@INSTANCE_NAME`_ReadTimerSaveCV(void);        // Deprecated
58  extern WORD w`@INSTANCE_NAME`_ReadCounter(void);            // Obsolete
59  extern WORD w`@INSTANCE_NAME`_ReadTimer(void);              // Deprecated
60  extern WORD w`@INSTANCE_NAME`_CaptureCounter(void);         // Obsolete
61
62
63  //-------------------------------------------------
64  // Constants for `@INSTANCE_NAME` API's.
65  //-------------------------------------------------
66
67  #define `@INSTANCE_NAME`_CONTROL_REG_START_BIT `@CTAB48`( 0x01 )
68  #define `@INSTANCE_NAME`_INT_REG_ADDR          `@CTAB48`( 0x`@TIMER16_MSB__ISR_ADDR
69  #define `@INSTANCE_NAME`_INT_MASK              `@CTAB48`( 0x`@TIMER16_MSB__ISR_MASK
70  #define `@INSTANCE_NAME`_CaptureINT_MASK       `@CTAB48`( 0x`@TIMER16_LSB__CaptureI
71
72
73  //-------------------------------------------------
74  // Constants for `@INSTANCE_NAME` user defined values
75  //-------------------------------------------------
76
77  #define `@INSTANCE_NAME`_PERIOD        `@CTAB48`( 0x`@Period` )
78  #define `@INSTANCE_NAME`_COMPARE_VALUE `@CTAB48`( 0x`@CompareValue` )
79  #define `@INSTANCE_NAME`_TIME_LAST_MODIFIED `@CTAB48` "`@TimeLastModified`"    //La
80
81  //-------------------------------------------------
82  // Register Addresses for `@INSTANCE_NAME`
83  //-------------------------------------------------
84
85  `@COUNTER_LSB_REG_IOH`    `@CTAB60`//Count register LSB
86  `@COUNTER_LSB_REG_H`
87  `@COUNTER_MSB_REG_IOH`    `@CTAB60`//Count register MSB
88  `@COUNTER_MSB_REG_H`
89  `@PERIOD_LSB_REG_IOH`     `@CTAB60`//Period register LSB
90  `@PERIOD_LSB_REG_H`
91  `@PERIOD_MSB_REG_IOH`     `@CTAB60`//Period register MSB
92  `@PERIOD_MSB_REG_H`
93  `@COMPARE_LSB_REG_IOH`    `@CTAB60`//Compare register LSB
94  `@COMPARE_LSB_REG_H`
```

3.2.2 Step 13: Implement Required Functionality in Wizard

Now we have to write the C# code to implement the desired Wizard functionality. The complexity of that code depends on required features which should be implemented in Wizard. The target of this guide is to provide the example how to use the interfaces available for the Wizards in PSoC Designer. The example of code which implements Timer16X Wizard is represented in Appendix A on page 55.

There are list of common rules and set of PSoC Designer commands which are used in UM Wizard implementation.

The following rules should be followed by Wizard author to achieve the UM Wizard requirements:

1. A current value, available value list, minimum and maximum range of UM parameters should be read from project DOM (Document Object Model) using WizardForm_Load . This must be done during Wizard form loading. See WizardForm_Load method in Appendix A on page 55.
2. The parameters values should be stored in project DOM during Wizard form closing if OK button is pressed. See button_OK_Click method in Appendix A on page 55. The ‘Cancel’ button handler should do nothing else than just close the form.
3. All controls on Wizard form should be initialized during Wizard form load. See code example of InitializeComboBox and InitializeNumericUpDown methods in Appendix A on page 55.
4. There is no need to store individual parameter value immediately in Wizard code – you can save all parameters in the project DOM later, all at once;
5. Handle PropertiesUpdated event after Wizard form closing if **OK** button is pressed to reflect UM parameters values on interconnect view and in Parameter Window.

The following diagram represents Wizard functionality in general:

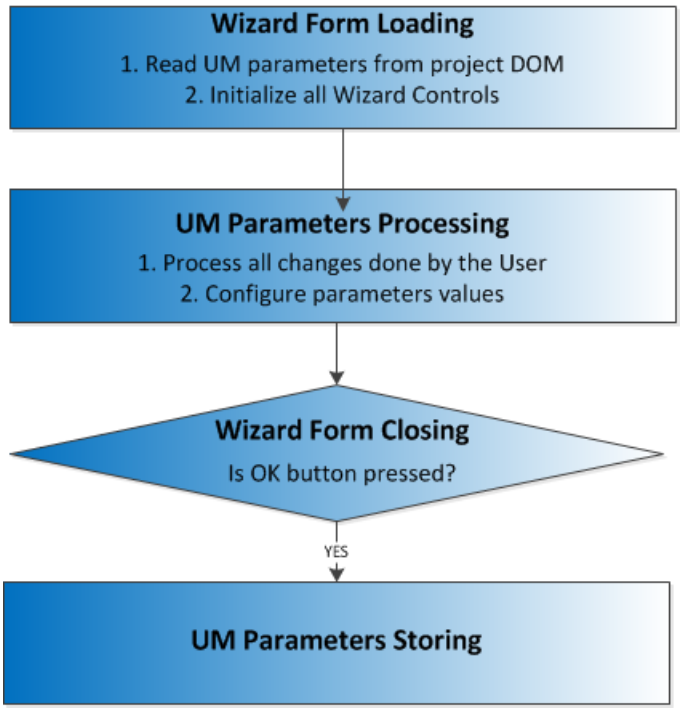


Table 3-1 on page 39 represents commands which can be used in UM Wizard. All these commands are used as second argument of CMXEngineAccessor.GetData method. There are details of CMX-EngineAccessor.GetData method:

1. Namespace: PSoCCommon.PsoCFramework
2. Parameters:
   a. Parameter #1: project ID;
   b. Parameter #2: command;
   c. Parameter#3,#4 and #5 depend on command;
3. Return object: depend on command, but always in string format.

The following example demonstrates how you can use CMXEngineAccessor.GetData method in UM Wizard:

```
string sOverlayName = CMXEngineAccessor.GetData(_cmxID,
"GET_OVERLAY_NAME_FROM_INSTANCE", _instanceName, "", "");
```

Table 3-1. CMXEngineAccessor.GetData Commands

#	Command	Parameters	Comments
1	GET_OVERLAY_NAME_FROM_INSTANCE	**Parameter#3 -** UM instance name **Parameter#4 -** empty string **Parameter#5 -** empty string **Return -** project name	Returns project name
2	GET_USERMODULE_PARAMETERS	**Parameter#3 -** project name **Parameter#4 -** UM instance name **Parameter#5 -** empty string **Return -** UM parameter DOM	Returns UM parameter DOM which contains current values, value list, maximum and minimum ranges
3	SET_USERMODULE_PARAM_NO_OVERLAY	**Parameter#3 -** UM instance name **Parameter#4 -** UM parameter name **Parameter#5 -** UM parameter value	Sets UM parameter value in project DOM
4	GET_GLOBAL_RESOURCES	**Parameter#3 -** project name **Parameter#4 -** UM instance name **Parameter#5 -** empty string **Return -** global parameters DOM	Returns global parameters DOM which contains current value of parameters
5	GET_DEVICE_DISPLAY_NAME	**Parameter#3 -** empty string **Parameter#4 -** empty string **Parameter#5 -** empty string **Return -** name of part number	Returns name of PSoC Device part number
6	GET_USERMMODULE_INFO_FROM_INSTANCENAME	**Parameter#3 -** UM instance name **Parameter#4 -** empty string **Parameter#5 -** empty string **Return** - Multi UM DOM	Returns DOM of Multi UM which includes path to custom UM, UM name, Multi UM name and etc..
7	GET_CURRENT_PIN_INFO	**Parameter#3 -** project name **Parameter#4 -** empty string **Parameter#5 -** empty string **Return** - pins DOM	Returns DOM of all pins which includes pin name, pin label and pin custom name of each of them.
8	GET_UM_ROOT_PATH	**Parameter#3 -** UM instance name **Parameter#4 -** empty string **Parameter#5 -** empty string **Return** - path to UM	Returns path to folder inside UM folder where UM package is located

Table 3-1. CMXEngineAccessor.GetData Commands *(continued)*

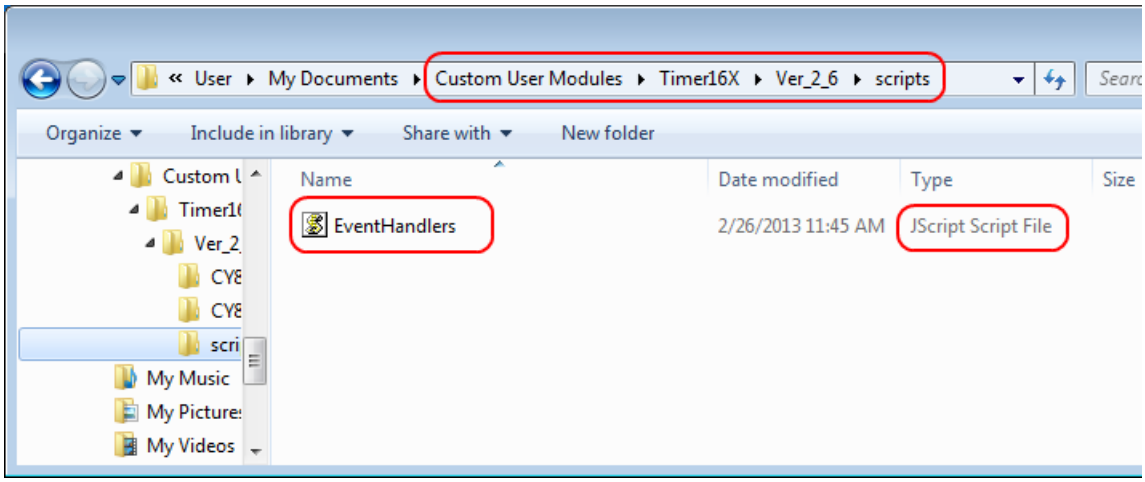
#	Command	Parameters	Comments
9	GET_UM_PATH	**Parameter#3 -** UM instance name **Parameter#4 -** empty string **Parameter#5 -** empty string **Return** - path to UM	Returns path to UM xml file which describes current UM
10	SET_RESERVED_RESOURCE	**Parameter#3 -** project name **Parameter#4 -** name of reserved resource **Parameter#5 -** value of reserved resource	Sets value of reserved resource (like AnalogMuxBus, Decimator)
11	GET_RESERVED_RESOURCE_CURRENT_VAL	**Parameter#3 -** project name **Parameter#4 -** name of reserved resource **Parameter#5 -** empty string **Return** - current value	Gets current value of reserved resource (like AnalogMuxBus, Decimator, Row_0_Output_1)
12	GET_PIN_INFO_VECTOR	**Parameter#3 -** project name **Parameter#4 -** empty string **Parameter#5 -** empty string **Return** - pins DOM	Returns DOM of all pins which includes type, interrupt and initial value list and current value.
13	CMX_GET_MEMORY_CAPACITY	**Parameter#3 -** project name **Parameter#4 -** empty string **Parameter#5 -** empty string **Return** - memory DOM	Returns DOM which includes RAM/ROM settings.
14	GET_DEVICE_NAME	**Parameter#3 -** project name **Parameter#4 -** UM instance name **Parameter#5 -** empty string **Return** - ID of part number	Returns internal ID of PSoC Device part number

3.3 Add Pre-Processing Javascript

An adding of pre-processing JS to UM requires a modifying of UM xml file and creating JS code with required functionality.

3.3.1 Step 14: Create JS file

1. Go to the **Timer16X** folder in the folder with custom UMs.
2. Create scripts folder on the **CY8C27** folder level.
3. Go to **scripts** folder.
4. Create file and name it *EventHandlers.js*.



3.3.2 Step 15: Attach Pre-Processing Javascript to UM

1. Go to the **Timer16X** folder in the folder with custom UMs.
2. Open the **CY8C27** UM sub-folder and open the *Tmr1627.xml* file
3. Go to '**CompareOut**' parameter description.
4. Add the pre-processing JS call:

```
<XGUI_PARAM>
    <PRE_PROCESS_LIST>
        <PRE_PROCESS SRC="..\scripts\EventHandlers.js" CALL="CompareOutSelect"
ORDER="0"/>
    </PRE_PROCESS_LIST>
</XGUI_PARAM>
```

The value of **SRC** attribute specifies name of JS file which implements pre-processing and relative path to it. The value of **CALL** attribute determines name of function which is called before value of '**CompareOut**' parameter gets changed. The value of **ORDER** attribute specifies order of pre-processing calls for current parameter.



Note: A post-processing JS to custom UM (see Step #14-15) can be added the same way. In this case you should add PRE_PROCESS_LIST element to UM parameter in Step #15.

3.3.3 Step 16: Implement Required Functionality in Pre-/Post-Processing JS

The pre-processing JS forms value list of '**CompareOut**' UM parameter depending on the value selected for '**TerminalCountOut**' parameter and filter this way row connections that are already being used.

The example of code which implements the filtering for '**CompareOut**' parameter is shown in Appendix B on page 66.

Pay attention that post- and pre-processing JS function has one argument which contains DOM of processed UM and its parameters. This DOM contains the following attributes:

- UM_INST_NAME - UM instance name;
- UM_NAME - UM name;
- PARAM_NAME - name of processed UM parameter;
- PARAM_VALUE - value of processed UM parameter.

There is a set of PSoC Designer methods and objects that can be used in UM pre-/post-processing JS. Table 3-2 represents the objects used in custom pre-/post-processing JS.

Table 3-2. Pre-/Post-Processing JS Global Objects

Object	Comments
PS	General Purpose Automation Interface. This object dispatches pointer to `IPSoCOttoHelper` interface and supports the following methods:
	`object GetXMLDOM();`
	`string ReadTextFile(string path);`
	`string WriteTextFile(string path, string text);`
	`object GetDeviceEditor();`
	`object GetProjectManager();`
	`object GetDebugTarget();`
	`object GetBuildManager();`
	`short FileExists(string path);`
	`long CreateDirectory(string path);`
	`long DeleteFile(string path);`
	`string ReadFile(string path);`
	`long WriteFile(string path, string data);`
	`long GetLastError();`
	`string FormatMessage(long id);`
	`long CopyFile(string from, string to, short failIfExists);`
	`long GetMainWnd();`
	`long FindWindow(long parentWnd, long childWnd, string className, string windowName);`
	`long SendMessage(long wnd, long msg, string wparamType, string wparam, string lparamType, string lparam, long post);`
	`long SendMessageToAllDevEditViews(long msg, string wparamType, string wparam, string lparamType, string lparam, long post);`
	`long RegSetValueEx(string key, string subKey, string valueName, string type, string data);`
	`long RegQueryValueEx(string key, string subKey, string valueName, string type, ref string data);`
	`long ShowMessageBox(string msg);`
	`long StartProcess(string cmdLine);`
	`string GetShortPathFromLong(string path);`
	`long RegQueryValueEx(string key, string subKey, string valueName, string type, ref string data);`
otto	Device Editor Document Interface. This object dispatches pointer to `IDeviceEditorDocument` interface and supports the following methods:
	`string GetCurUMParamValue(string param);`
	`int SetCurUMParamValue(string param, string val);`
	`string GetCurUMParamValueRange(string param);`
	`int SetCurUMParamValueRange(string param, string valRange);`
	`string GetGlobalResourceValue(string param);`
	`int SetGlobalResourceValue(string param, string val);`
	`string GetGlobalResourceValueRange(string param);`
	`int SetGlobalResourceValueRange(string param, string valRange);`
	`string CallMethod(string type, string callBack, string paramList, string wantReturn);`

The most usable method of PS interface is GetDeviceEditor() which returns Device Editor Automation Interface which has set of useful methods:

```
var devEditor  = PS.GetDeviceEditor();
```

Table 3-3. List of Device Editor Methods

Object	Comments
PS.GetDeviceEditor();	Device Editor Automation Interface. This object dispatches pointer to IPSoCDeviceEditorOtto interface and supports the following methods:
	`int GenerateSrc();`
	`int SelectUserModule(string szName, string szInstanceName);`
	`int UnSelectUserModule(string szName);`
	`int PlaceUserModule(string szName);`
	`int SetUserModuleParameter(string szInstanceName, string szValueName, string szValueValue);`
	`int NextShape(string szInstanceName);`
	`int ClearConfiguration();`
	`int ClearUserModuleParameter(string szInstanceName, string szParamName);`
	`int lGenerateSrc();`
	`int lSelectUserModule(string szName, string szInstanceName);`
	`int lUnSelectUserModule(string szName);`
	`int lPlaceUserModule(string szName);`
	`int lSetUserModuleParameter(string szInstanceName, string szValueName, string szValueValue);`
	`int lNextShape(string szInstanceName);`
	`int lClearConfiguration();`
	`int lClearUserModuleParameter(string szInstanceName, string szParamName);`
	`int SetUserModuleParameter2(string szOverLayName, string szInstanceName, string szValueName, string szValueValue);`
	`int lSetUserModuleParameter2(string szOverLayName, string szInstanceName, string szValueName, string szValueValue);`
	`string GetUserModuleParameter2(string szOverLayName, string szInstanceName, string szValueName);`
	`string GetUserModule(string szOverLayName, string szBlkName);`
	`string GetPinGrid(string szOverLayName);`
	`string GetUserModulePath(string szUMName);`
	`string GetDevicePath(string lpszDeviceName);`
	`string GetBaseDevice();`
	`int EnumerateConfigs(object lpdCallbackTarget, string lpszCallBack);`
	`int EnumerateSelectedUMs(object lpdCallbackTarget, string lpszConfigName, string lpszCallBack);`
	`string GetDBMainPath();`
	`string GetGlobalResourceValue(string lpszOverLayName, string lpszResourceName);`
	`int EnumerateUMParams(object lpdCallbackTarget, string lpszConfigName, string strUMInstName, string strUMName, string lpszCallBack);`
	`short IsUMPlaced(string lpszConfigName, string lpszUMInstName);`
	`int lSetGlobalResourceValue(string sz1, string sz2, string sz3);`

Table 3-3. List of Device Editor Methods *(continued)*

Object	Comments
	`string GetReservedResourceValue(string sz1, string sz2);`
	`int lSetReservedResourceValue(string sz1, string sz2, string sz3);`
	`int lSelectUserModule2(string sz1, string sz2, string sz3);`
	`int lUnSelectUserModule2(string sz1, string sz2);`
	`int lPlaceUserModule2(string sz1, string sz2);`
	`int lUnPlaceUserModule2(string sz1, string sz2);`
	`int lAddConfigOverlay(string sz1);`
	`int lDeleteConfigOverlay(string sz1);`
	`int SetMultipleUserModule(string sOverlayName, string sIntanceName, string sMultiModuleName);`
	`string GetDeviceName();`
	`short IsUMPlaceable(string lpszConfigName, string lpszUMInstName);`
	`int EnumUnsyncDigitalBlocks(object lpdCallbackTarget, string lpszConfigName, string strUMInstName, string strUMName, string lpszCallBack);`
	`int MoveUserModuleShapeToBlock(string sz1, string sz2, string sz3);`
	`int MoveUserModuleShapeToBlock2(string szOverlayName, string szInstanceName, string szBlockName, string szShapeBlkName);`
	`int NextShape2(string szOverlayName, string szInstanceName);`
	`int lNextShape2(string szOverlayName, string szInstanceName);`
	`string GetConfigInfo(string szPath);`
	`int OpenConfiguration(string szPath);`
	`string GetDeviceDisplayName();`
	`string GetPartIncludeFolder(string szDevName);`
	`string GetPartTemplateFolder(string szDevName);`
	`string GetMemoryCapacity();`
	`int EnumerateConfigsXGUI(object lpdCallbackTarget, string lpszCallBack);`
	`int EnumerateSelectedUMsXGUI(object lpdCallbackTarget, string lpszConfigName, string lpszCallBack);`
	`string GetData(string sCmd, string sArg1, string sArg2, string srArg3, string sArg4);`
	`string EnumerateConfigs2();`
	`string EnumerateSelectedUMs2(string lpszConfigName);`
	`string EnumerateUMParams2(string lpszConfigName, string lpszUMInstName, string lpszUMName);`
	`string GetUserModuleFilePathName(string szUMName);`
	`int RefreshGlobalResources(string configName);`
	`int RefreshReservedResources(string configName);`
	`string GetVersionedUserModulePath(string szUMName, string umVersion);`
	`string GetVersionedUserModulePathName(string szUMName, string umVersion);`

You may use the commands from Table 3-1 on page 39 for Device Editor GetData () method, but pay attention that in this case command is the first argument:

```
var devEditor  = PS.GetDeviceEditor();
```

```
var sOverlay = devEditor.GetData("GET_OVERLAY_NAME_FROM_INSTANCE", sInstName,
"", "", "");
```

3.4 Add Placement Javascript

Now we will add a placement JS to our UM. Placement JS sets value of '**TC_PulseWidth**' parameter to '**One-Half Clock**' if doubling the SysClk is enabled ('**SysClk*2 Disable**' global parameter is set to '**No**').

3.4.1 Step 17: Attach Placement Javascript to UM

1. Go to the **Timer16X** folder in the folder with custom UMs.
2. Open the **CY8C27** UM sub-folder and open the *Tmr1627.xml* file
3. Go to **XGUI_LIST** element description.
4. Add the following after TIMER16X_WIZARD XGUI element:

```
<XGUI NAME = "PlaceScript" SRC = "..\scripts\EventHandlers.js" TYPE="UM_PLACE"
>
    <XGUI_FUNC_LIST>
        <XGUI_FUNC FUNC_NAME="Timer16xPlaceUM" />
    </XGUI_FUNC_LIST>
</XGUI>
```

The value of **NAME** attribute is free. A value of **SRC** attribute specifies name of JS file with a function that implements changing value list of 'Clock' UM parameter during UM placing and relative path to it. The value of **FUNC_NAME** attribute determines name of function which is called. The value of **TYPE** attribute should be **UM_PLACE**.

Tmr1627.xml | EventHandlers.js

```
297                 <API_REGISTER_ALIAS NAME = "FUNC_LSB_REG"
298                         SOURCE = "TIMER16_LSB"
299                  REGISTER_NAME = "DIG_BasicFunction"
300         />
301                 <API_REGISTER_ALIAS NAME = "FUNC_MSB_REG"
302                         SOURCE = "TIMER16_MSB"
303                  REGISTER_NAME = "DIG_BasicFunction"
304         />
305                 <API_REGISTER_ALIAS NAME = "INPUT_LSB_REG"
306                         SOURCE = "TIMER16_LSB"
307                  REGISTER_NAME = "DIG_Input"
308         />
309                 <API_REGISTER_ALIAS NAME = "INPUT_MSB_REG"
310                         SOURCE = "TIMER16_MSB"
311                  REGISTER_NAME = "DIG_Input"
312         />
313                 <API_REGISTER_ALIAS NAME = "OUTPUT_LSB_REG"
314                         SOURCE = "TIMER16_LSB"
315                  REGISTER_NAME = "DIG_Output"
316         />
317                 <API_REGISTER_ALIAS NAME = "OUTPUT_MSB_REG"
318                         SOURCE = "TIMER16_MSB"
319                  REGISTER_NAME = "DIG_Output"
320         />
321             </API_REGISTER_ALIAS_LIST>
322
323             <XGUI_LIST>
324                 <XGUI NAME="TIMER16X_WIZARD" DISPLAY_NAME="TIMER16X Wizard..." SRC="
325                 <XGUI NAME = "PlaceScript" SRC = "..\scripts\EventHandlers.js" TYPE=
326                     <XGUI_FUNC_LIST>
327                         <XGUI_FUNC FUNC_NAME="Timer16xPlaceUM" />
328                     </XGUI_FUNC_LIST>
329                 </XGUI>
330             </XGUI_LIST>
331
332             <API_FILE_LIST>
333                 <API_FILE NAME = "Timer16INT.asm"
334           API_PREFIX = "Timer16"
335             API_PATH = "\"
336         />
```

3.4.2 Step #18: Implement Required Functionality in Placement JS

1. Go to **scripts** folder and open *EventHandlers.js* file.
2. Add **Timer16xPlaceUM** function implementation.

Pay attention that placing function has one argument which contains instance name of processed UM.

The example of code which initializes the '**TC_PulseWidth**' parameter is represented in Appendix B on page 66.

Only **PS** global object which is listed in Table 3-2 on page 43 is available in code generation JS. All methods from Table 3-2 on page 43 and Table 3-3 on page 44 may be used in this type of JS.

3.5 Add Code Generation Javascript

An adding of code generation JS to UM require a modifying of UM xml file and creating JS code with required functionality. Code generation JS changes value of ‘CompareValue’ parameter if it exceeds ‘Period’ parameter value.

3.5.1 Step 19: Attach Code Generation Javascript to UM

1. Go to the **Timer16X** folder in the folder with custom UMs.
2. Open the **CY8C27** UM sub-folder and open the *Tmr1627.xml* file
3. Go to **XGUI_LIST** element description.
4. Add the following after PlaceScript element:

```
<XGUI NAME="CodeGenScript" SRC="..\scripts\EventHandlers.js"
TYPE="UM_CODE_GEN">
    <XGUI_FUNC_LIST>
        <XGUI_FUNC FUNC_NAME="Timer16xGenerateCode"/>
    </XGUI_FUNC_LIST>
</XGUI>
```

The value of **NAME** attribute is free. A value of **SRC** attribute specifies name of JS file where function which implements changing value of ‘CompareValue’ UM parameter, and relative path to it. The value of **FUNC_NAME** attribute determines name of function which is called. The value of **TYPE** attribute should be **UM_CODE_GEN**.

EventHandlers.js | Tmr1627.xml | Tmr1627.inc | EventHandlers.js

```
                    <API_REGISTER_ALIAS NAME = "CONTROL_LSB_REG"
                            SOURCE = "TIMER16_LSB"
                     REGISTER_NAME = "CONTROL_0"
            />
                    <API_REGISTER_ALIAS NAME = "CONTROL_MSB_REG"
                            SOURCE = "TIMER16_MSB"
                     REGISTER_NAME = "CONTROL_0"
            />
                    <API_REGISTER_ALIAS NAME = "FUNC_LSB_REG"
                            SOURCE = "TIMER16_LSB"
                     REGISTER_NAME = "DIG_BasicFunction"
            />
                    <API_REGISTER_ALIAS NAME = "FUNC_MSB_REG"
                            SOURCE = "TIMER16_MSB"
                     REGISTER_NAME = "DIG_BasicFunction"
            />
                    <API_REGISTER_ALIAS NAME = "INPUT_LSB_REG"
                            SOURCE = "TIMER16_LSB"
                     REGISTER_NAME = "DIG_Input"
            />
                    <API_REGISTER_ALIAS NAME = "INPUT_MSB_REG"
                            SOURCE = "TIMER16_MSB"
                     REGISTER_NAME = "DIG_Input"
            />
                    <API_REGISTER_ALIAS NAME = "OUTPUT_LSB_REG"
                            SOURCE = "TIMER16_LSB"
                     REGISTER_NAME = "DIG_Output"
            />
                    <API_REGISTER_ALIAS NAME = "OUTPUT_MSB_REG"
                            SOURCE = "TIMER16_MSB"
                     REGISTER_NAME = "DIG_Output"
            />
                </API_REGISTER_ALIAS_LIST>

                <XGUI_LIST>
                    <XGUI NAME="TIMER16X_WIZARD" DISPLAY_NAME="TIMER16X Wizard..." SRC="..\Timer16xPackage_
                    <XGUI NAME = "PlaceScript" SRC = "..\scripts\EventHandlers.js" TYPE="UM_PLACE" >
                    <XGUI NAME="CodeGenScript" SRC="..\scripts\EventHandlers.js" TYPE="UM_CODE_GEN">
                        <XGUI_FUNC_LIST>
                            <XGUI_FUNC FUNC_NAME="Timer16xGenerateCode"/>
                        </XGUI_FUNC_LIST>
                    </XGUI>
                </XGUI_LIST>

                <API_FILE_LIST>
                    <API_FILE NAME = "Timer16INT.asm"
                  API_PREFIX = "Timer16"
                    API_PATH = "\"
            />
                    <API_FILE NAME = "Tmr1627.inc"   />
                    <API_FILE NAME = "Tmr1627.asm"   />
                    <API_FILE NAME = "Tmr1627.h"     />
                </API_FILE_LIST>
            </USER_MODULE>
        </USER_MODULE_LIST>
    </PSOC_DEVICE_DB>
```

5. CRET.

3.5.2 Step 20: Implement Required Functionality in Code Generation JS

1. Go to **scripts** folder and open *EventHandlers.js* file.
2. Add **Timer16xGenerateCode** function implementation.

Ensure that code generation function has one argument which contains instance name of processed UM. The example of code which changes value of '**CompareValue**' parameter is represented in Appendix B on page 66.

Only **PS** global object which is listed in Table 3-2 on page 43 is available in code generation JS. All methods from Table 3-2 on page 43 and Table 3-3 on page 44 may be used in this type of JS.

3.6 Add Design Rule Check Javascript

An adding of DRC JS to UM requires modifying a UM xml file and creating JS code with required functionality. DRC displays a warning message in 'Output' window if calculated value of '**Timer Output Clock**' is greater than 8 MHz. '**Timer Output Clock**' parameter value depends on values of global parameters. See Add Wizard on page 28 for details. Text message is: '*Timer Output Clock for UM_NAME is greater than 8 MHz. That is why UM_NAME can not be connected to internal source.*' where **UM_NAME** will be replaced with current UM instance name.

Follow Step 14: Create JS file on page 41 to create JS file for DRC implementation. Name this file *Timer16xRules.js*. We recommend using separate file for DRC, but it is not obligatory. You may use the same JS file (*EventHandlers.js*) where placement, un-placement, code generation and pre-/post-processing scripts are implemented.

3.6.1 Step 21: Attach Design Rule Check Javascript to UM

1. Go to the **Timer16X** folder in the folder with custom UMs.
2. Open the **CY8C27Timer** UM sub-folder and open the *Tmr1627.xml* file
3. Add the following after API_FILE_LIST element:

```
<RULE_LIST>
    <RULE CODE="" PATH="..\scripts\Timer16xRules.js" CALL="CheckTimerOutput-
Clock" SEVERITY="1"/>
</RULE_LIST>
```

The value of **CODE** attribute is free and may be empty string. A value of **PATH** attribute specifies name of JS file which contains DRC function implementation, and relative path to it. The value of **CALL** attribute determines name of function which is called. The value of **SEVERITY** attribute defines severity of the Rule and possible values are numeric from 1 to 5.

Note: If original UM has DRC, the actual path is set in **PATH** attribute. In this case you should update actual path to relative in PATH attribute like in example above.

EventHandlers.js | Tmr1627.xml

```
303                     <API_REGISTER_ALIAS NAME = "FUNC_LSB_REG"
304                             SOURCE = "TIMER16_LSB"
305                      REGISTER_NAME = "DIG_BasicFunction"
306         />
307                     <API_REGISTER_ALIAS NAME = "FUNC_MSB_REG"
308                             SOURCE = "TIMER16_MSB"
309                      REGISTER_NAME = "DIG_BasicFunction"
310         />
311                     <API_REGISTER_ALIAS NAME = "INPUT_LSB_REG"
312                             SOURCE = "TIMER16_LSB"
313                      REGISTER_NAME = "DIG_Input"
314         />
315                     <API_REGISTER_ALIAS NAME = "INPUT_MSB_REG"
316                             SOURCE = "TIMER16_MSB"
317                      REGISTER_NAME = "DIG_Input"
318         />
319                     <API_REGISTER_ALIAS NAME = "OUTPUT_LSB_REG"
320                             SOURCE = "TIMER16_LSB"
321                      REGISTER_NAME = "DIG_Output"
322         />
323                     <API_REGISTER_ALIAS NAME = "OUTPUT_MSB_REG"
324                             SOURCE = "TIMER16_MSB"
325                      REGISTER_NAME = "DIG_Output"
326         />
327             </API_REGISTER_ALIAS_LIST>
328
329             <XGUI_LIST>
330                 <XGUI NAME="TIMER16X_WIZARD" DISPLAY_NAME="TIMER16X Wizard..." SR
331                 <XGUI NAME = "PlaceScript" SRC = "..\scripts\EventHandlers.js" TY
336                 <XGUI NAME = "CodeGenScript" SRC="..\scripts\EventHandlers.js" TY
341                 <XGUI NAME = "UnPlaceScript" SRC = "..\scripts\EventHandlers.js"
346             </XGUI_LIST>
347
348             <API_FILE_LIST>
349                 <API_FILE NAME = "Timer16INT.asm"
350           API_PREFIX = "Timer16"
351             API_PATH = "\"
352         />
353                 <API_FILE NAME = "Tmr1627.inc"   />
354                 <API_FILE NAME = "Tmr1627.asm"   />
355                 <API_FILE NAME = "Tmr1627.h"     />
356             </API_FILE_LIST>
357             <RULE_LIST>
358                 <RULE CODE="" PATH="..\scripts\Timer16xRules.js" CALL="CheckTimer
359             </RULE_LIST>
360         </USER_MODULE>
361     </USER_MODULE_LIST>
362 </PSOC_DEVICE_DB>
```

4. CRET.

3.6.2 Step 22: Implement Required Functionality in Design Rule Check JS

1. Go to **scripts** folder and open T*imer16xRules.js* file.
2. Add **CheckTimerOutputClock** function implementation.

Be sure that the DRC function has one argument which contains instance name of processed UM. The example of code which display message in 'Output' window during project generation is represented in Appendix C on page 68.

Only **PS** global object which is listed in Table 3-2 on page 43 is available DRC JS. All methods from Table 3-2 on page 43 and Table 3-3 on page 44 may be used in this type of JS. There is one limitation - you cannot change current values, filter value list of UM parameters, change current values of global or reserved resources. You can only read current values and display a warning message.

3.7 Add Un-placement Javascript

An adding of un-placement JS to UM require a modifying of UM xml file and creating JS code with required functionality. Un-placement JS displays information message to notice user about UM removing.

3.7.1 Step 23: Attach Un-placement Javascript to UM

1. Go to the **Timer16X** folder in the folder with custom UMs.
2. Open the **CY8C27** UM sub-folder and open the *Tmr1627.xml* file
3. Go to **XGUI_LIST** element description.
4. Add the following after CodeGenScript element:

```
<XGUI NAME = "UnPlaceScript" SRC = "..\scripts\EventHandlers.js"
TYPE="UM_UNPLACE" >
    <XGUI_FUNC_LIST>
        <XGUI_FUNC FUNC_NAME="Timer16xUnPlaceUM" />
    </XGUI_FUNC_LIST>
</XGUI>
```

The value of **NAME** attribute is free. A value of **SRC** attribute specifies name of JS file where function which displays information message, and relative path to it. The value of **FUNC_NAME** attribute determines name of function which is called. The value of **TYPE** attribute should be **UM_UNPLACE**.

EventHandlers.js | Tmr1627.xml

```
305                 REGISTER_NAME = "DIG_BasicFunction"
306         />
307                 <API_REGISTER_ALIAS NAME = "FUNC_MSB_REG"
308                         SOURCE = "TIMER16_MSB"
309                 REGISTER_NAME = "DIG_BasicFunction"
310         />
311                 <API_REGISTER_ALIAS NAME = "INPUT_LSB_REG"
312                         SOURCE = "TIMER16_LSB"
313                 REGISTER_NAME = "DIG_Input"
314         />
315                 <API_REGISTER_ALIAS NAME = "INPUT_MSB_REG"
316                         SOURCE = "TIMER16_MSB"
317                 REGISTER_NAME = "DIG_Input"
318         />
319                 <API_REGISTER_ALIAS NAME = "OUTPUT_LSB_REG"
320                         SOURCE = "TIMER16_LSB"
321                 REGISTER_NAME = "DIG_Output"
322         />
323                 <API_REGISTER_ALIAS NAME = "OUTPUT_MSB_REG"
324                         SOURCE = "TIMER16_MSB"
325                 REGISTER_NAME = "DIG_Output"
326         />
327             </API_REGISTER_ALIAS_LIST>
328
329             <XGUI_LIST>
330                 <XGUI NAME="TIMER16X_WIZARD" DISPLAY_NAME="TIMER16X Wizard..." SR
331                 <XGUI NAME = "PlaceScript" SRC = "..\scripts\EventHandlers.js" TY
336                 <XGUI NAME="CodeGenScript" SRC="..\scripts\EventHandlers.js" TYPE
341                 <XGUI NAME = "UnPlaceScript" SRC = "..\scripts\EventHandlers.js"
342                     <XGUI_FUNC_LIST>
343                         <XGUI_FUNC FUNC_NAME="Timer16xUnPlaceUM" />
344                     </XGUI_FUNC_LIST>
345                 </XGUI>
346             </XGUI_LIST>
347
348             <API_FILE_LIST>
349                 <API_FILE NAME = "Timer16INT.asm"
350           API_PREFIX = "Timer16"
351             API_PATH = "\"
352         />
353                 <API_FILE NAME = "Tmr1627.inc"    />
354                 <API_FILE NAME = "Tmr1627.asm"    />
355                 <API_FILE NAME = "Tmr1627.h"      />
356             </API_FILE_LIST>
357
358       </USER_MODULE>
359     </USER_MODULE_LIST>
360 </PSOC_DEVICE_DB>
361
```

5. CRET.

3.7.2 Step 24: Implement Required Functionality in Un-placement JS

1. Go to **scripts** folder and open *EventHandlers.js* file.
2. Add **Timer16xUnPlaceUM** function implementation.

Be sure that the un-placement function has no argument. The example of code which display message is represented in Appendix B on page 66.

Only **PS** global object which is listed in Table 3-2 on page 43 is available in un-placement JS. All methods from Table 3-2 on page 43 and Table 3-3 on page 44 may be used in this type of JS. There is one limitation - you cannot change current values or filter value list of UM parameters due to UM instance are not available in PD project at this moment.

A. Appendices

A.1 Appendix A

A.1.1 Timer16xPackage\Timer16xPackage\Timer16xPackage.cs

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using PSoCCommon.PSoCFramework;

namespace Timer16xPackage
{
    [Guid("F68431D4-FCD1-4e9a-A2FA-F59494CE753A"),
    ClassInterface(ClassInterfaceType.None)]
    public class Timer16xPackage:IPSoCDriverPackage
    {
        private string _instanceName;  // Name of UM instance
        private string _packageType;   // "PACKAGE:PROPERTY"
        private string _cmxID;         // project ID
        private object _lock = new object(); // lock object
        protected string _installPath = ""; // Path to UM folder

        #region IPSoCDriverPackage Members

        public string InstallPath
        {
            get { return _installPath; }
            set { _installPath = value; }
        }

        public void OnMessage(object sender, PSoCEventArgs e)
        {

        }

        public void OnQuit(object sender, PSoCEventArgs e)
        {

        }

        public event EventHandler<PSoCEventArgs> PropertiesUpdated;

        public void SetCmxID(string cmxID)
        {
            _cmxID = cmxID;
        }
```

```
        public void SetInstanceName(string instanceName)
        {
            _instanceName = instanceName;
        }

        public void SetPackageType(string packageType)
        {
            _packageType = packageType;
        }

        public void Start()
        {

            if (_packageType == "PACKAGE:PROPERTY")
            {
                DoWizard();
            }

        }

        #endregion

        /// <summary>
        /// cause UM parameters updating
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void OnPropertiesUpdated(object sender, EventArgs e)
        {
            EventHandler<PSoCEventArgs> handler;
            lock (_lock)
            {
                handler = PropertiesUpdated;
            }
            if (handler != null)
            {
                PSoCEventArgs eventArgs = new PSoCEventArgs();
                eventArgs.Data = _instanceName;
                handler(this, eventArgs);
            }
        }

        /// <summary>
        /// launch Wizard Form
        /// </summary>
        private void DoWizard()
        {
            string sOverlayName = CMXEngineAccessor.GetData(_cmxID,
"GET_OVERLAY_NAME_FROM_INSTANCE", _instanceName, "", "");
            WizardForm wizardForm = new WizardForm(_cmxID, _instanceName, sOverlayName);
            if (wizardForm.ShowDialog() == DialogResult.OK)
            {
                OnPropertiesUpdated(this, null);
            }       }
    }
}
```

A.1.2 Timer16xPackage\Timer16xPackage\WizardForm.Designer.cs

```
namespace Timer16xPackage
{
    partial class WizardForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button_OK = new System.Windows.Forms.Button();
            this.grB_Parameters = new System.Windows.Forms.GroupBox();
            this.label_InputClockValue = new System.Windows.Forms.Label();
            this.label_InputClock = new System.Windows.Forms.Label();
            this.label_TimerOutClockValue = new System.Windows.Forms.Label();
            this.label_TimerOutClock = new System.Windows.Forms.Label();
            this.label_Clock = new System.Windows.Forms.Label();
            this.cmB_Clock = new System.Windows.Forms.ComboBox();
            this.label_CompareType = new System.Windows.Forms.Label();
            this.cmB_CompareType = new System.Windows.Forms.ComboBox();
            this.label_Period = new System.Windows.Forms.Label();
            this.label_CompareValue = new System.Windows.Forms.Label();
            this.numUD_Period = new System.Windows.Forms.NumericUpDown();
            this.numUD_CompareValue = new System.Windows.Forms.NumericUpDown();
            this.grB_Parameters.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.numUD_Period)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.numUD_CompareValue)).Begi-
nInit();
            this.SuspendLayout();
            //
            // button_OK
            //
            this.button_OK.Location = new System.Drawing.Point(184, 230);
            this.button_OK.Name = "button_OK";
            this.button_OK.Size = new System.Drawing.Size(75, 23);
            this.button_OK.TabIndex = 0;
            this.button_OK.Text = "OK";
            this.button_OK.UseVisualStyleBackColor = true;
```

```
this.button_OK.Click += new System.EventHandler(this.button_OK_Click);
//
// grB_Parameters
//
this.grB_Parameters.Controls.Add(this.numUD_CompareValue);
this.grB_Parameters.Controls.Add(this.numUD_Period);
this.grB_Parameters.Controls.Add(this.label_CompareValue);
this.grB_Parameters.Controls.Add(this.label_Period);
this.grB_Parameters.Controls.Add(this.cmB_CompareType);
this.grB_Parameters.Controls.Add(this.label_CompareType);
this.grB_Parameters.Controls.Add(this.label_InputClockValue);
this.grB_Parameters.Controls.Add(this.label_InputClock);
this.grB_Parameters.Controls.Add(this.label_TimerOutClockValue);
this.grB_Parameters.Controls.Add(this.label_TimerOutClock);
this.grB_Parameters.Controls.Add(this.label_Clock);
this.grB_Parameters.Controls.Add(this.cmB_Clock);
this.grB_Parameters.Location = new System.Drawing.Point(12, 12);
this.grB_Parameters.Name = "grB_Parameters";
this.grB_Parameters.Size = new System.Drawing.Size(247, 212);
this.grB_Parameters.TabIndex = 1;
this.grB_Parameters.TabStop = false;
this.grB_Parameters.Text = "Parameters";
//
// label_InputClockValue
//
this.label_InputClockValue.AutoSize = true;
this.label_InputClockValue.Location = new System.Drawing.Point(110, 58);
this.label_InputClockValue.Name = "label_InputClockValue";
this.label_InputClockValue.Size = new System.Drawing.Size(19, 13);
this.label_InputClockValue.TabIndex = 5;
this.label_InputClockValue.Text = "----";
//
// label_InputClock
//
this.label_InputClock.AutoSize = true;
this.label_InputClock.Location = new System.Drawing.Point(43, 58);
this.label_InputClock.Name = "label_InputClock";
this.label_InputClock.Size = new System.Drawing.Size(61, 13);
this.label_InputClock.TabIndex = 4;
this.label_InputClock.Text = "Input Clock";
//
// label_TimerOutClockValue
//
this.label_TimerOutClockValue.AutoSize = true;
this.label_TimerOutClockValue.Location = new System.Drawing.Point(110, 89);
this.label_TimerOutClockValue.Name = "label_TimerOutClockValue";
this.label_TimerOutClockValue.Size = new System.Drawing.Size(19, 13);
this.label_TimerOutClockValue.TabIndex = 3;
this.label_TimerOutClockValue.Text = "----";
//
// label_TimerOutClock
//
this.label_TimerOutClock.AutoSize = true;
this.label_TimerOutClock.Location = new System.Drawing.Point(6, 89);
this.label_TimerOutClock.Name = "label_TimerOutClock";
this.label_TimerOutClock.Size = new System.Drawing.Size(98, 13);
this.label_TimerOutClock.TabIndex = 2;
this.label_TimerOutClock.Text = "Timer Output Clock";
```

```
            //
            // label_Clock
            //
            this.label_Clock.AutoSize = true;
            this.label_Clock.Location = new System.Drawing.Point(70, 27);
            this.label_Clock.Name = "label_Clock";
            this.label_Clock.Size = new System.Drawing.Size(34, 13);
            this.label_Clock.TabIndex = 1;
            this.label_Clock.Text = "Clock";
            //
            // cmB_Clock
            //
            this.cmB_Clock.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.cmB_Clock.FormattingEnabled = true;
            this.cmB_Clock.Location = new System.Drawing.Point(110, 24);
            this.cmB_Clock.Name = "cmB_Clock";
            this.cmB_Clock.Size = new System.Drawing.Size(121, 21);
            this.cmB_Clock.TabIndex = 0;
            this.cmB_Clock.SelectedIndexChanged += new System.EventHan-
dler(this.cmB_Clock_SelectedIndexChanged);
            //
            // label_CompareType
            //
            this.label_CompareType.AutoSize = true;
            this.label_CompareType.Location = new System.Drawing.Point(28, 117);
            this.label_CompareType.Name = "label_CompareType";
            this.label_CompareType.Size = new System.Drawing.Size(76, 13);
            this.label_CompareType.TabIndex = 6;
            this.label_CompareType.Text = "Compare Type";
            //
            // cmB_CompareType
            //
           this.cmB_CompareType.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDown-
List;
            this.cmB_CompareType.FormattingEnabled = true;
            this.cmB_CompareType.Location = new System.Drawing.Point(113, 114);
            this.cmB_CompareType.Name = "cmB_CompareType";
            this.cmB_CompareType.Size = new System.Drawing.Size(121, 21);
            this.cmB_CompareType.TabIndex = 7;
            //
            // label_Period
            //
            this.label_Period.AutoSize = true;
            this.label_Period.Location = new System.Drawing.Point(67, 146);
            this.label_Period.Name = "label_Period";
            this.label_Period.Size = new System.Drawing.Size(37, 13);
            this.label_Period.TabIndex = 8;
            this.label_Period.Text = "Period";
            //
            // label_CompareValue
            //
            this.label_CompareValue.AutoSize = true;
            this.label_CompareValue.Location = new System.Drawing.Point(25, 177);
            this.label_CompareValue.Name = "label_CompareValue";
            this.label_CompareValue.Size = new System.Drawing.Size(79, 13);
            this.label_CompareValue.TabIndex = 9;
            this.label_CompareValue.Text = "Compare Value";
            //
```

```
            // numUD_Period
            //
            this.numUD_Period.Location = new System.Drawing.Point(113, 144);
            this.numUD_Period.Name = "numUD_Period";
            this.numUD_Period.Size = new System.Drawing.Size(120, 20);
            this.numUD_Period.TabIndex = 10;
            this.numUD_Period.ValueChanged += new System.EventHan-
dler(this.numUD_Period_ValueChanged);
            //
            // numUD_CompareValue
            //
            this.numUD_CompareValue.Location = new System.Drawing.Point(113, 175);
            this.numUD_CompareValue.Name = "numUD_CompareValue";
            this.numUD_CompareValue.Size = new System.Drawing.Size(120, 20);
            this.numUD_CompareValue.TabIndex = 11;
            //
            // WizardForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(267, 261);
            this.Controls.Add(this.grB_Parameters);
            this.Controls.Add(this.button_OK);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "WizardForm";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Timer16X Wizard";
            this.Load += new System.EventHandler(this.WizardForm_Load);
            this.grB_Parameters.ResumeLayout(false);
            this.grB_Parameters.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.numUD_Period)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.numUD_CompareValue)).EndInit();
            this.ResumeLayout(false);
        }
        #endregion
        private System.Windows.Forms.Button button_OK;
        private System.Windows.Forms.GroupBox grB_Parameters;
        private System.Windows.Forms.Label label_InputClockValue;
        private System.Windows.Forms.Label label_InputClock;
        private System.Windows.Forms.Label label_TimerOutClockValue;
        private System.Windows.Forms.Label label_TimerOutClock;
        private System.Windows.Forms.Label label_Clock;
        private System.Windows.Forms.ComboBox cmB_Clock;
        private System.Windows.Forms.NumericUpDown numUD_CompareValue;
        private System.Windows.Forms.NumericUpDown numUD_Period;
        private System.Windows.Forms.Label label_CompareValue;
        private System.Windows.Forms.Label label_Period;
        private System.Windows.Forms.ComboBox cmB_CompareType;
        private System.Windows.Forms.Label label_CompareType;
    }
}
```

A.1.3 Timer16xPackage\Timer16xPackage\WizardForm.cs

```
using System;
using System.Text.RegularExpressions;
using System.Windows.Forms;
using System.Xml;
using PSoCCommon.PSoCFramework;

namespace Timer16xPackage
{
    public partial class WizardForm : Form
    {
        #region Private Vars
        private string _cmxID;         // project ID
        private string _instanceName;  // Name of UM instance
        private string _overlayName;   // project name
        private XmlDocument _xmlDoc = new XmlDocument();
        #endregion Private Vars

        #region Instructor/Destructor
        public WizardForm()
        {
            InitializeComponent();
        }

        public WizardForm(string cmxId, string iName, string sOverlayName)
        {
            InitializeComponent();
            _instanceName = iName; //set instance name of User Module
            _cmxID = cmxId; //set ID of CMX engine
            _overlayName = sOverlayName;
        }
        #endregion Instructor/Destructor

        #region Handlers
        private void WizardForm_Load(object sender, EventArgs e)
        {

            string sUmParamsDom = CMXEngineAccessor.GetData(_cmxID,
"GET_USERMODULE_PARAMETERS", _overlayName, _instanceName, "");
            _xmlDoc.LoadXml(sUmParamsDom);

            //Initialize Clock combobox with values of 'Clock' UM parameter
            InitializeComboBox(cmB_Clock,"Clock");
            if (cmB_Clock.SelectedItem!=null)
                SetInputClock();

            //Initialize CompareType combobox with values of 'CompareType' UM parameter
            InitializeComboBox(cmB_CompareType, "CompareType");

            //Initialize Period numeric up down control with values of 'Period' UM parameter
            InitializeNumericUpDown(numUD_Period, "Period");
            //Initialize CompareValue numeric up down control with values of 'CompareValue'
UM parameter
            InitializeNumericUpDown(numUD_CompareValue, "CompareValue");
        }

        private void button_OK_Click(object sender, EventArgs e)
```

```
        {
            //set User Module Parameters
            CMXEngineAccessor.GetData(_cmxID, "SET_USERMODULE_PARAM_NO_OVERLAY",
_instanceName, "Clock", cmB_Clock.SelectedItem.ToString());
            CMXEngineAccessor.GetData(_cmxID, "SET_USERMODULE_PARAM_NO_OVERLAY",
_instanceName, "CompareType", cmB_CompareType.SelectedItem.ToString());
            CMXEngineAccessor.GetData(_cmxID, "SET_USERMODULE_PARAM_NO_OVERLAY",
_instanceName, "Period", numUD_Period.Value.ToString());
            CMXEngineAccessor.GetData(_cmxID, "SET_USERMODULE_PARAM_NO_OVERLAY",
_instanceName, "CompareValue", numUD_CompareValue.Value.ToString());
CMXEngineAccessor.GetData(_cmxID, "SET_USERMODULE_PARAM_NO_OVERLAY", _instanceName, "Tim-
eLastModified", DateTime.Now.ToShortDateString() + " " + DateTime.Now.ToShortTimeString());
            DialogResult = DialogResult.OK;
            Close();
        }

        private void cmB_Clock_SelectedIndexChanged(object sender, EventArgs e)
        {
            SetInputClock();
            SetOutputClock();
        }

        private void numUD_Period_ValueChanged(object sender, EventArgs e)
        {
            SetOutputClock();
        }
        #endregion Handlers

        #region Private Methods
        /// <summary>
        /// Initialize combobox with values of corresponding User Module parameters
        /// </summary>
        /// <param name="cmb">Instance of Combobox on Wizard</param>
        /// <param name="sParameterName">User Module parameter name</param>
        private void InitializeComboBox(ComboBox cmb, string sParameterName)
        {
            //value list
            XmlNodeList propValues = _xmlDoc.SelectNodes("//CMX_PROPERTY[@NAME=\"" + sParam-
eterName + "\"]//CMX_PROPERTY_VALUE_LIST//CMX_PROPERTY_VALUE");
            if (propValues != null)
            {
                for (int i = 0; i < propValues.Count; i++)
                {
                    cmb.Items.Add((propValues[i] as XmlElement).GetAttribute("NAME"));
                }
            }

            //current value
            XmlElement propElem = _xmlDoc.SelectSingleNode("//CMX_PROPERTY[@NAME=\"" + sPa-
rameterName + "\"]") as XmlElement;
            if (propElem != null)
                cmb.SelectedItem = propElem.GetAttribute("CURRENT_VALUE");
        }

       private void InitializeNumericUpDown (NumericUpDown numericUpDown, string sParameter-
Name)
        {
```

```
            XmlElement propElem = _xmlDoc.SelectSingleNode("//CMX_PROPERTY[@NAME=\"" + sPa-
rameterName + "\"]") as XmlElement;
            if (propElem != null)
            {
                string sTemp = propElem.GetAttribute("MIN");
                numericUpDown.Minimum = Int32.Parse(sTemp);
                sTemp = propElem.GetAttribute("MAX");
                numericUpDown.Maximum = Int32.Parse(sTemp);
                sTemp = propElem.GetAttribute("CURRENT_VALUE");
                numericUpDown.Value = Int32.Parse(sTemp);
            }
        }

        /// <summary>
        /// Get current value of global resource
        /// </summary>
        /// <param name="resourceName">Global Resource Name</param>
        /// <returns>Current value of global resource</returns>
        private string GetGlobalResourceValue(string resourceName)
        {
            string sGlobalResources = CMXEngineAccessor.GetData(_cmxID,
"GET_GLOBAL_RESOURCES", _overlayName, _instanceName, "");
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.LoadXml(sGlobalResources); // Load global resource DOM
            XmlElement resourceElem = xmlDoc.SelectSingleNode("//CMX_PROPERTY[@NAME=\"" +
resourceName + "\"]") as XmlElement;
            if (resourceElem != null)
                return resourceElem.GetAttribute("CURRENT_VALUE");
            return null;
        }

        /// <summary>
        /// Set Input Clock value
        /// This value depends on 'Clock' UM parameter and
        /// values of global resource
        /// </summary>
        private void SetInputClock()
        {
            long lInputClockValue;
            //get current value of 'Clock' parameter
            string sClockValue = cmB_Clock.SelectedItem.ToString();
            //get value of 'Power Setting [ Vcc / SysClk freq ]'
            string sSysClkValue = GetGlobalResourceValue("Power Setting [ Vcc / SysClk freq
]");
            sSysClkValue = sSysClkValue.Substring(sSysClkValue.IndexOf('/') + 1);
            sSysClkValue = Regex.Match(sSysClkValue, @"\d+").Value;
            long lSysClkValue = Int32.Parse(sSysClkValue) * 1000000; //convert to Hz

            //get value of 'VC1= SysClk/N'
            string sVC1Value = GetGlobalResourceValue("VC1= SysClk/N");
            long lVC1Value = lSysClkValue / long.Parse(sVC1Value);

            //get value of 'VC2= VC1/N'
            string sVC2Value = GetGlobalResourceValue("VC2= VC1/N");
            long lVC2Value = lVC1Value / long.Parse(sVC2Value);

            //get value of 'VC3'
            string sVC3Source = GetGlobalResourceValue("VC3 Source");
```

```
        string sVC3Divider = GetGlobalResourceValue("VC3 Divider");
        long lVC3Value;
        //calculate VC3 value: VC3 Source/VC3 Divider
        switch (sVC3Source)
        {
            case "SysClk/1":
                lVC3Value = lSysClkValue / Int16.Parse(sVC3Divider);
                break;
            case "VC1":
                lVC3Value = lVC1Value / Int16.Parse(sVC3Divider);
                break;
            case "SysClk*2":
                lVC3Value = lSysClkValue * 2 / Int16.Parse(sVC3Divider);
                break;
            default: //VC2
                lVC3Value = lVC2Value / Int16.Parse(sVC3Divider);
                break;
        }
        //calculate value of Input Clock
        switch (sClockValue)
        {
            case "SysClk*2":
                lInputClockValue = lSysClkValue * 2;
                break;
            case "VC1":
                lInputClockValue = lVC1Value;
                break;
            case "VC2":
                lInputClockValue = lVC2Value;
                break;
            case "VC3":
                lInputClockValue = lVC3Value;
                break;
            case "CPU_32_KHz":
                lInputClockValue = 32000;
                break;
            default:
                lInputClockValue = -1;
                break;
        }
        //display value of Input Clock
        if (lInputClockValue == -1)
            label_InputClockValue.Text = "----";
        else label_InputClockValue.Text = lInputClockValue + " Hz";
    }

    /// <summary>
    /// Set value of Output Clock
    /// Output Clock = Input Clock/(Period+1)
    /// </summary>
    private void SetOutputClock()
    {
        if (label_InputClockValue.Text != "----")
        {
            //get value of Input Clock
            string sInputClock = Regex.Match(label_InputClockValue.Text, @"\d+").Value;
            //convert string to long
            long lInputClock = long.Parse(sInputClock);
```

```
                //get value of Period
                long lPeriod = long.Parse(numUD_Period.Value.ToString());
                //calculate Output Clock
                long lOutputClock = lInputClock/(lPeriod + 1);
                //Display Output Clock
                label_TimerOutClockValue.Text = lOutputClock + " Hz";
            }
        }

        #endregion Private Methods
    }
}
```

A.2 Appendix B

A.2.1 Timer16X\Ver_2_6\scripts\EventHandlers.js

```
/*
This function is calling before CompareOut
value parameter changing
*/
function CompareOutSelect(strParam)
{
    var devEditor  = PS.GetDeviceEditor();
    var xmlDOM = new ActiveXObject("Microsoft.XMLDOM");
    xmlDOM.validateOnParse = false;
    xmlDOM.setProperty("SelectionLanguage","XPath");
    xmlDOM.loadXML(strParam);

    var sInstName = xmlDOM.documentElement.getAttribute("UM_INST_NAME");
    var sOverlay = devEditor.GetData("GET_OVERLAY_NAME_FROM_INSTANCE", sInstName, "", "",
"");

    var sTerminalCountOut = devEditor.GetUserModuleParameter2(sOverlay, sInstName, "Termi-
nalCountOut");//get current value
    var sCompareOutSettingsArray = otto.GetCurUMParamValueRange("CompareOut").split(",");//
get values and compose array from them
    //Remove the value that is being used by the TerminalCountOut UM parameter from the list
    for (var i = 0; i < sCompareOutSettingsArray.length; i++)
    {
        if (sCompareOutSettingsArray[i] == sTerminalCountOut)
        {
            sCompareOutSettingsArray.splice(i, 1);
        }
        }
    //Generate new list with the value, being use by TerminalCountOut, taken out
    otto.SetCurUMParamValueRange("CompareOut", sCompareOutSettingsArray.join());
}


/*configure current value of 'TC_PulseWidth' UM parameter
during User Module placement
if value of 'SysClk*2 Disable' global parameter is 'Yes'
'TC_PulseWidth' parameter should have 'Full Clock', otherwise - 'One-Half Clock'
*/
 function Timer16xPlaceUM(sInstName)
{
    var devEditor  = PS.GetDeviceEditor();
    var sOverlay = devEditor.GetData("GET_OVERLAY_NAME_FROM_INSTANCE", sInstName, "", "",
"");//get project ID
   var sSysclk = devEditor.GetGlobalResourceValue(sOverlay, "SysClk*2 Disable");//get value
of 'SysClk*2 Disable' parameter

    if(sSysclk=='No')
        devEditor.SetUserModuleParameter2(sOverlay, sInstName, "TC_PulseWidth", "One-Half
Clock");//set current value
}


/*set current value of 'CompareValue' UM parameter
```

```
to 'Period' parameter value
if 'CompareValue' is great then 'Period'
during project generation
*/
function Timer16xGenerateCode(sInstName)
{
    var devEditor  = PS.GetDeviceEditor();
    var sOverlay = devEditor.GetData("GET_OVERLAY_NAME_FROM_INSTANCE", sInstName, "", "",
"");//get project ID

    var sPeriod = devEditor.GetUserModuleParameter2(sOverlay, sInstName, "Period");//get
current value
    var sCompareValue = devEditor.GetUserModuleParameter2(sOverlay, sInstName, "Compare-
Value");//get current value

    if(parseInt(sCompareValue)>parseInt(sPeriod))//if CompareValue > Period
        //set current value of CompareValue to Period
        devEditor.SetUserModuleParameter2(sOverlay, sInstName, "CompareValue", sPeriod);
}

/*Show a dialog when the UM is removed
*/
function Timer16xUnPlaceUM()
{
    PS.ShowMessageBox("Timer16X Custom User Module was removed from current project.")
}
```

A.3 Appendix C

A.3.1 Timer16X\Ver_2_6\scripts\Timer16xRules.js

```
function CheckTimerOutputClock(sInstName)
{
    var devEditor  = PS.GetDeviceEditor();
    var sOverlay = devEditor.GetData("GET_OVERLAY_NAME_FROM_INSTANCE", sInstName, "", "",
"");

    //get current value of 'Clock' parameter
    var sClockValue = devEditor.GetUserModuleParameter2(sOverlay, sInstName, "Clock");
    var sPeriodValue = devEditor.GetUserModuleParameter2(sOverlay, sInstName, "Period");

    //get value of 'Power Setting [ Vcc / SysClk freq ]'
    var sSysClkValue = devEditor.GetGlobalResourceValue(sOverlay, "Power Setting [ Vcc / Sys-
Clk freq ]");
    var lSysClkValue;
    switch(sSysClkValue)
    {
        case "5.0V / 24MHz":
        case "3.3V / 24MHz": lSysClkValue = "24000000"; break;
        case "2.7V / 12MHz": lSysClkValue = "12000000"; break;
        case "5.0V / 6MHz" :
        case "3.3V / 6MHz" :
        case "2.7V / 6MHz" : lSysClkValue = "6000000";  break;
        default: lSysClkValue = "24000000"; break;
    }


    //get value of 'VC1= SysClk/N'
    var sVC1Value = devEditor.GetGlobalResourceValue(sOverlay, "VC1= SysClk/N");
    var lVC1Value = lSysClkValue/parseInt(sVC1Value);
    //get value of 'VC2= VC1/N'
    var sVC2Value = devEditor.GetGlobalResourceValue(sOverlay, "VC2= VC1/N");
    var lVC2Value = lVC1Value/parseInt(sVC2Value);
    //get value of 'VC3'
    var sVC3Source  = devEditor.GetGlobalResourceValue(sOverlay, "VC3 Source");
    var sVC3Divider   = devEditor.GetGlobalResourceValue(sOverlay, "VC3 Divider");
     //calculate VC3 value: VC3 Source/VC3 Divider
    switch (sVC3Source)
    {
        case "SysClk/1":
            lVC3Value = lSysClkValue/parseInt(sVC3Divider);
            break;
        case "VC1":
            lVC3Value = lVC1Value/parseInt(sVC3Divider);
            break;
        case "SysClk*2":
            lVC3Value = lSysClkValue * 2/parseInt(sVC3Divider);
            break;
        default: //VC2
            lVC3Value = lVC2Value/parseInt(sVC3Divider);
            break;
    }
    //calculate value of Input Clock
    switch (sClockValue)
```

```
    {
        case "SysClk*2":
            lInputClockValue = lSysClkValue * 2;
            break;
        case "VC1":
            lInputClockValue = lVC1Value;
            break;
        case "VC2":
            lInputClockValue = lVC2Value;
            break;
        case "VC3":
            lInputClockValue = lVC3Value;
            break;
        case "CPU_32_KHz":
            lInputClockValue = 32000;
            break;
        default:
            lInputClockValue = -1;
            break;
    }

    var lTimerOutputClock;
    if(lInputClockValue!=-1)
        lTimerOutputClock = lInputClockValue/(parseInt(sPeriodValue)+1);
    if(lTimerOutputClock>8000000)
        ShowWarningMsg("Timer Output Clock for " + sInstName + " is greater than 8 MHz. " +
" That is why " + sInstName +" can not be connected to internal source.");

}
```