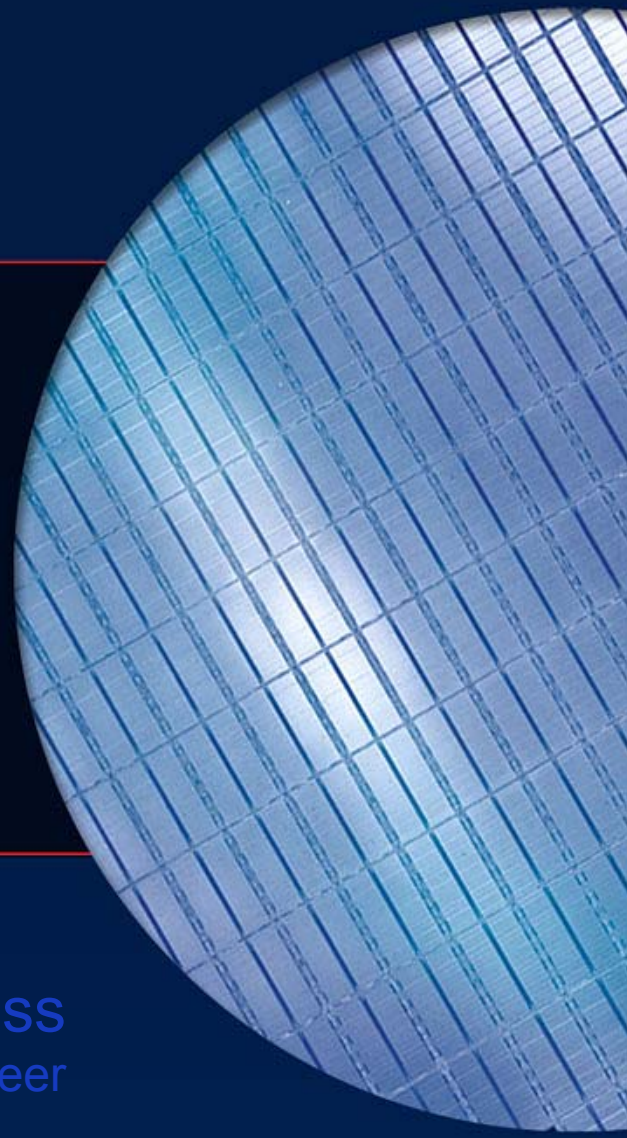
Creating and Modifying User Modules

And other Unnatural Acts



Dave Van Ess
Principal Applications Engineer

CYPRESS

PERFORM

Objectives

Objective:

- Introductive the fundamental components of a PSoC User Module.

- Show how XML is used to describe the intraconnection of the PSoC recourses.

- Create an improved UM (timer16X)using the old UM (timer16) as a template.

User Modules

A User Module is:

- The information required to connect PSoC blocks.
- Software (API) to control it
- A Datasheet
- An Icon

User Modules

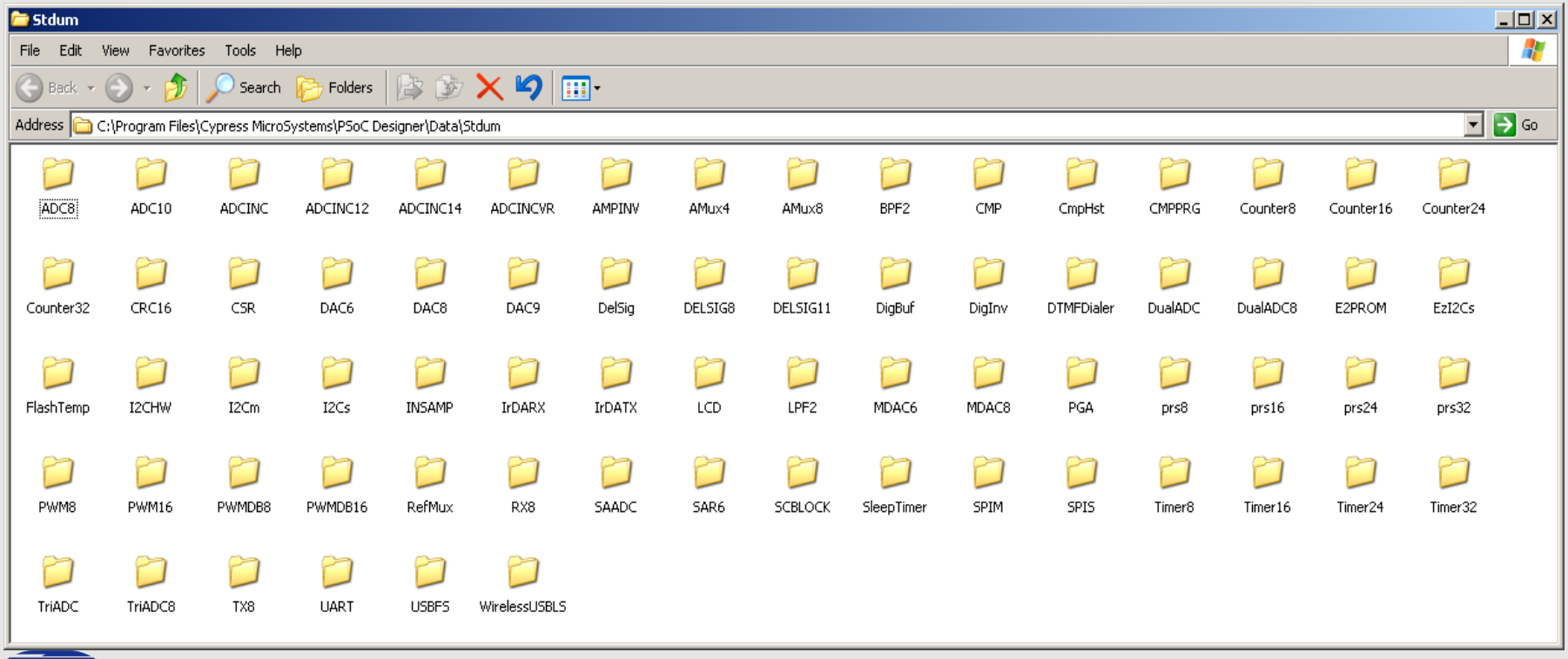
User Module

- At its most basic form is a directory and an XML file with the same name.
  - The same name must also appear as the NAME attribute in the <PSOC_USER_MODULE> element in the UM XML file.
- If done correctly the UM is added to the active UM library.
- Other files exist in the User Module directory, but their naming is controlled by attributes within the UM XML file.
- The other files which are relevant to the UM
  - *.ico – User Module icon
  - *.htm – User Module data sheet
  - *.emf – User Module block diagram

User Modules

User Module Directory

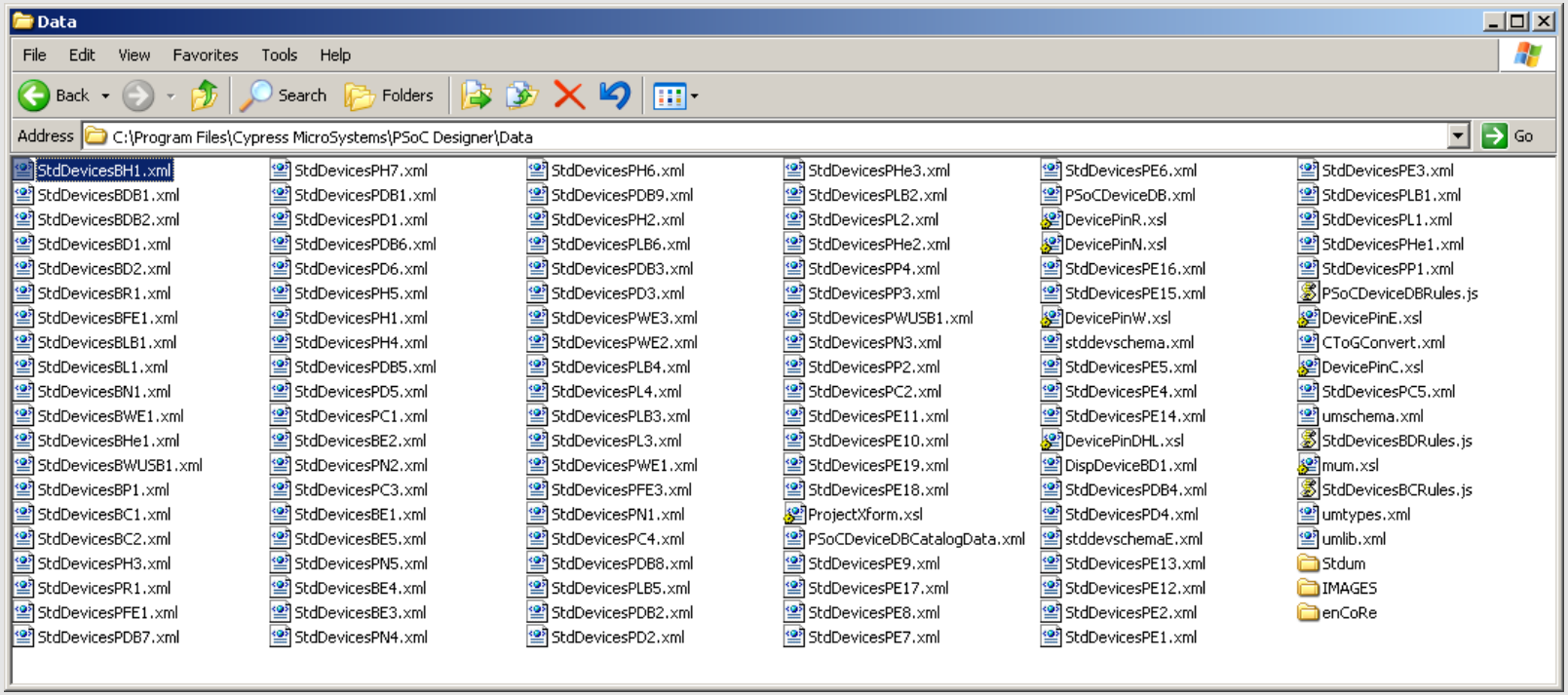
- All UMs located at C:\Program Files\Cypress MicroSystems\PSoC Designer\Data\Stdum



User Modules

User Module Directory

- All device descriptions located at C:\Program Files\Cypress MicroSystems\PSoC Designer\Data
  - Think of each as a dictionary of acceptable register and bit field values for each particular device.



User Modules

Making New User Modules:

- Building a UM from scratch is extremely difficult to do.
- It is far easier to start with a copy of an established UM and modify it to meet your needs.
- Information from several different UMs can be combined to produce a new UM.
  - The counter8 XML and the DAC6 XML could be combined to produce a waveform generator UM.
    - Think of it as gene splicing.

User Modules

Project Goal

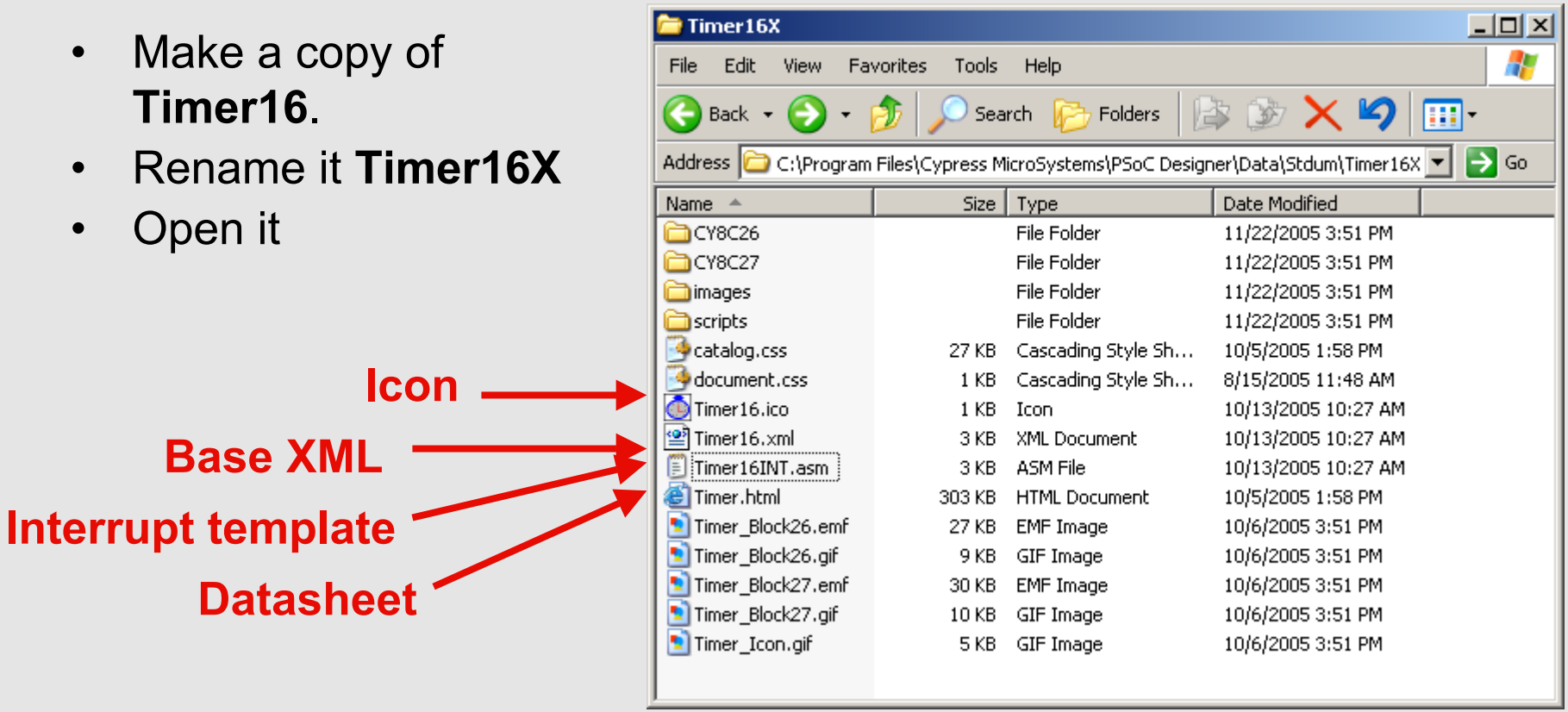
Build an improved Timer16

- The upper digital block of the present Timer16 has a parameter selection to interrupt on:
  - Capture
  - Terminal Count
  - Compare True
- The goal is to create a new UM (timer16X) that:
  - Generates a Terminal Count interrupt for the upper digital block.
  - Generates a Capture interrupt for the lower digital block.

User Modules

Step 1

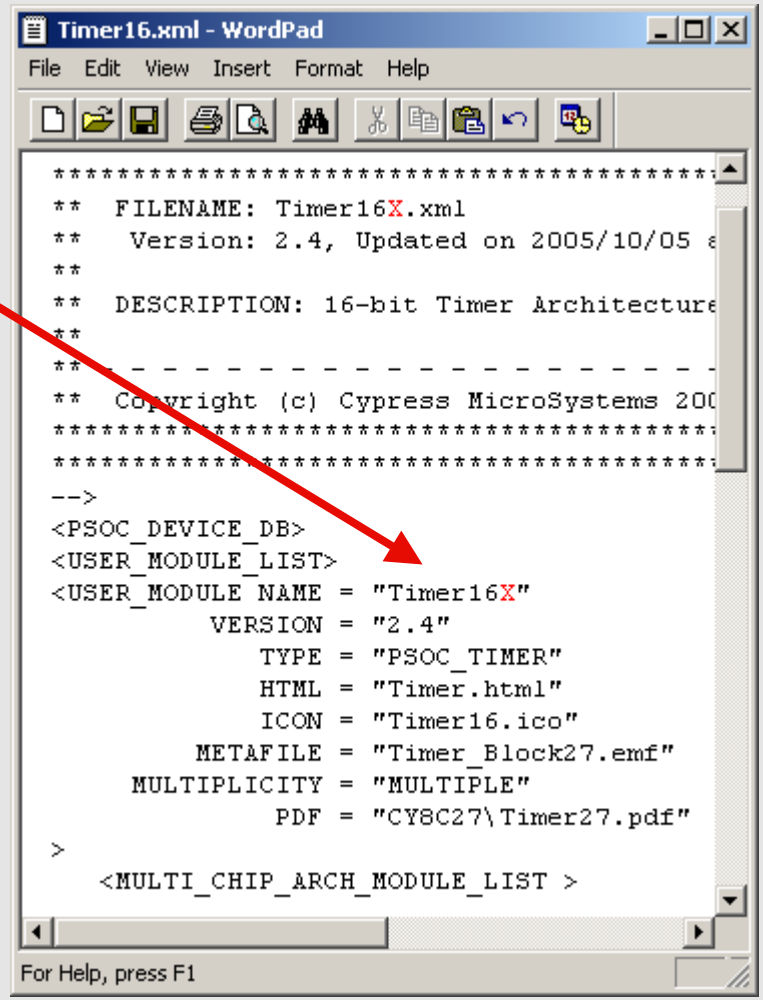
- Make a copy of **Timer16**.
- Rename it **Timer16X**
- Open it



User Modules

Step 2

- Open **Timer16.xml**
  - Use your preferred text editor
- Change **USER_MODULE_NAME** to **TimerX.**
- Might as well change the FILENAME: while you're here.
  - Its only a comment but it can't hurt.
- Close the file and reopen with Internet Explorer.
  - Should open with no problems.
  - This checking process will be done frequently.
  - Lets call it CRET



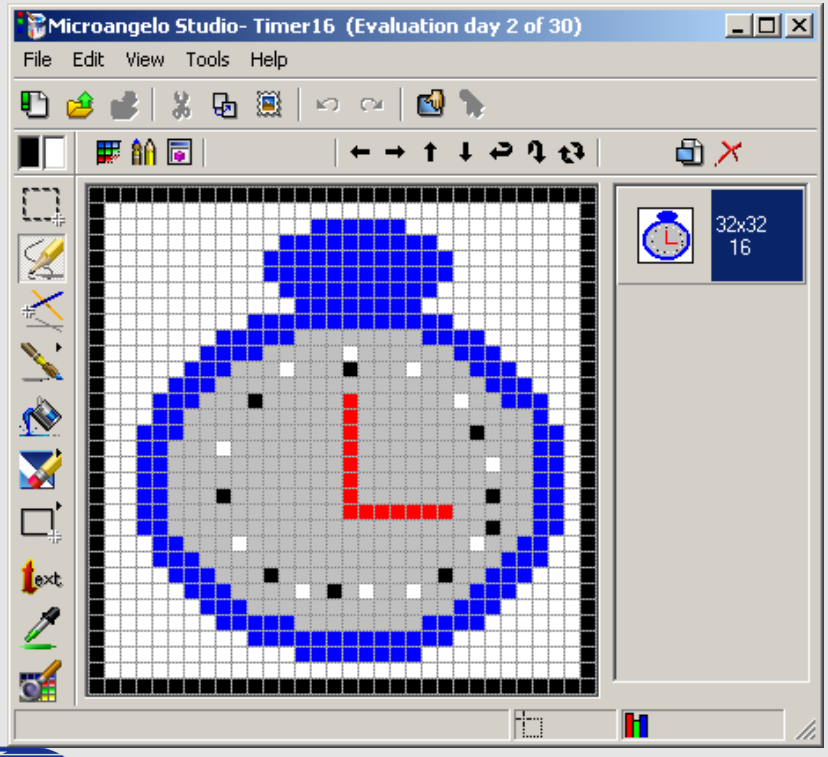

```
<PSOC_DEVICE_DB>
<USER_MODULE_LIST>
<USER_MODULE NAME = "Timer16X"
             VERSION = "2.4"
                TYPE = "PSOC_TIMER"
                HTML = "Timer.html"
                ICON = "Timer16.ico"
            METAFILE = "Timer_Block27.emf"
       MULTIPLICITY = "MULTIPLE"
                 PDF = "CY8C27\Timer27.pdf"
>
   <MULTI_CHIP_ARCH_MODULE_LIST >
```

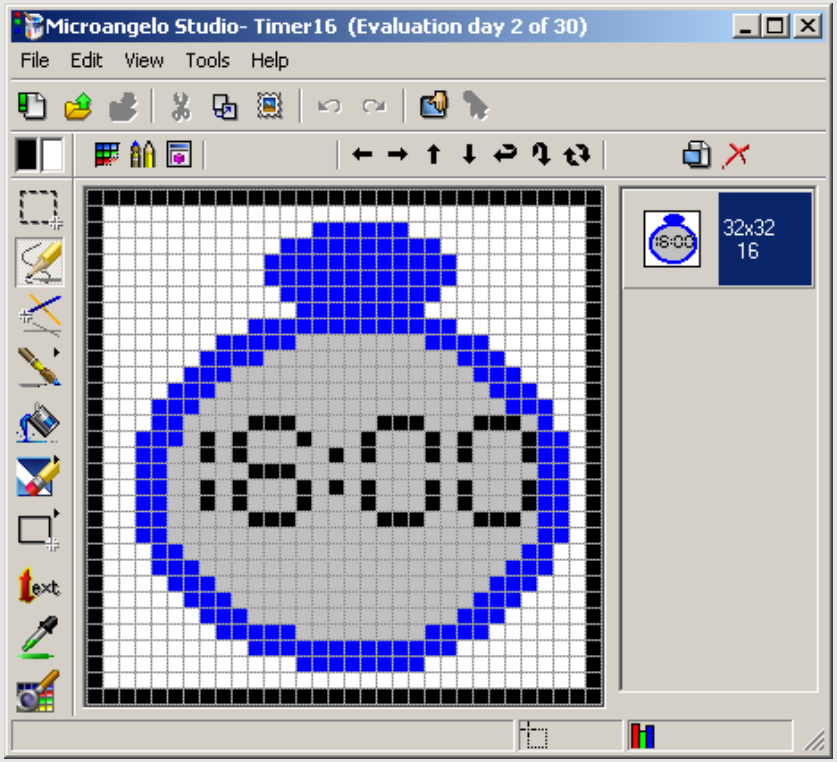

Close, Reopen with Explorer to Test

User Modules

Step 3

- Open **Timer16.ico**
- Alter Icon





User Modules

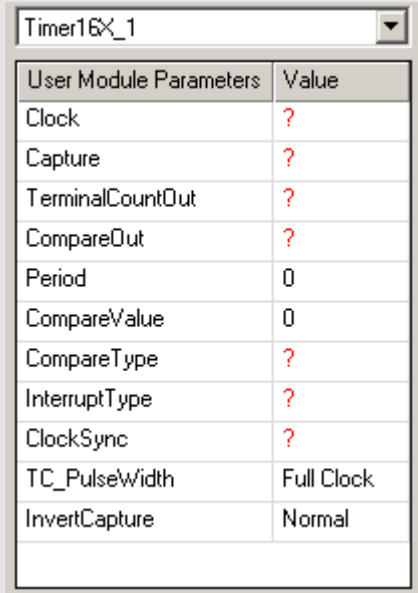
Sanity Check!

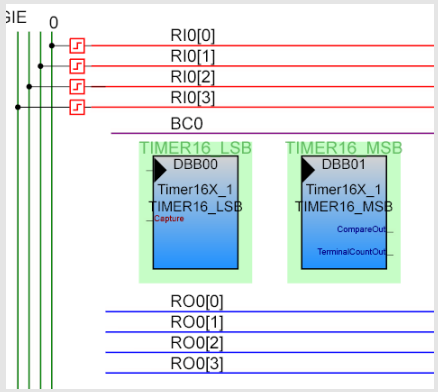
Open a 27x project with Designer

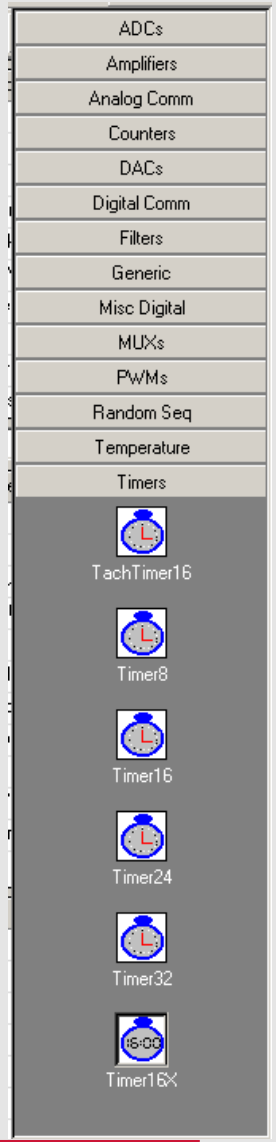
- Verify that UM is present.
- Verify that it places.

For many UMs this is all the XML changes that are required.

- For UMs that have significant changes in software but no hardware changes.
  - This is your chance to change the APIs of your favorite UM to the way that you (*and God)* think they should be, while leaving the base UM hardware configuration alone!







CYPRESS PERFORM

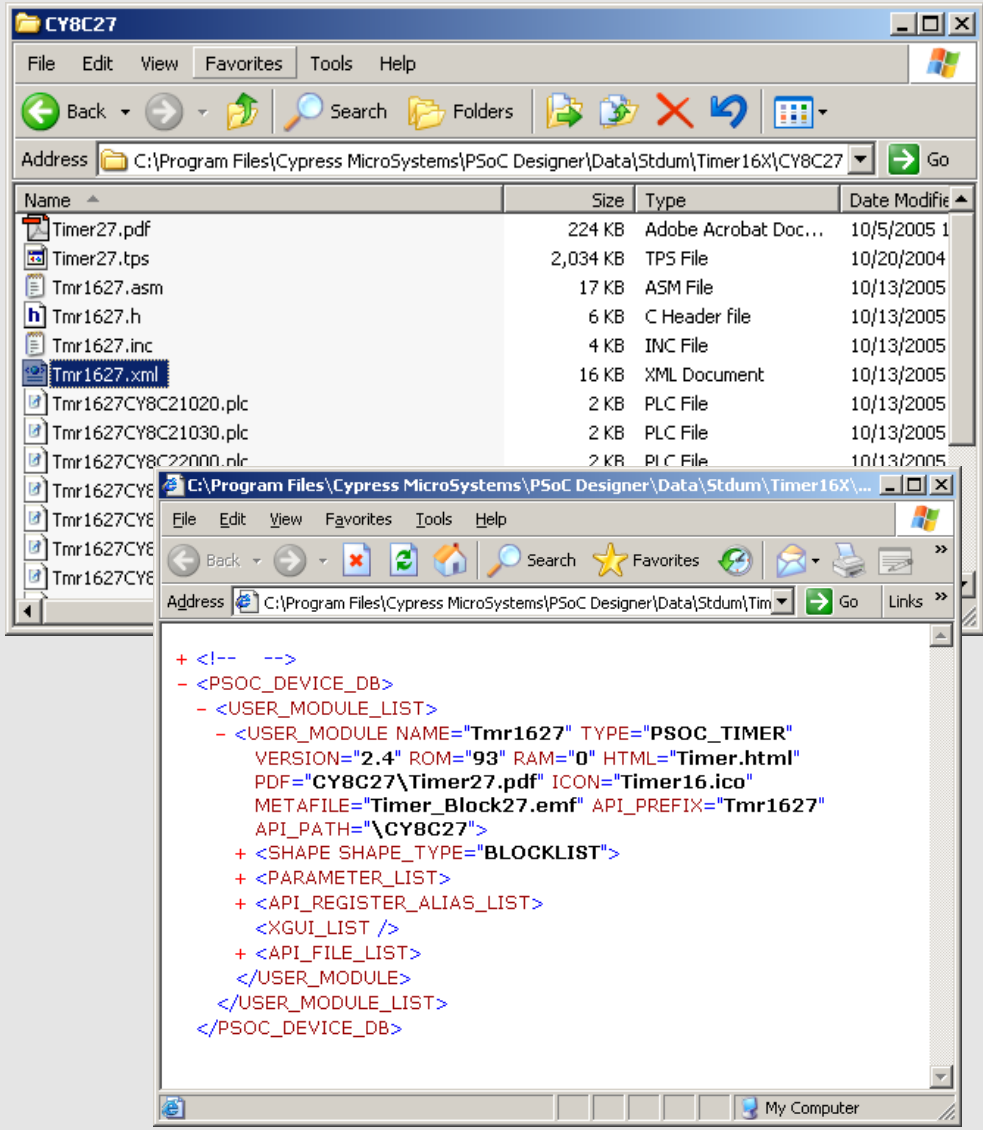
User Modules

Open **CY8C27**

Open **Tmr1627.xml**

XML has 4 main sections

- SHAPE
  - For bit fields are set that don't change.
- PARAMETER_LIST
  - For bit fields are set by user selection.
- API_REGISTER_ALIS_LIST
  - Allows appropriate names of registers.
- API_FILE LIST
  - Lists the API files to be generated.

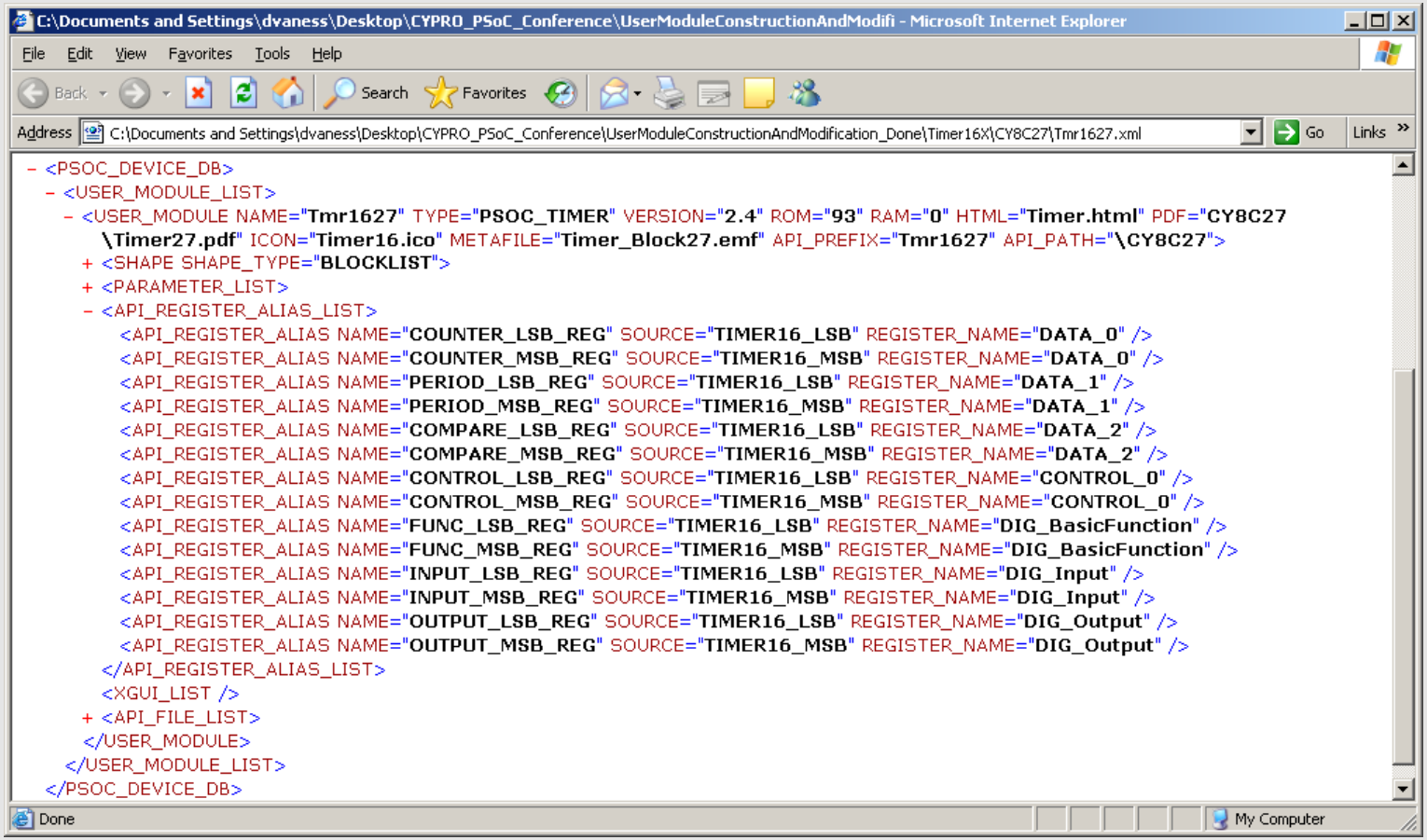


PERFORM

User Modules

API_REGISTER_ALIAS_LIST

- Allows you to define register names.

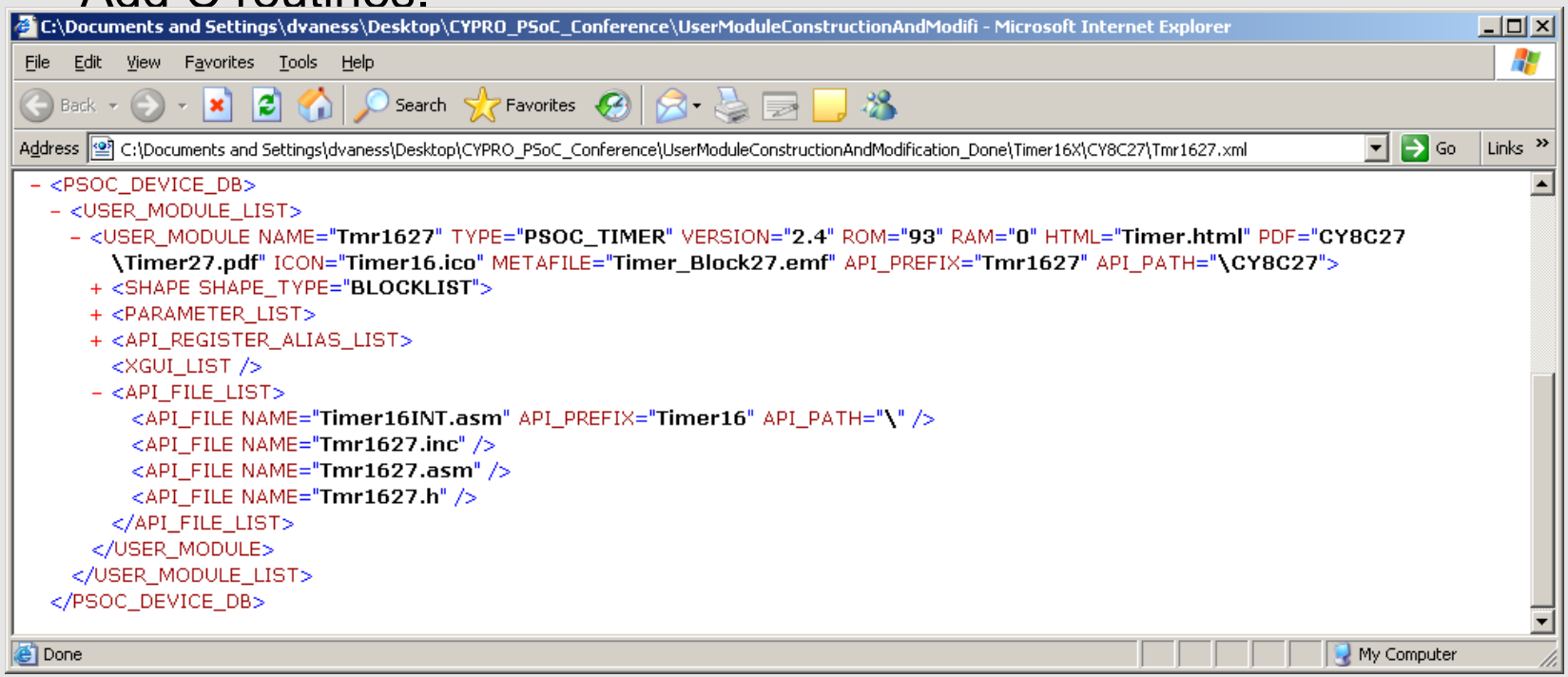


CYPRESS PERFORM

User Modules

API_FILE

- Allow you to define name and location of all API files.
  - Is possible to add multiple assembly files
  - Add C routines.



User Modules

To modify this UM

- Lower Block must be set to also be an interrupt source in the SHAPE section.
- The parameter used to select 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:
  - Terminal Count for the Upper Block
  - Capture for the Lower Block

User Modules

Step 4

This user module will not work for 25x and 26x family of parts. So

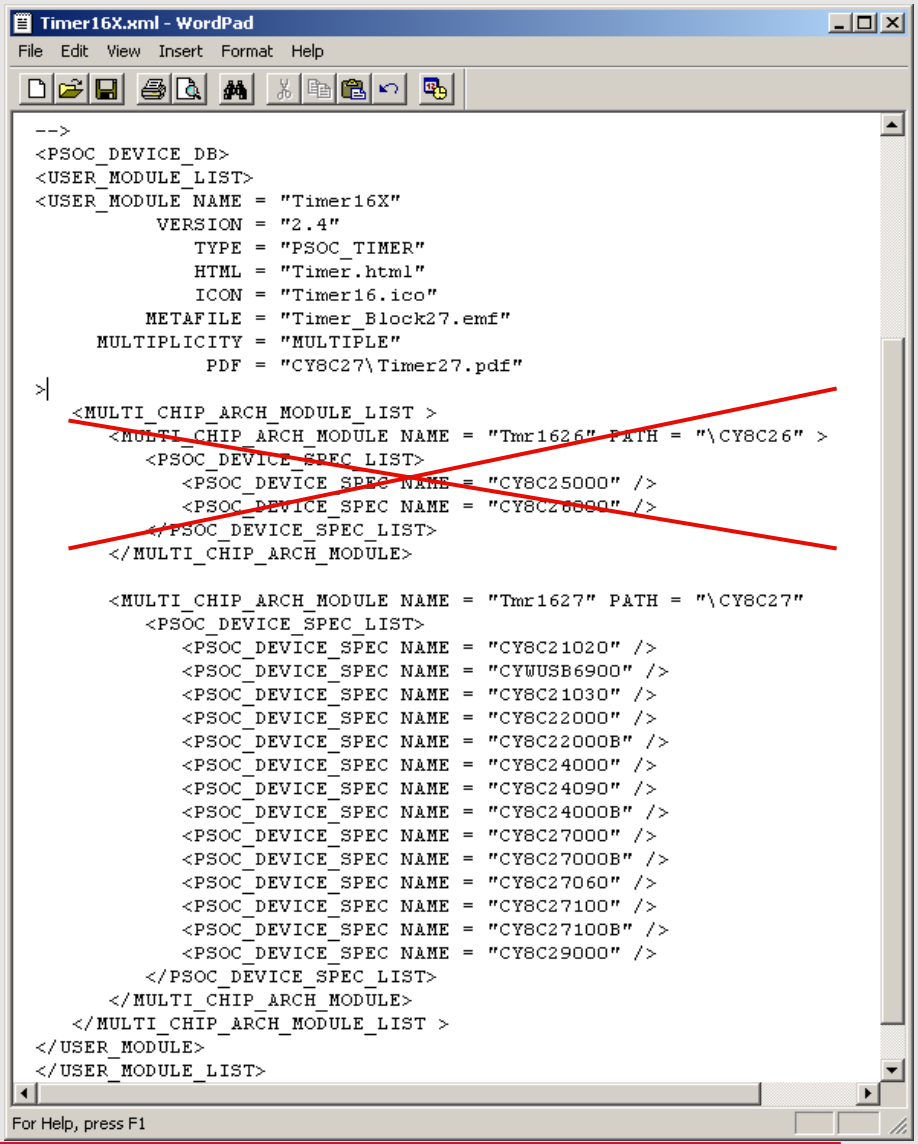
- Open **Timer16X.xml**

  Remove

```
<MULTI_CHIP_ARCH_MODULE NAME = "Tmr1626" PATH = "\CY8C26" >
   <PSOC_DEVICE_SPEC_LIST>
      <PSOC_DEVICE_SPEC NAME = "CY8C25000" />
      <PSOC_DEVICE_SPEC NAME = "CY8C26000" />
   </PSOC_DEVICE_SPEC_LIST>
 </MULTI_CHIP_ARCH_MODULE>
```

  CRET

- The CY8C26 directory is no longer needed.
  - Remove it!

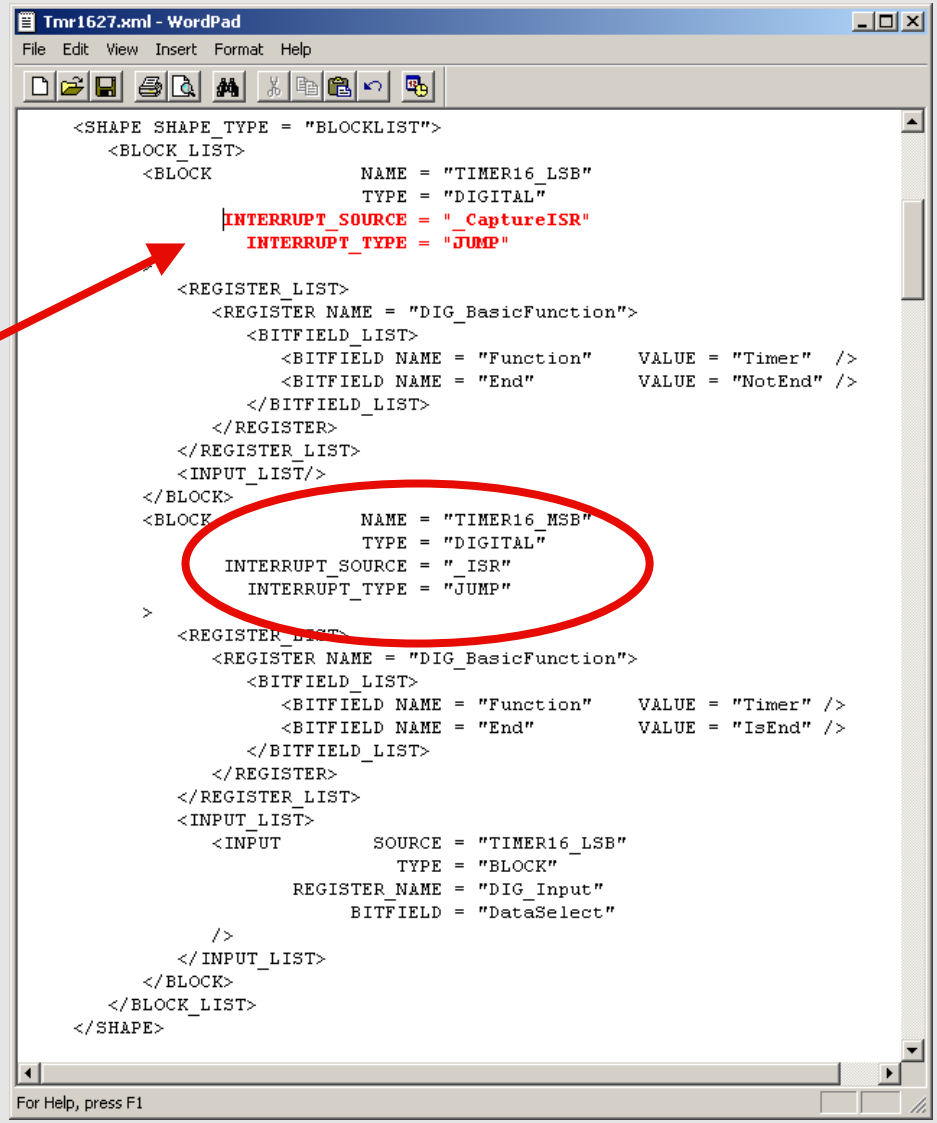


User Modules

Step 5 *Add Interrupt*

- Open **Tmr1627.xml**
- Use interrupt code from **TIMER16_MSB** block to add an interrupt to Timer16LSB block.
  - Name this new interrupt source **_CaptureISR**

CRET



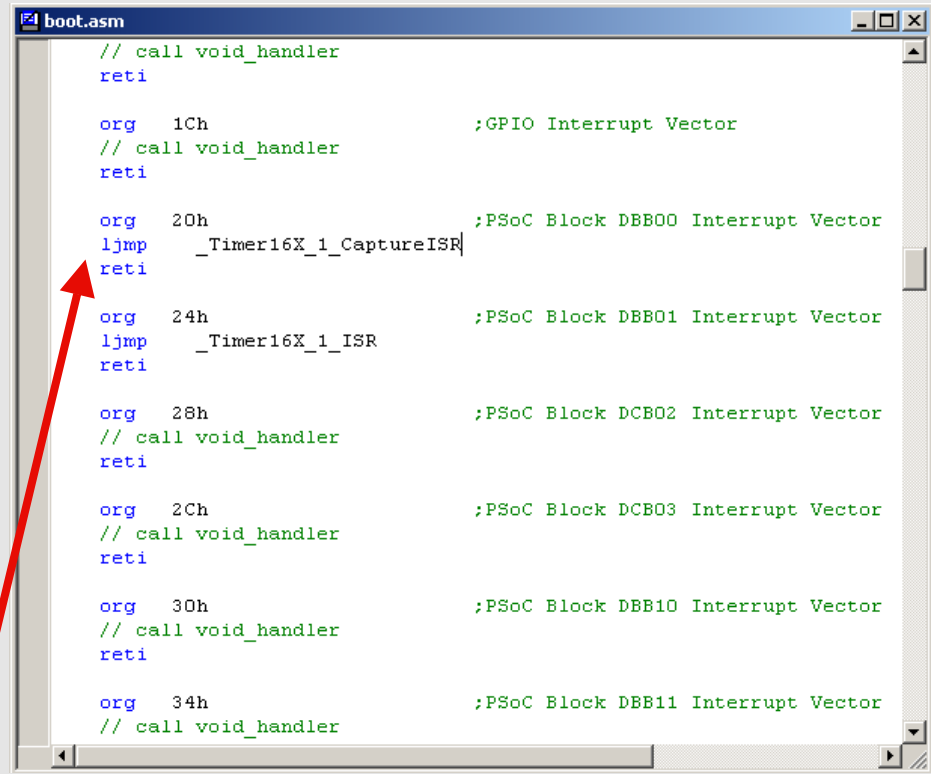
PERFORM

User Modules

Sanity Check!

Open a 27x project with Designer

- Place Timer16X
- Generate application.
- Go to application editor
  - Open boot.asm
- Verify that the new interrupt vector has been added.

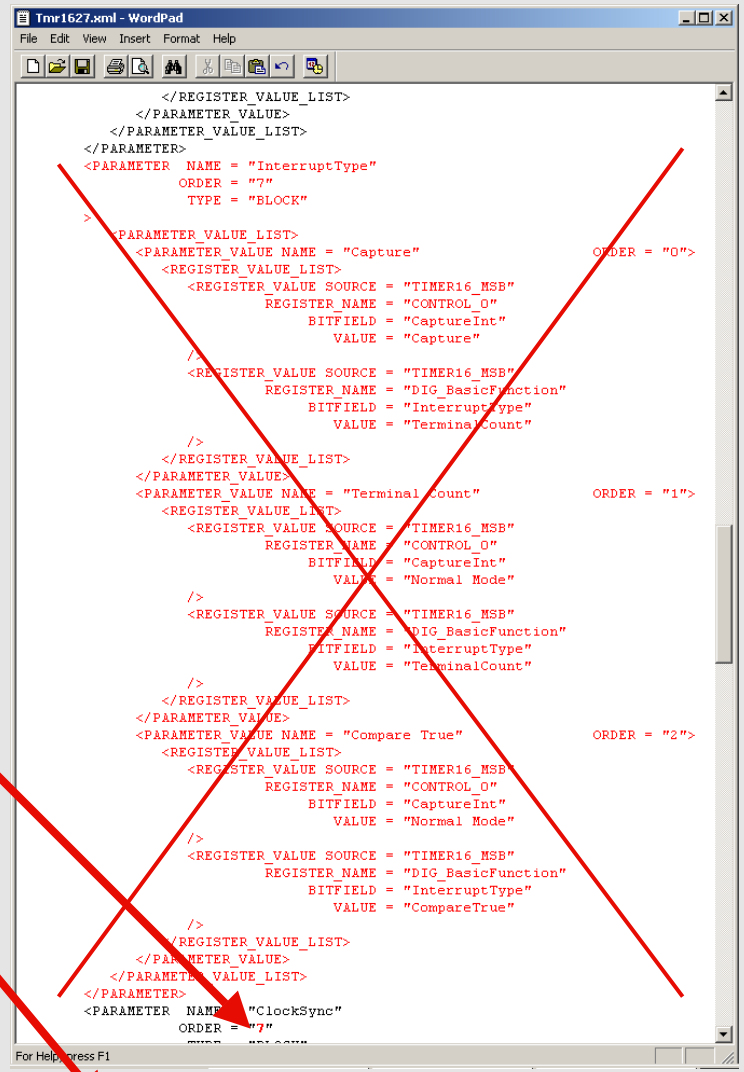


User Modules

Step 6 *Remove Interrupt Type Parameter*

- Open **Tmr1627.xml**
- Remove the **InterruptType** parameter.
  - Save removed portion in a temporary file.
- For **ClockSync** parameter
  - Change order from 8 to 7
- For **TC_Width** parameter
  - Change order to from 9 to 8

CRET



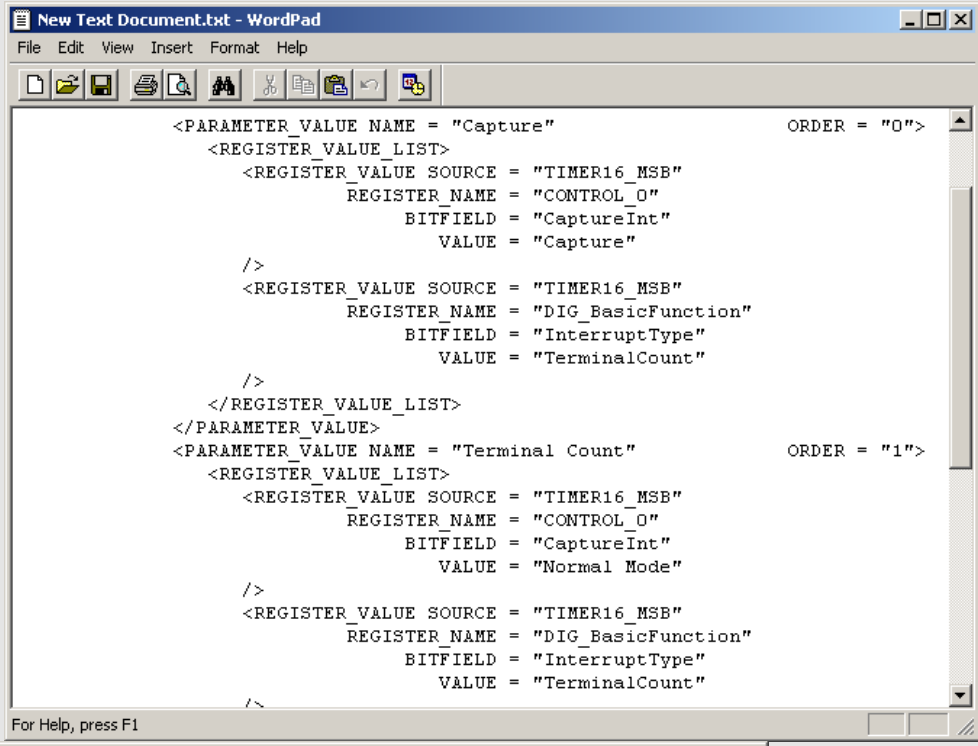
PERFORM

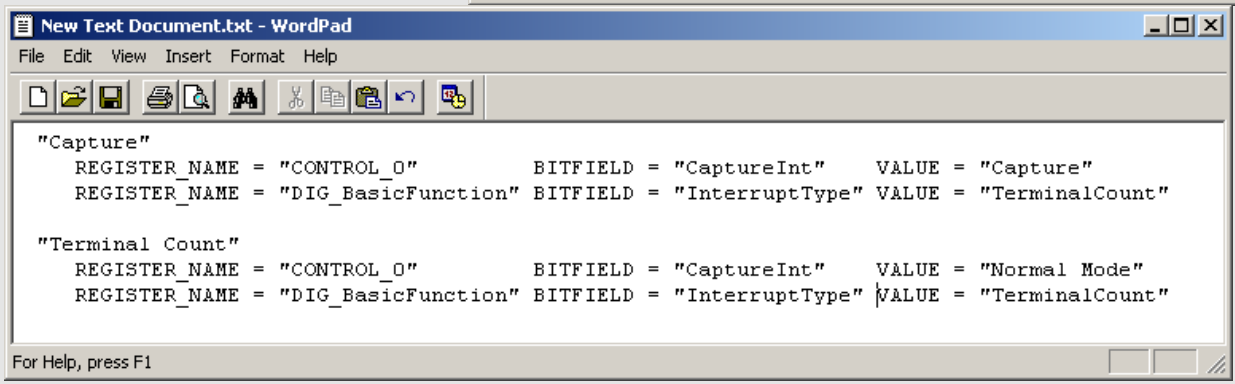
User Modules

Step 7 *Add Interrupts to SHAPE*

Open temporary file.

- The **REGISTER_NAME**, **BITFIELD**, and **VALUE** for each type of interrupt can be used to define the personalize the requires registers.





CYPRESS PERFORM

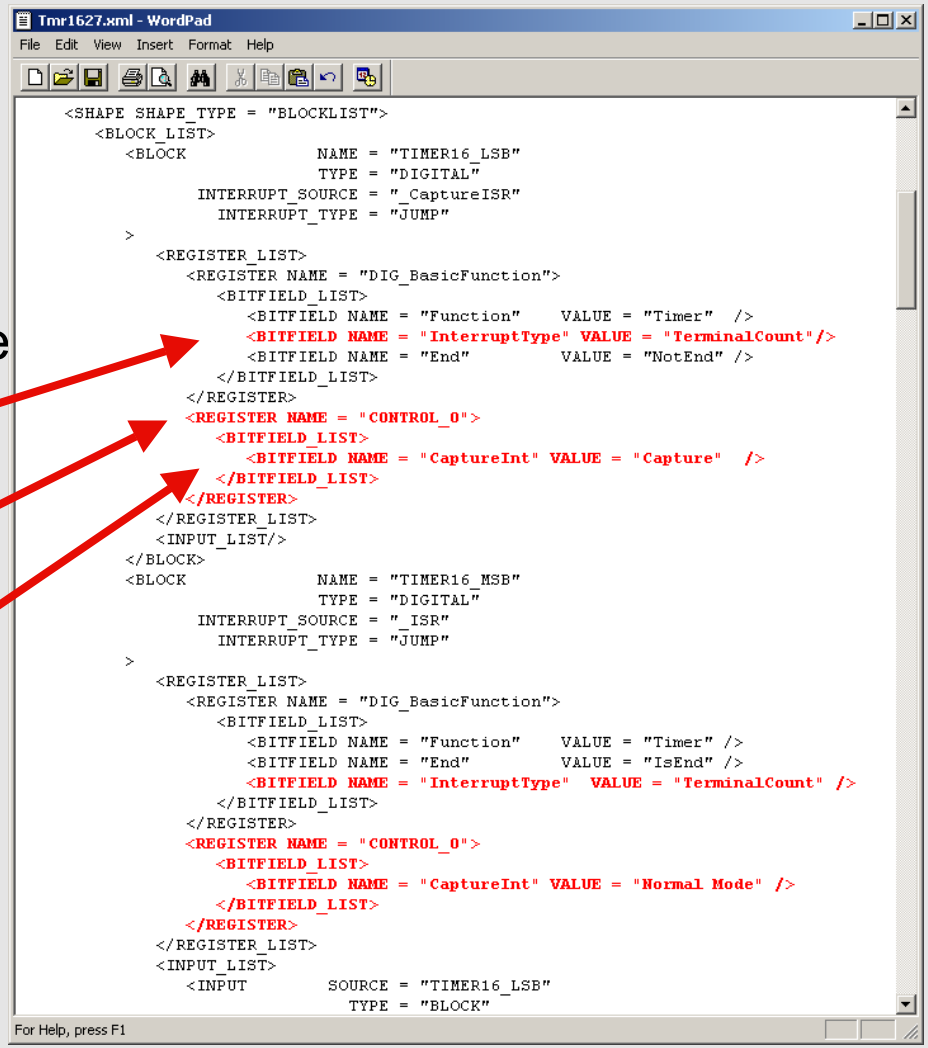
User Modules

Step 7b *Add Interrupts to SHAPE*

- Open **Tmr1627.XML**

For TIMER16_LSB

- Add **“InterruptType”** bit field to the **“DIG_BasicFunction”** register.
  - Set its value to **“TerminalCount”**
- Add **CONTROL_0** register
- Add **“CaptureInt”** bit field to it.
  - Set its value to **“Capture”.**



PERFORM

User Modules

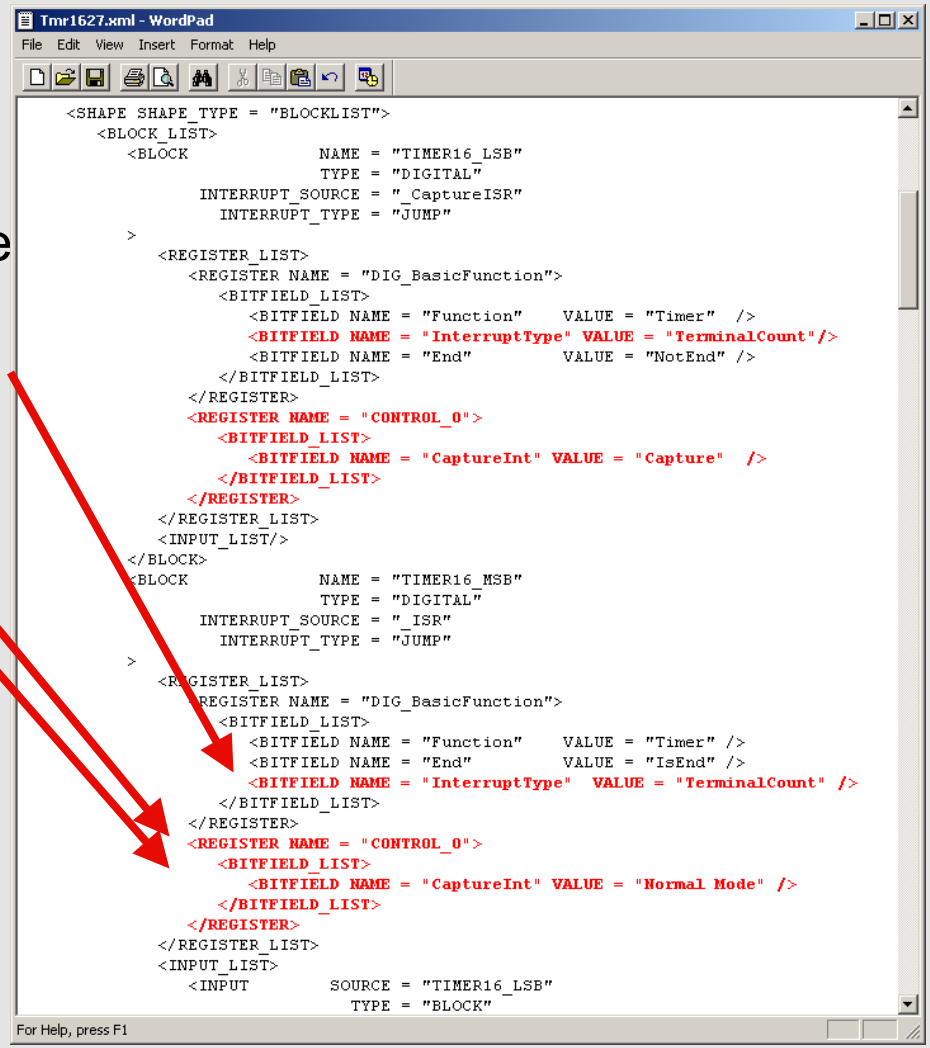
Step 7c *Add Interrupts to SHAPE*

For TIMER16_MSB

- Add **“InterruptType”** bit field to the **“DIG_BasicFunction”** register.
  - Set its value to **“TerminalCount”**
- Add **CONTROL_0** register
- Add **“CaptureInt”** bit field to it.
  - Set its value to **“Normal Mode”**

CRET

All XML changes are done!



User Modules

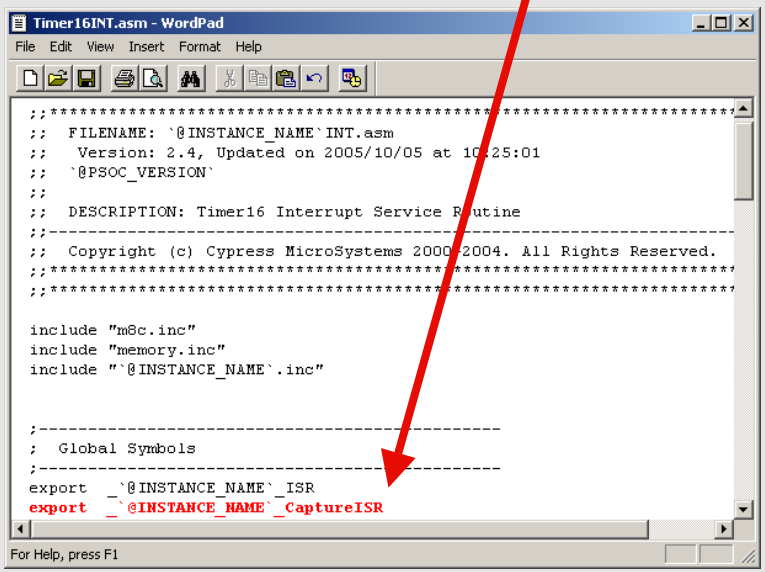
To modify software for this UM

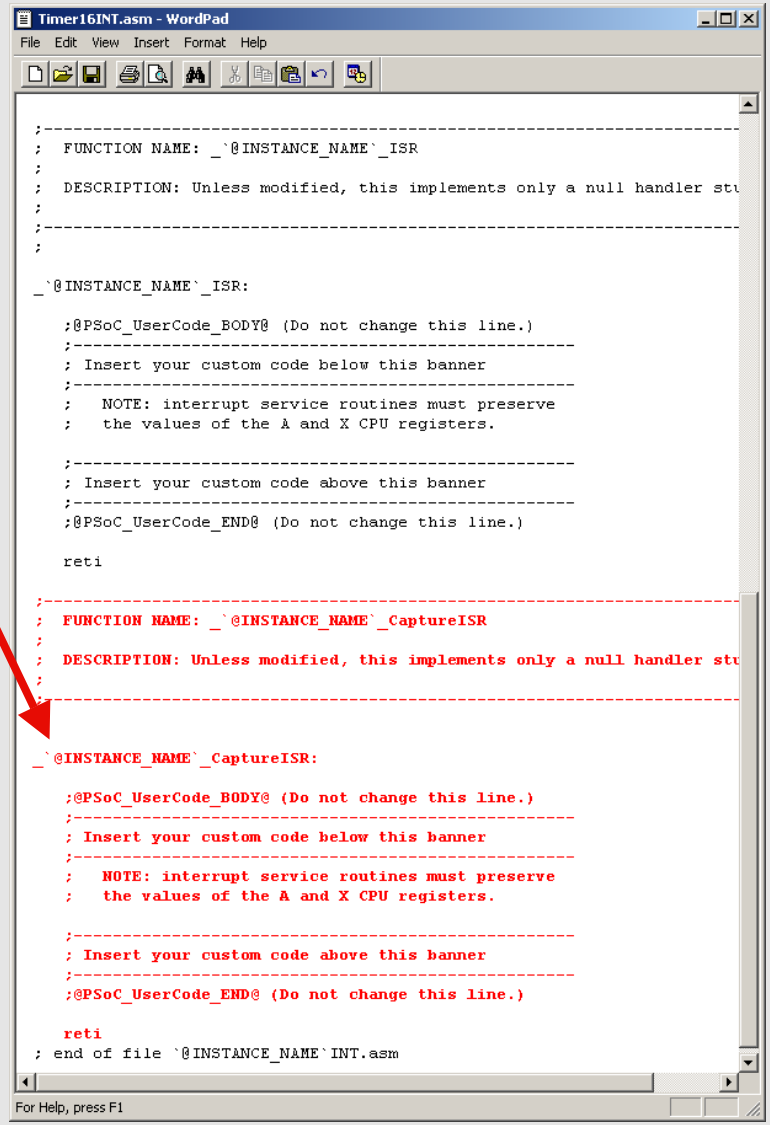
- A new interrupt handler must be added for the new interrupt.
- API must be changed to pass the intended interrupt to enable or disable type .
- An new interrupt mask must be generated for this interrupt.
  - For both C (.h) and assembly (.inc)
- Placement file must be change for 29x parts so both interrupt masks are always in the same interrupt register.

User Modules

Step 8 *Add interrupt handler.*

- Back up a directory
- Open **Timer16INT.asm**
- Duplicate the **_ISR** function and rename to **_CaptureISR**
- Add export for new Label





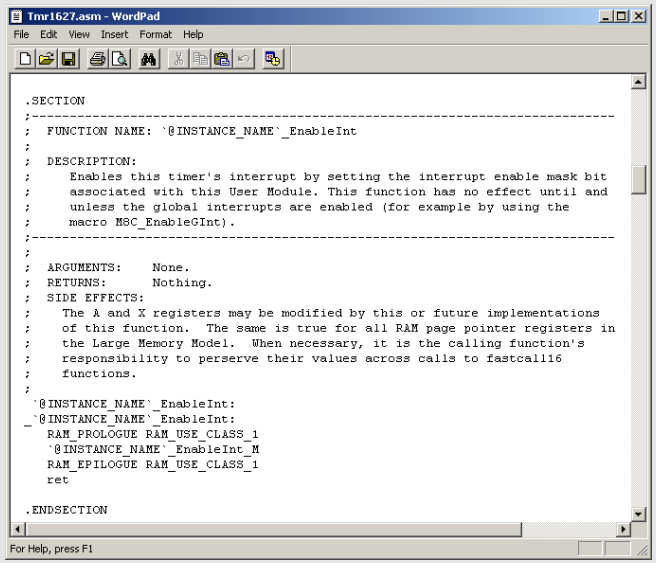
User Modules

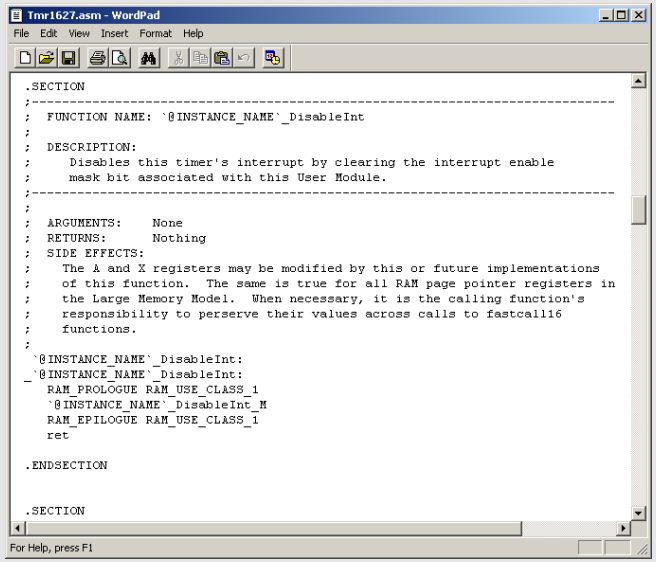
Step 9 *Change APIs*

- Open **CY8C27**
- Open **Tmr1627.asm**

Both APIs are macros. These will both changed when Tmr1627.inc is changed.

Although not needed for this particular example. This step is still shown. Many UMs you develop will require some assembly code alteration.

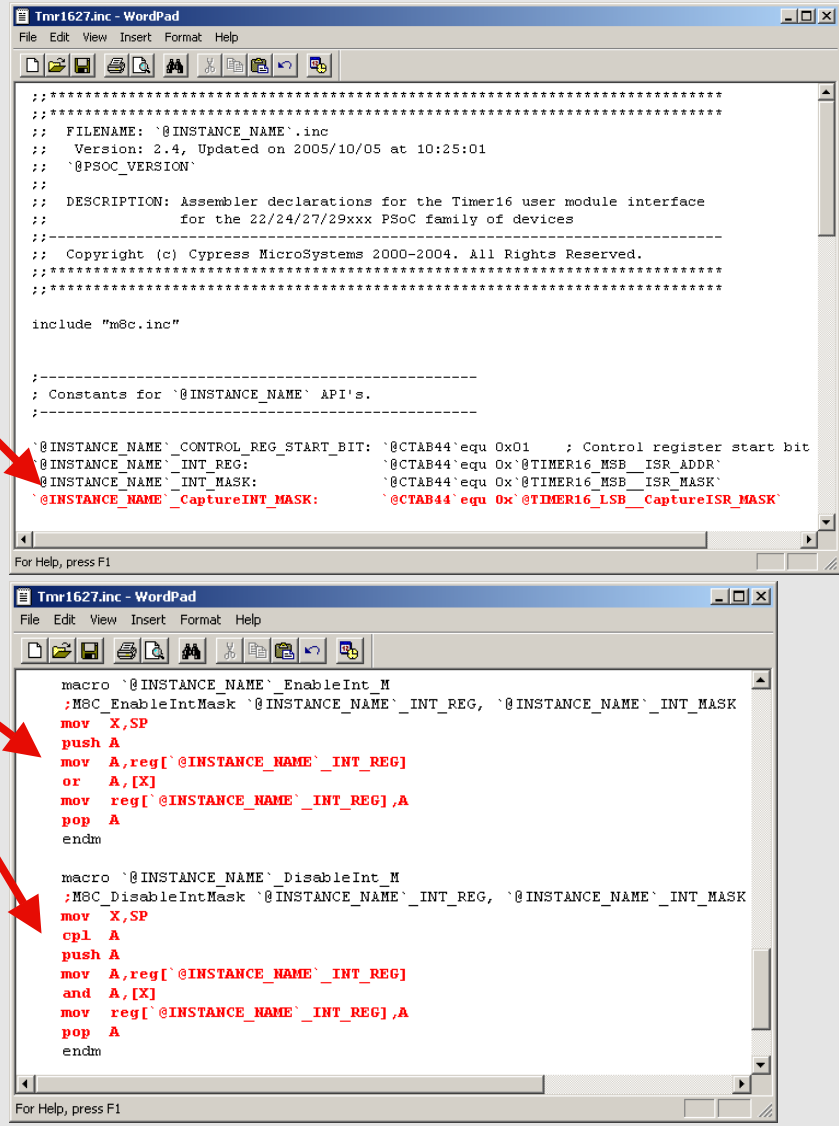




User Modules

Step 10 *Alter include file.*

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



User Modules

Step 11 *Alter .h file.*

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

Tmr1627.h - WordPad

File Edit View Insert Format Help

```
//-------------------------------------------------
// Prototypes of the `@INSTANCE_NAME` API.
//-------------------------------------------------

//extern void `@INSTANCE_NAME`_EnableInt(void);                          // Proxy 1
extern void `@INSTANCE_NAME`_EnableInt(BYTE bMask);                     // Proxy 1
//extern void `@INSTANCE_NAME`_DisableInt(void);                         // Proxy 1
extern void `@INSTANCE_NAME`_DisableInt(BYTE bMask);                    // Proxy 1
extern void `@INSTANCE_NAME`_Start(void);                               // Proxy 1
extern void `@INSTANCE_NAME`_Stop(void);                                // Proxy 1
extern WORD `@INSTANCE_NAME`_wReadTimer(void);                          // Proxy 1
extern WORD `@INSTANCE_NAME`_wReadTimerSaveCV(void);                    // Proxy 2
extern void `@INSTANCE_NAME`_WritePeriod(WORD wPeriod);                 // Proxy 1
extern WORD `@INSTANCE_NAME`_wReadCompareValue(void);                   // Proxy 1
extern void `@INSTANCE_NAME`_WriteCompareValue(WORD wCompareValue);     // Proxy 1

// The following functions are deprecated.
// They may be omitted in future releases
//
extern WORD w`@INSTANCE_NAME`_ReadCompareValue(void);       // Deprecated
extern WORD w`@INSTANCE_NAME`_ReadTimerSaveCV(void);        // Deprecated
extern WORD w`@INSTANCE_NAME`_ReadCounter(void);            // Obsolete
extern WORD w`@INSTANCE_NAME`_ReadTimer(void);              // Deprecated
extern WORD w`@INSTANCE_NAME`_CaptureCounter(void);         // Obsolete


//-------------------------------------------------
// Constants for `@INSTANCE_NAME` API's.
//-------------------------------------------------

#define `@INSTANCE_NAME`_CONTROL_REG_START_BIT `@CTAB48`( 0x01 )
#define `@INSTANCE_NAME`_INT_REG_ADDR          `@CTAB48`( 0x`@TIMER16_MSB__ISR_ADDR` )
#define `@INSTANCE_NAME`_INT_MASK              `@CTAB48`( 0x`@TIMER16_MSB__ISR_MASK` )
#define `@INSTANCE_NAME`_CaptureINT_MASK       `@CTAB48`( 0x`@TIMER16_LSB__CaptureISR_MASK` )

//-------------------------------------------------
// Constants for `@INSTANCE_NAME` user defined values
//-------------------------------------------------
```

For Help, press F1

User Modules

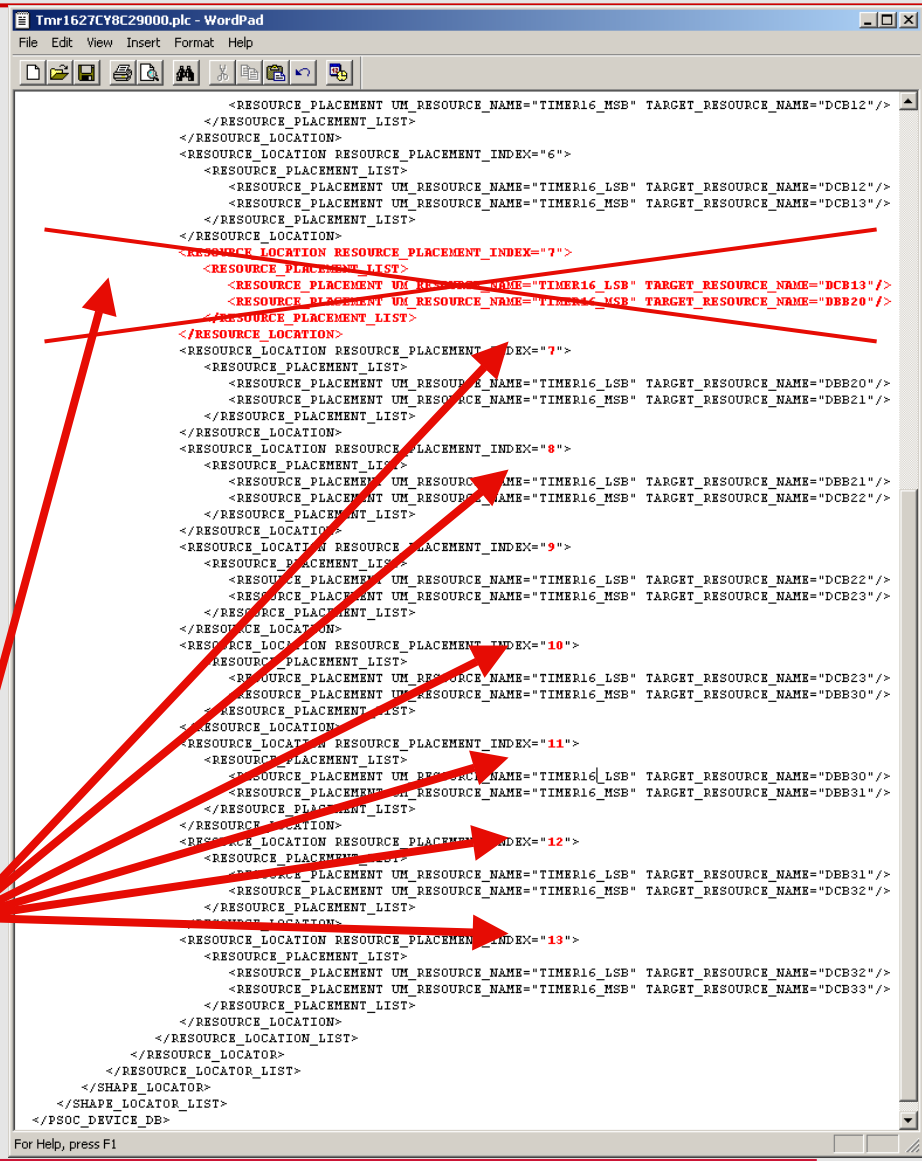
Step 12

Alter 29x placement file

Placement files list all legal placements for a particular chip family.

Only the 29x family allows the placement so the each block has a different interrupt register. ***DCB13 DBB20***

- Open **Tmr1627CY8C2900.plc**
- Remove **RESOURCE_PLACEMENT_INDEX="7"**
- Renumber all following indexes.



PERFORM

User Modules

Final Step.

- Start up Designer for a 29x part
- Verify the correct placement combinations.
- Generate application
- Verify that API compile correctly

Summary

- Each User Module is a combination of information on the intraconnections of PSoC resources, the software to control it, an icon, and a data sheet.
- It is possible to generate new or modify existing ones
- Different UMs can be combined to produce a new UM
- New UMs can be old ones with no change to the hardware. Only changes to the your custom APIs.

Questions