**Freescale Semiconductor**
Demo/Hands-On Lab

Rev. 1.0, 05/2011

# Processor Expert Hands-On Lab

## (Using the Kinetis K60 Tower Board)

*by Jim Trudeau*
*Industrial and Multi-Market Microcontrollers*
*Freescale Semiconductor, Inc.*
*Austin, TX*

In this exercise you will build a "bare metal" (no RTOS) application from scratch, using Processor Expert embedded components, targeting a Freescale K60 board. You will:

- Create a CodeWarrior project that uses Processor Expert
- Add several embedded components to the project (logical device drivers, LDDs)
- Configure the components
- Generate code for those components
- Write some interrupt service routines (ISRs) to complete component functionality
- Download and run the code on a Kinetis K60 Tower System board

**Contents**

# 1 Introduction

To successfully complete this exercise you need the following board and development environment.

- The K60 Tower card, TWR-K60N512
- CodeWarrior for Microcontrollers v 10.1

There is no pre-built CodeWarrior project file. You will create a project from scratch. The only other thing you need to accomplish this exercise successfully is this document.

## 1.1 What Will Happen

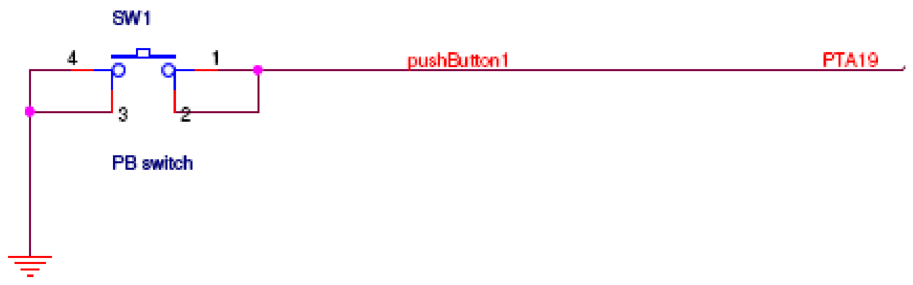
You will create a project, then add and configure embedded components using Processor Expert technology. The Processor Expert tool will create all the initialization code for the drivers. You will instantiate the drivers, and write some simple event handling code.

This demo is a classic "Flash the LED" application. There are four colored LEDs on the board. When complete, this is how the LEDs will behave.

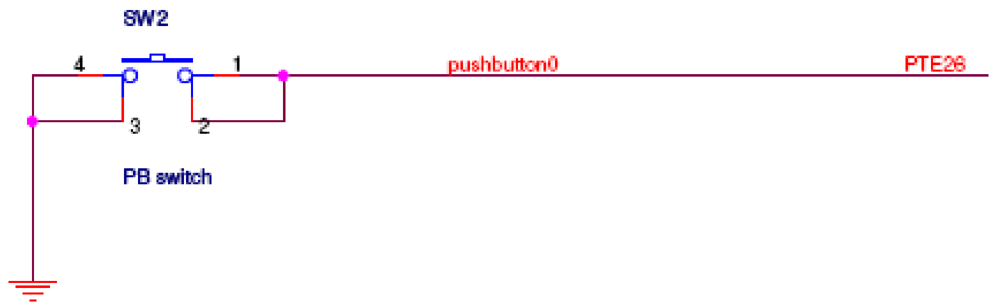
1. The blue LED comes on when code starts, and stays on all the time
2. The green LED blinks at 0.5 seconds intervals, controlled by a timer
3. The yellow LED turns on when you hold down SW2, and turns off when you let go
4. The orange LED turns on and off using SW1 as a toggle.

## 1.2 Hardware Background

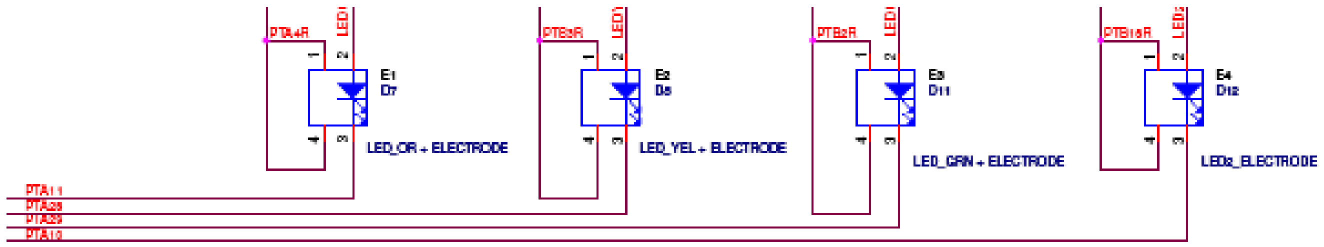
The schematics shows the switches and LED connections.



**SW1**



**SW2**



**LEDs**

This data is encapsulated in the Hardware to Port Mapping table immediately below. Review the table carefully to understand why you configure the components the way you do in this exercise. In the exercise you will use Processor Expert to set up these ports (and pins) to work properly.

**Hardware to Port Mapping**

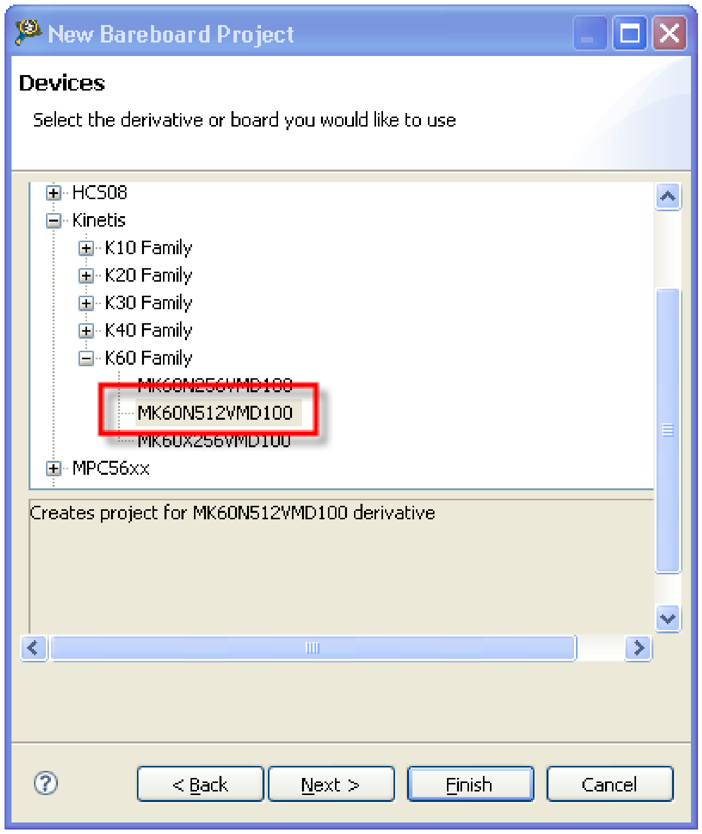
| Hardware | Chip Port/Pin | Comment |
|---|---|---|
| SW1 | PTA19 | Will use as a toggle |
| SW2 | PTE 26 | Will use as “on while pushed” |
| LED 1 (orange) | PTA 11 | |
| LED 2 (yellow) | PTA 28 | |
| LED 3 (green) | PTA 29 | |
| LED 4 (blue) | PTA 10 | |

## 2 Create a project for code development.

Launch the CodeWarrior development environment. To create a new project, use the File menu. Point to File ->New->Bareboard Project. The **New Bareboard Project** dialog appears. We will walk through a series of panels in this Wizard.

First, name the project. We will use the name “K60 PE.” You can name it whatever you want. Click Next.

The Devices panel appears, as shown below.



New Bareboard Project

**Devices**

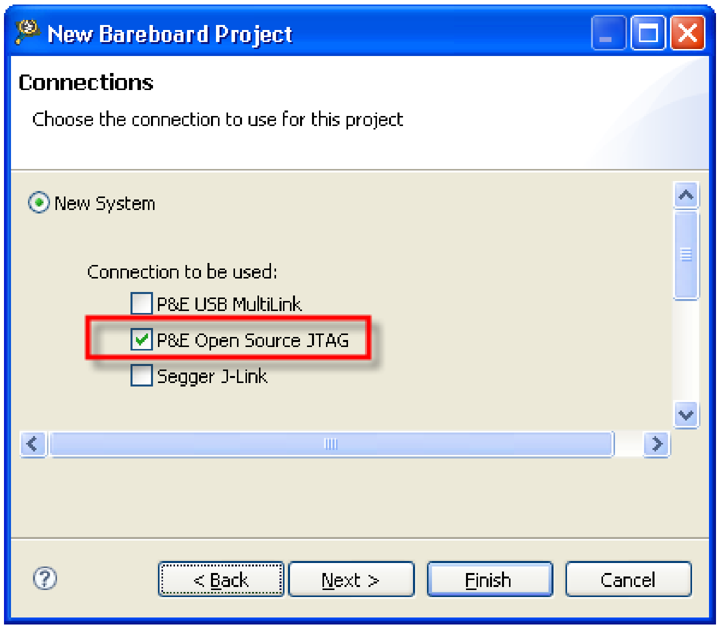
Select the derivative or board you would like to use

```
HCS08
Kinetis
  K10 Family
  K20 Family
  K30 Family
  K40 Family
  K60 Family
    MK60N256VMD100
    MK60N512VMD100
    MK60X256VMD100
MPC56xx
```

Creates project for MK60N512VMD100 derivative

Navigate to the Kinetis K60 family and select MK60N512MD100. Then click Next.

The **Connections** panel appears, as shown below.


New Bareboard Project

Connections

Choose the connection to use for this project

[✓ New System]

Connection to be used:

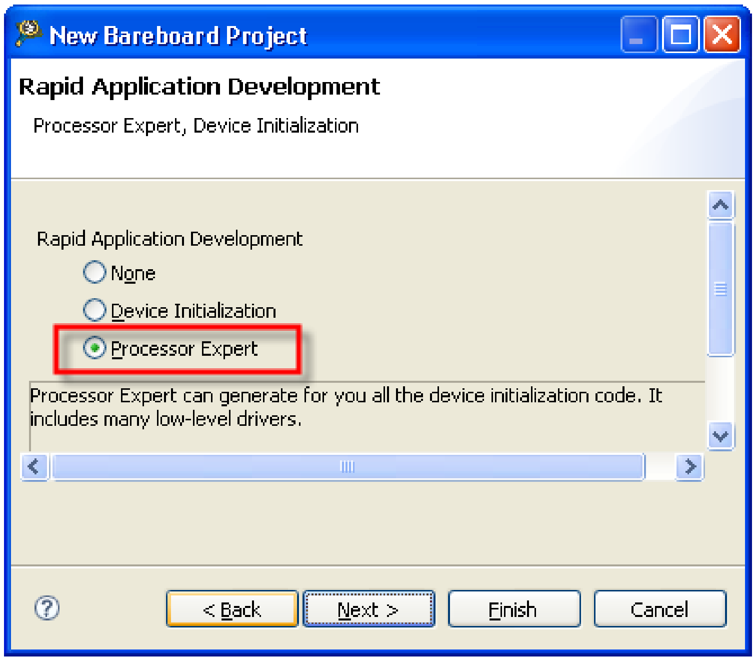
- [ ] P&E USB MultiLink
- [✓] P&E Open Source JTAG
- [ ] Segger J-Link

Choose P&E Open Source JTAG. Then click Next.

The **Add Files** panel appears (not shown). Default values are good, you will copy files into the project, and create a main.c file. Click Next.

The **Languages** panel appears (not shown). Use C. Click Next.

The **Rapid Application Development** panel appears, as shown below.


New Bareboard Project

Rapid Application Development

Processor Expert, Device Initialization

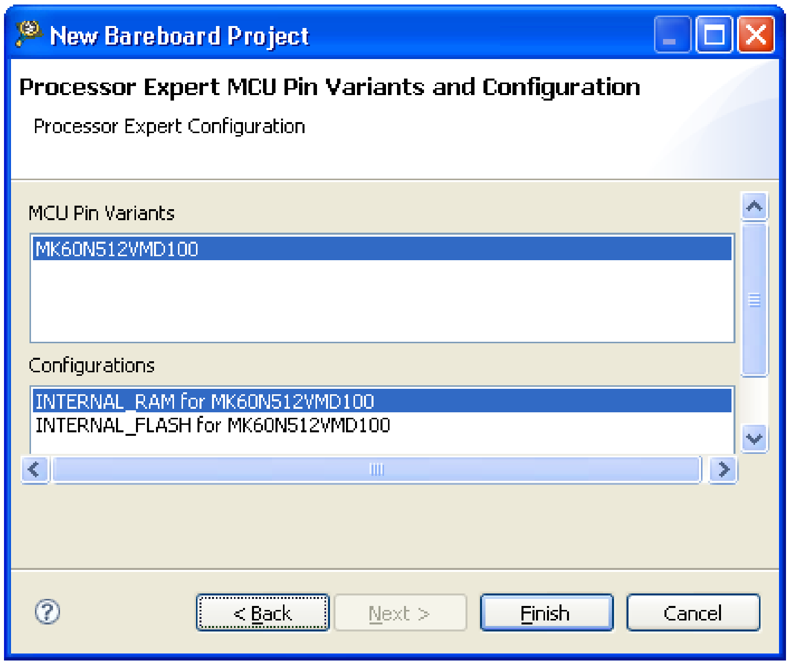
Rapid Application Development

None [✓ Processor Expert] Device Initialization

Processor Expert can generate for you all the device initialization code. It includes many low-level drivers.

Choose Processor Expert. Then click Next.

The **Processor Expert MCU Pin Variants and Configuration** panel appears, as shown below.



Processor Expert MCU Pin Variants and Configuration

Processor Expert Configuration

MCU Pin Variants

MK60N512VMD100

Configurations

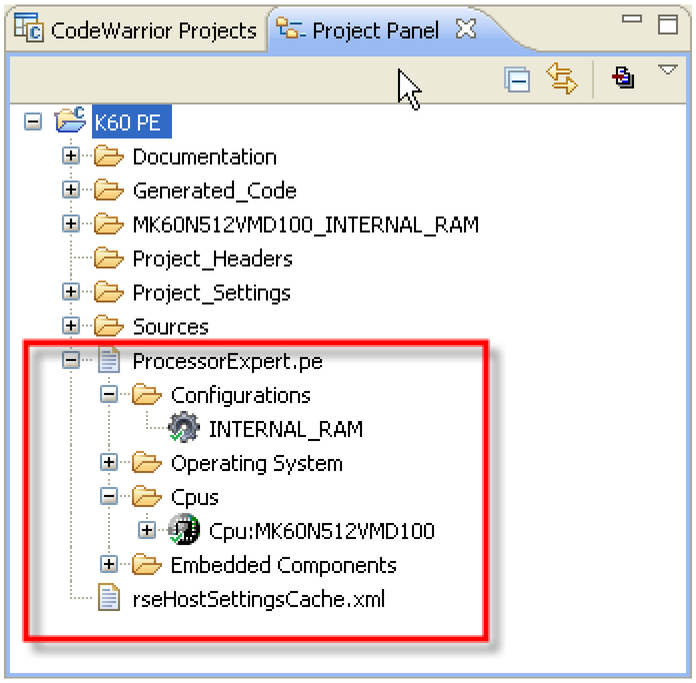
INTERNAL_RAM for MK60N512VMD100
INTERNAL_FLASH for MK60N512VMD100

Select the MK60N512VMD100. Then select the Internal_RAM configuration as shown. In this exercise we will not modify the board's flash.

Click Finish.

Typically a progress dialog will appear, telling you that the tools are creating a Processor Expert project (You may have previously configured CodeWarrior tools to do this in the background.)

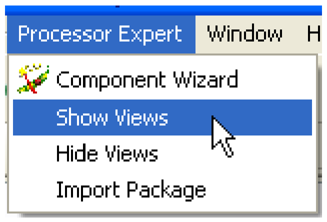
That's it. Your new project appears in the CodeWarrior Project panel of the IDE. Inside is a Processor Expert project as well, named `ProcessorExpert.pe`. If you expand that project, it looks like this:



In subsequent steps you will add embedded components to the Processor Expert project: a timer, and GPIO drivers. You will also generate code, and modify some source files.

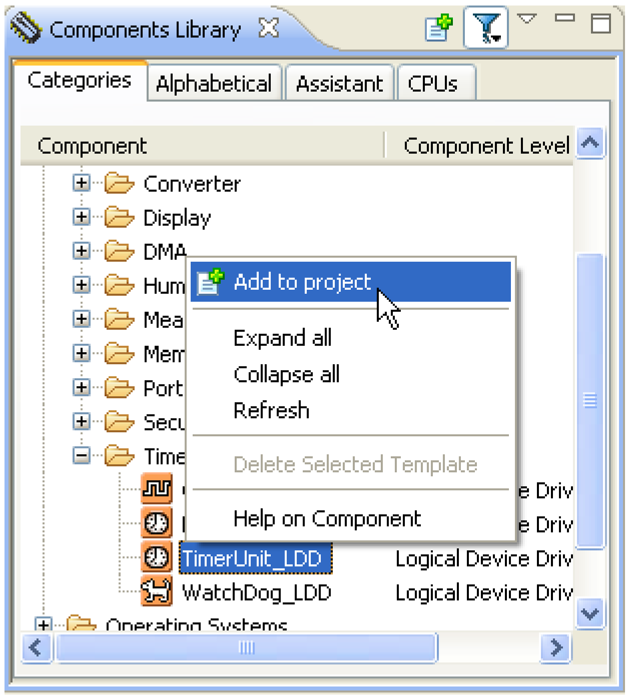
## 3 Add and Configure a Timer

In this and subsequent steps you will use various views related to Processor Expert technology, including the **Components Library** and **Components Inspector**.

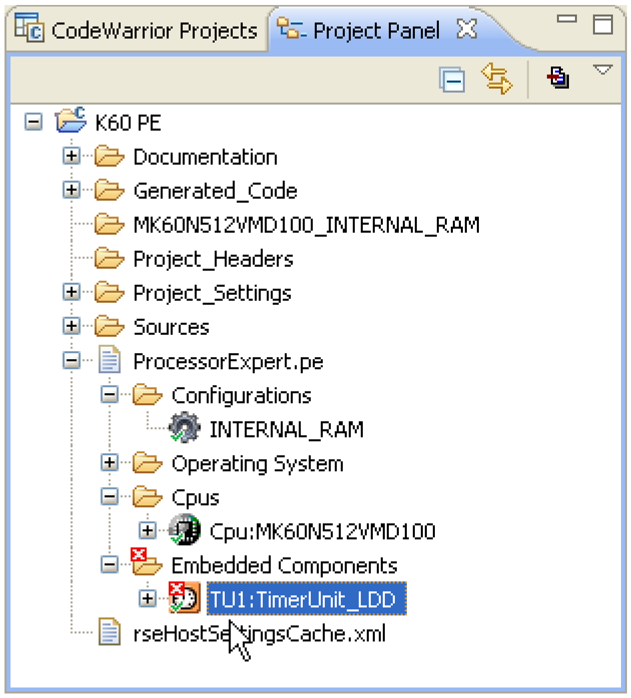


If these views are not visible in the IDE, choose the Show Views item in the Processor Expert menu. The views will appear. You can use this menu to hide the views when you need to.

From the Components Library, add a `TimerUnit_LDD` to the project. There are a variety of ways you can do this. One way is to do as shown in the view below. Select the Categories tab, then navigate to Logical Device Drivers->Timer. Right click on the item and choose Add to Project. See the figure below for guidance.



The Timer driver appears in the Embedded Components section of the Processor Expert part of the project, as shown here.

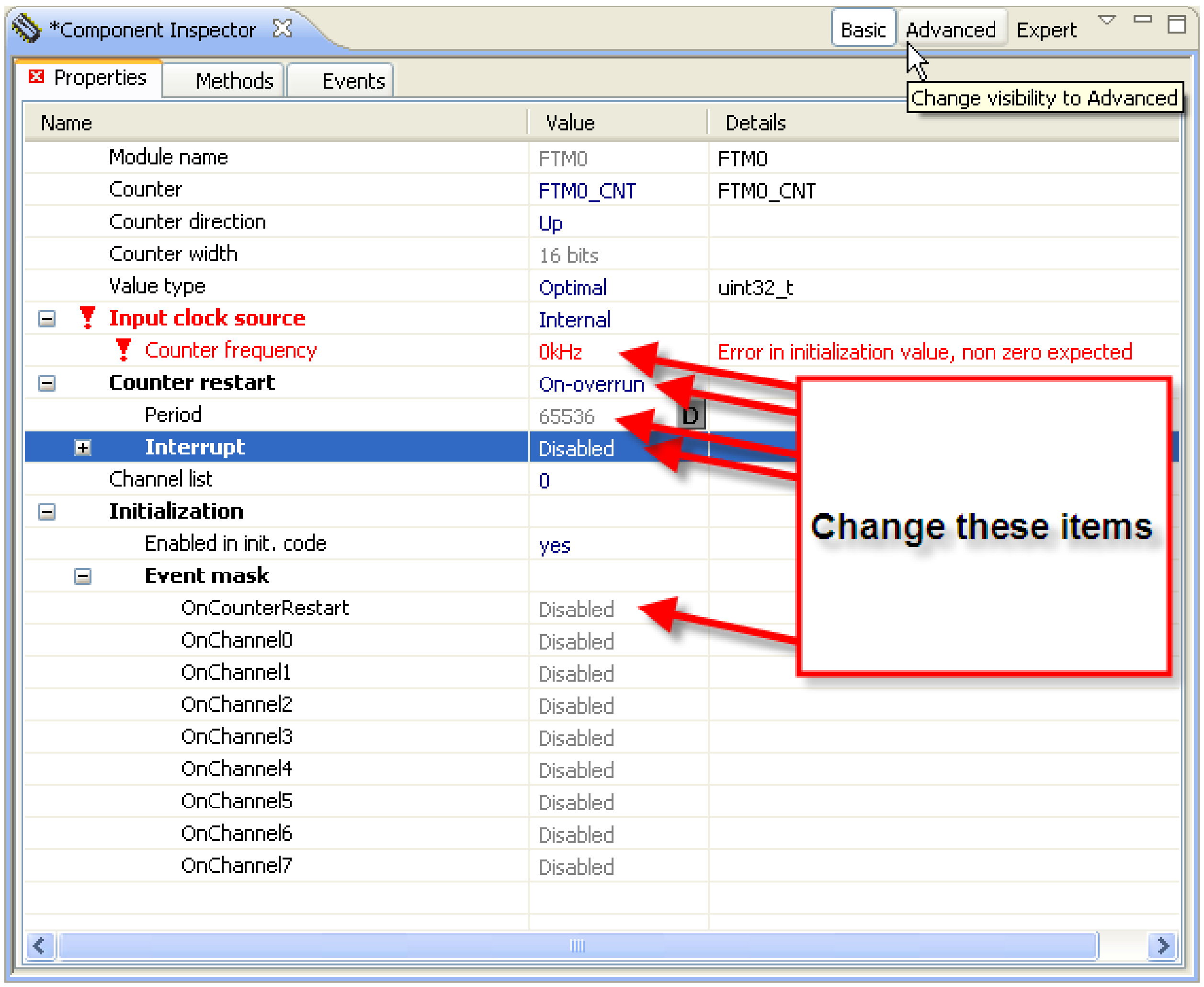


You can now configure this driver. You do this in the Component Inspector, as shown below. Note that you can set the inspector to Basic, Advanced, or Expert level granularity. In this exercise you will work in Basic view.

You will do three things:

1. Enable interrupts for this driver
2. Set the Counter Restart to one second
3. Adjust the counter frequency

Note throughout this process that as you set values, Processor Expert will alert you to problems that need resolution.

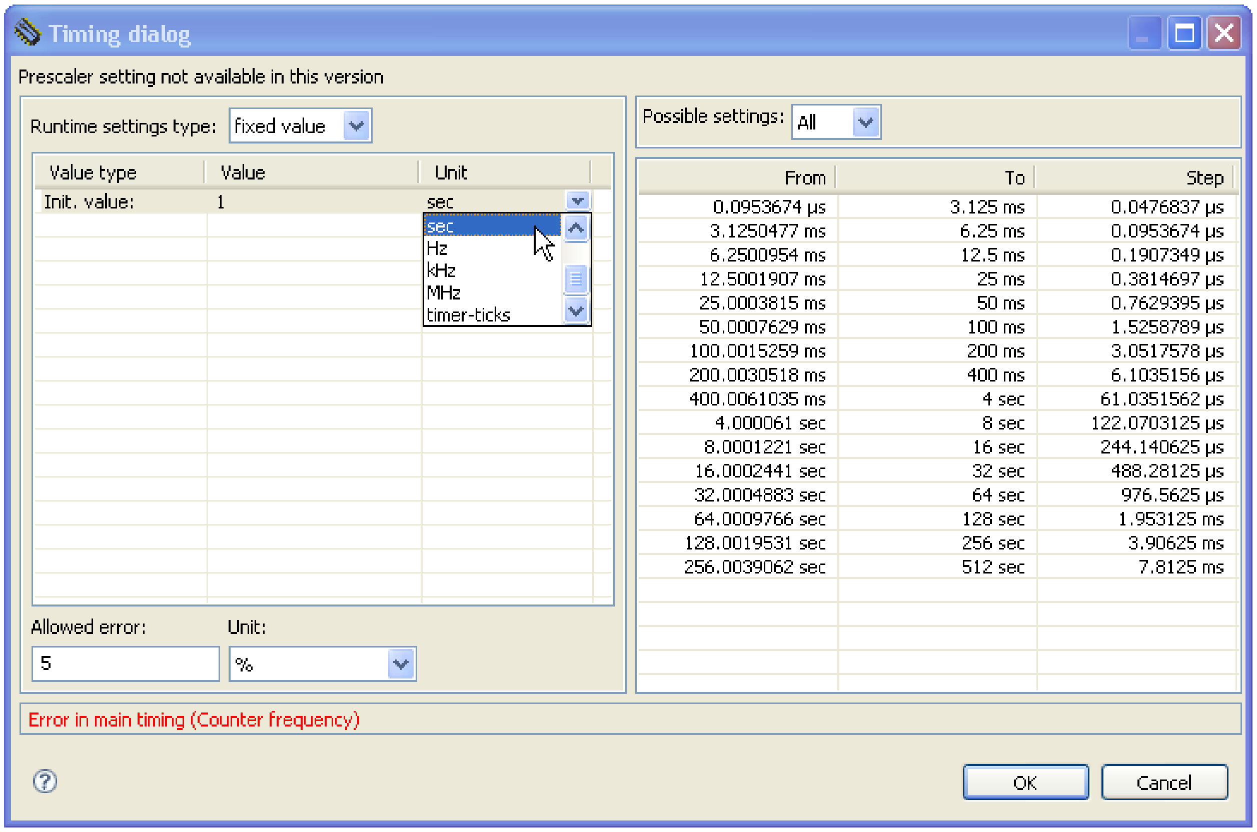


| Name | Value | Details |
|---|---|---|
| Module name | FTM0 | FTM0 |
| Counter | FTM0_CNT | FTM0_CNT |
| Counter direction | Up | |
| Counter width | 16 bits | |
| Value type | Optimal | uint32_t |
| **Input clock source** | Internal | |
| Counter frequency | 0kHz | Error in initialization value, non zero expected |
| **Counter restart** | On-overrun | |
| Period | 65536 | D |
| **Interrupt** | Disabled | |
| Channel list | 0 | |
| **Initialization** | | |
| Enabled in init. code | yes | |
| **Event mask** | | |
| OnCounterRestart | Disabled | |
| OnChannel0 | Disabled | |
| OnChannel1 | Disabled | |
| OnChannel2 | Disabled | |
| OnChannel3 | Disabled | |
| OnChannel4 | Disabled | |
| OnChannel5 | Disabled | |
| OnChannel6 | Disabled | |
| OnChannel7 | Disabled | |

First, set the **Interrupt** property to Enabled. Note that the lines **Initialization**, **Event mask**, and **OnCounterRestart** all turn red if **OnCounterRestart** is not enabled.

If interrupts are enabled, you must have an `OnCounterRestart()` function. So set **OnCounterRestart** to Enabled as well. Processor Expert will create the function stub for you automatically. Later on you will write some code in this function to control an LED. Because this component is named TU1, the name of the function will be `TU1_OnCounterRestart()`.

Set the **Counter restart** property to On-match. Note that the **Period** property becomes red because it now must have a value. The Details column tells you what's wrong. Click inside the Value box, and type 1 sec. Alternatively, once you click in the box to activate data entry, you can click this button: . That will bring up a timing dialog as shown here, where you can set the values.



Prescaler setting not available in this version

Runtime settings type: fixed value

| Value type | Value | Unit |
|---|---|---|
| Init. value: | 1 | sec |

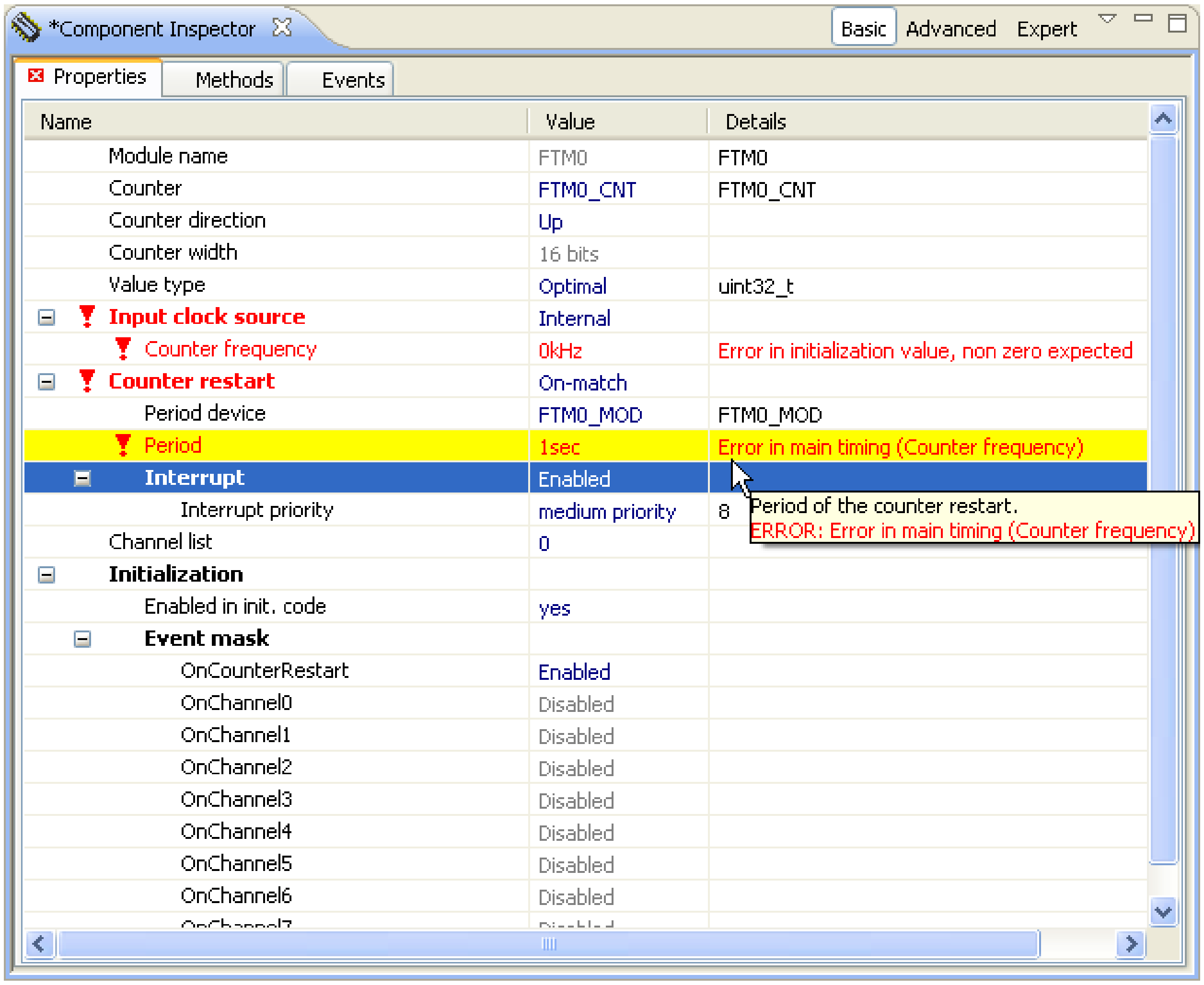
Allowed error: 5 Unit: %

Possible settings: All

| From | To | Step |
|---|---|---|
| 0.0953674 µs | 3.125 ms | 0.0476837 µs |
| 3.1250477 ms | 6.25 ms | 0.0953674 µs |
| 6.2500954 ms | 12.5 ms | 0.1907349 µs |
| 12.5001907 ms | 25 ms | 0.3814697 µs |
| 25.0003815 ms | 50 ms | 0.7629395 µs |
| 50.0007629 ms | 100 ms | 1.5258789 µs |
| 100.0015259 ms | 200 ms | 3.0517578 µs |
| 200.0030518 ms | 400 ms | 6.1035156 µs |
| 400.0061035 ms | 4 sec | 61.0351562 µs |
| 4.000061 sec | 8 sec | 122.0703125 µs |
| 8.0001221 sec | 16 sec | 244.140625 µs |
| 16.0002441 sec | 32 sec | 488.28125 µs |
| 32.0004883 sec | 64 sec | 976.5625 µs |
| 64.0009766 sec | 128 sec | 1.953125 ms |
| 128.0019531 sec | 256 sec | 3.90625 ms |
| 256.0039062 sec | 512 sec | 7.8125 ms |

Error in main timing (Counter frequency)

Note that once you set the period to 1 second, a new problem appears. The counter frequency does not support this period.



*Component Inspector

Basic Advanced Expert

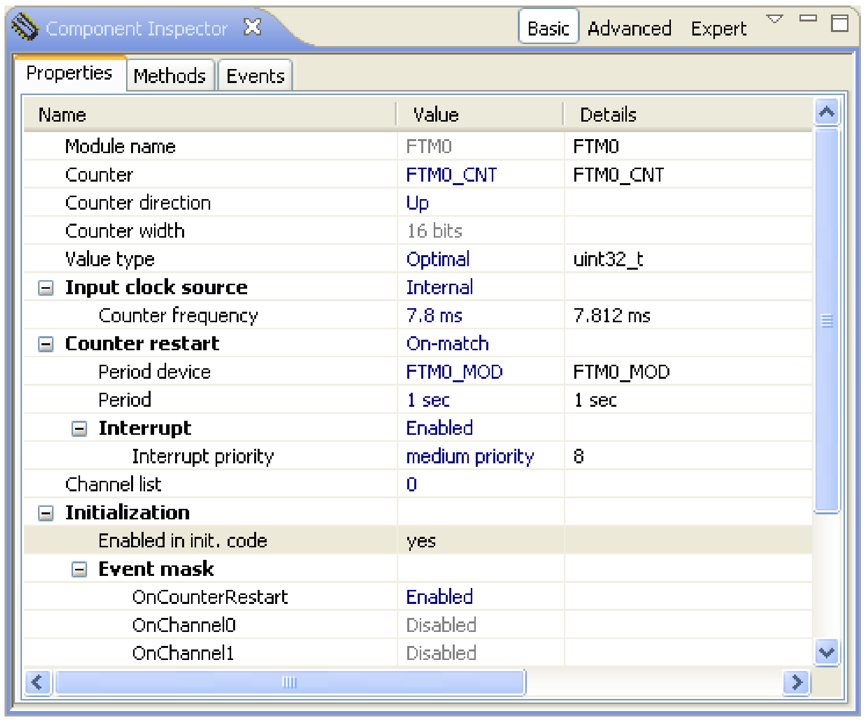
Properties Methods Events

| Name | Value | Details |
|---|---|---|
| Module name | FTM0 | FTM0 |
| Counter | FTM0_CNT | FTM0_CNT |
| Counter direction | Up | |
| Counter width | 16 bits | |
| Value type | Optimal | uint32_t |
| **Input clock source** | Internal | |
| Counter frequency | 0kHz | Error in initialization value, non zero expected |
| **Counter restart** | On-match | |
| Period device | FTM0_MOD | FTM0_MOD |
| Period | 1sec | Error in main timing (Counter frequency) |
| **Interrupt** | Enabled | |
| Interrupt priority | medium priority | 8 |
| Channel list | 0 | |
| **Initialization** | | |
| Enabled in init. code | yes | |
| **Event mask** | | |
| OnCounterRestart | Enabled | |
| OnChannel0 | Disabled | |
| OnChannel1 | Disabled | |
| OnChannel2 | Disabled | |
| OnChannel3 | Disabled | |
| OnChannel4 | Disabled | |
| OnChannel5 | Disabled | |
| OnChannel6 | Disabled | |
| OnChannel7 | Disabled | |

Period of the counter restart.
ERROR: Error in main timing (Counter frequency)

So, set the **Counter frequency** property to a good value. In this case, 7.8 ms works fine. Again, you can either type that directly into the Value field, or use the timing dialog to set the value.

When this step is complete, the Component Inspector will look like this:



| Name | Value | Details |
|---|---|---|
| Module name | FTM0 | FTM0 |
| Counter | FTM0_CNT | FTM0_CNT |
| Counter direction | Up | |
| Counter width | 16 bits | |
| Value type | Optimal | uint32_t |
| **Input clock source** | Internal | |
| Counter frequency | 7.8 ms | 7.812 ms |
| **Counter restart** | On-match | |
| Period device | FTM0_MOD | FTM0_MOD |
| Period | 1 sec | 1 sec |
| **Interrupt** | Enabled | |
| Interrupt priority | medium priority | 8 |
| Channel list | 0 | |
| **Initialization** | | |
| Enabled in init. code | yes | |
| **Event mask** | | |
| OnCounterRestart | Enabled | |
| OnChannel0 | Disabled | |
| OnChannel1 | Disabled | |

For subsequent steps the instructions will be somewhat less comprehensive, now that you are familiar with the Component Inspector and how to set values.

## 4 Add and Configure GPIO1

Go to the Components Library. In the Categories tab, look for Logical Device Drivers->PortIO. Add the `GPIO_LDD` to the project. This is general purpose IO, and you will use it to control the LEDs.

When you add it to the project, it will appear as `GPIO1`.

In the Component Inspector, the **Port** defaults to PTA, and that's what we need.

Set the **Interrupt service/event** value to Enabled.

Set the **Bit Fields** value to 5. When you do, a series of five bit fields will appear in the component inspector.  We need these to control the various LEDs.

Set these properties for the first bit field.

| Property | Value | Note |
|---|---|---|
| Field Name | LED1 | this name will appear in source code, so case matters |
| Pin | PTA11 | this pin controls LED #1 |
| Initial Pin Direction | Output | Sending a signal to the LED |

Set these properties for the second bit field:

| Property | Value | Note |
|---|---|---|
| Field Name | LED2 | this name will appear in source code, so case matters |
| Pin | PTA28 | this pin controls LED #2 |
| Initial Pin Direction | Output | Sending a signal to the LED |

Set these properties for the third bit field:

| Property | Value | Note |
|---|---|---|
| Field Name | LED3 | this name will appear in source code, so case matters |
| Pin | PTA29 | this pin controls LED #3 |
| Initial Pin Direction | Output | Sending a signal to the LED |

Set these properties for the fourth bit field:

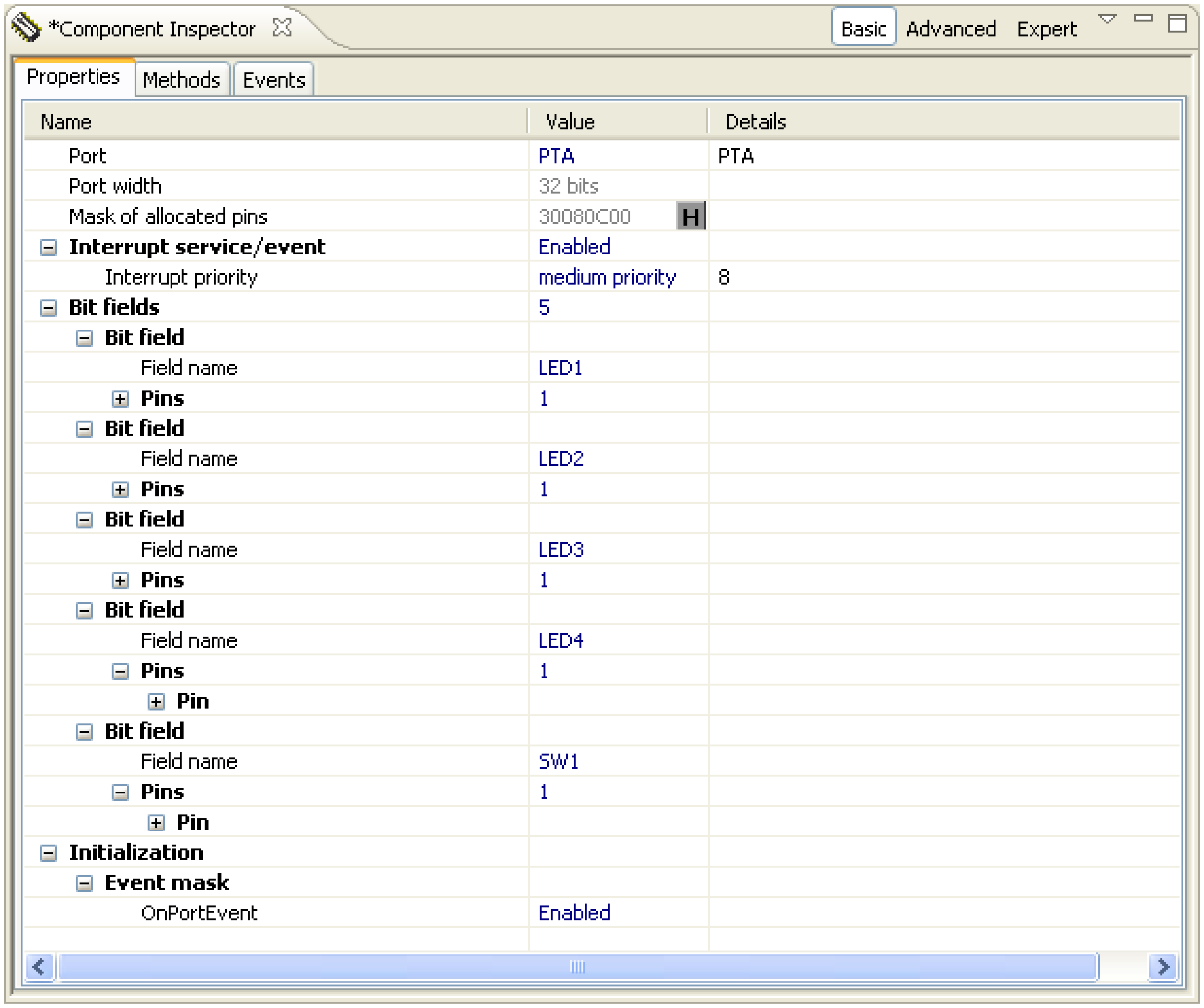
| Property | Value | Note |
|---|---|---|
| Field Name | LED4 | this name will appear in source code, so case matters |
| Pin | PTA10 | this pin controls LED #4 |
| Initial Pin Direction | Output | Sending a signal to the LED |

Set these properties for the fifth and final bit field.

| Property | Value | Note |
|---|---|---|
| Field Name | SW1 | this name will appear in source code, so case matters |
| Pin | PTA19 | this pin is connected to SW1 |
| Initial Pin Direction | Input | |
| Initial Pin Event | Falling Edge | This switch works as a toggle. So all we need is the signal when the SW is “unpressed” |

Finally, at the bottom of the inspector is the Initialization area. Set **OnPortEvent** to Enabled.

In the end, this component looks like this (with some levels collapsed).



*Component Inspector

Basic Advanced Expert

Properties | Methods | Events

| Name | Value | Details |
|---|---|---|
| Port | PTA | PTA |
| Port width | 32 bits | |
| Mask of allocated pins | 30080C00 | |
| ⊟ **Interrupt service/event** | Enabled | |
| Interrupt priority | medium priority | 8 |
| ⊟ **Bit fields** | 5 | |
| ⊟ **Bit field** | | |
| Field name | LED1 | |
| ⊞ **Pins** | 1 | |
| ⊟ **Bit field** | | |
| Field name | LED2 | |
| ⊞ **Pins** | 1 | |
| ⊟ **Bit field** | | |
| Field name | LED3 | |
| ⊞ **Pins** | 1 | |
| ⊟ **Bit field** | | |
| Field name | LED4 | |
| ⊟ **Pins** | 1 | |
| ⊞ **Pin** | | |
| ⊟ **Bit field** | | |
| Field name | SW1 | |
| ⊟ **Pins** | 1 | |
| ⊞ **Pin** | | |
| ⊟ **Initialization** | | |
| ⊟ **Event mask** | | |
| OnPortEvent | Enabled | |

## 5 Add and Configure GPIO2

In this step you add another GPIO component, which you will use to work with SW2.

Go to the Components Library. In the Categories tab, look for Logical Device Drivers->PortIO. Add the `GPIO_LDD` to the project again. When you add it to the project, it will appear as `GPIO2`.

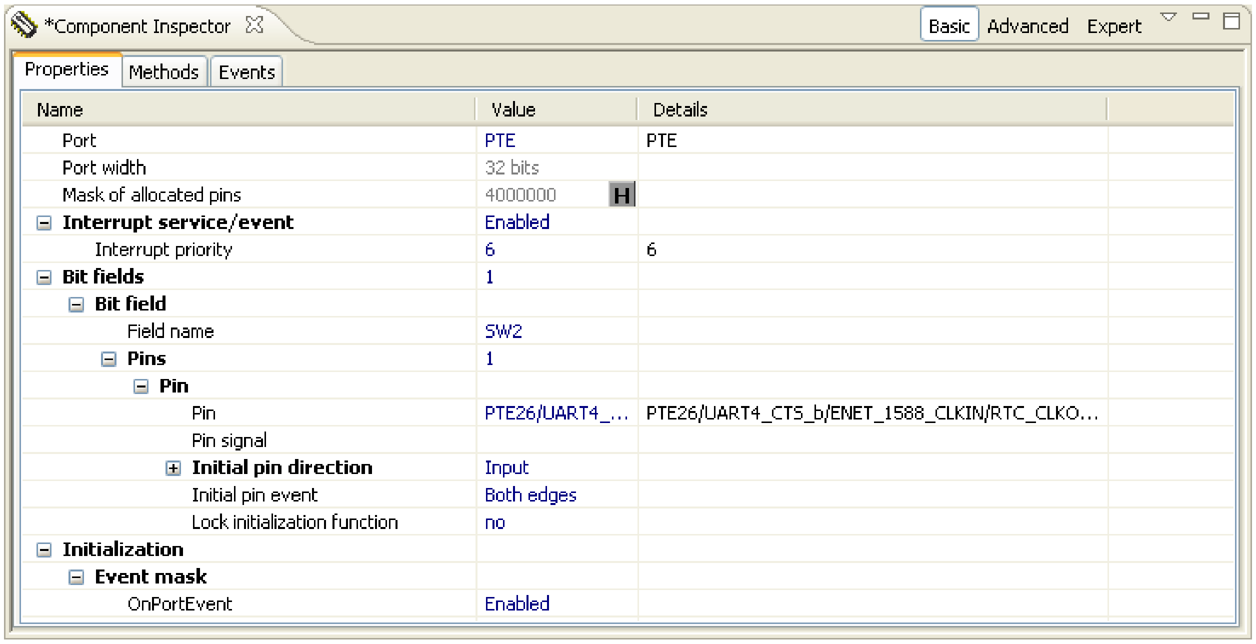
In the Component Inspector, the **Port** defaults to PTA, and in this case *that is not correct*. SW2 is connected to PTE.

In the Component Inspector, set these properties to these values for the GPIO2 component.

| Property | Value | Note |
|---|---|---|
| Port | PTE | Required for SW2 |
| Interrupt service/event | Enabled | You will capture an event when the switch is pressed |
| Interrupt Priority | 6 | You'll handle the switch at a higher priority than other events |
| Bit Fields | 1 | This is the default value |

| | | |
|---|---|---|
| Field Name | SW2 | Will appear in source code, so case matters |
| Pin | PTE26 | Per schematic |
| Initial pin direction | Input | |
| Initial pin event | Both edges | This switch turns on an LED as long as the switch is pushed, so we need both the “pressed” and “unpressed” signals of the same push. |
| OnPortEvent | Enabled | |

When complete, the Component Inspector should look like this:



| Name | Value | Details |
|---|---|---|
| Port | PTE | PTE |
| Port width | 32 bits | |
| Mask of allocated pins | 4000000 | |
| Interrupt service/event | Enabled | |
| Interrupt priority | 6 | 6 |
| Bit fields | 1 | |
| Bit field | | |
| Field name | SW2 | |
| Pins | 1 | |
| Pin | | |
| Pin | PTE26/UART4_... | PTE26/UART4_CTS_b/ENET_1588_CLKIN/RTC_CLKO... |
| Pin signal | | |
| Initial pin direction | Input | |
| Initial pin event | Both edges | |
| Lock initialization function | no | |
| Initialization | | |
| Event mask | | |
| OnPortEvent | Enabled | |

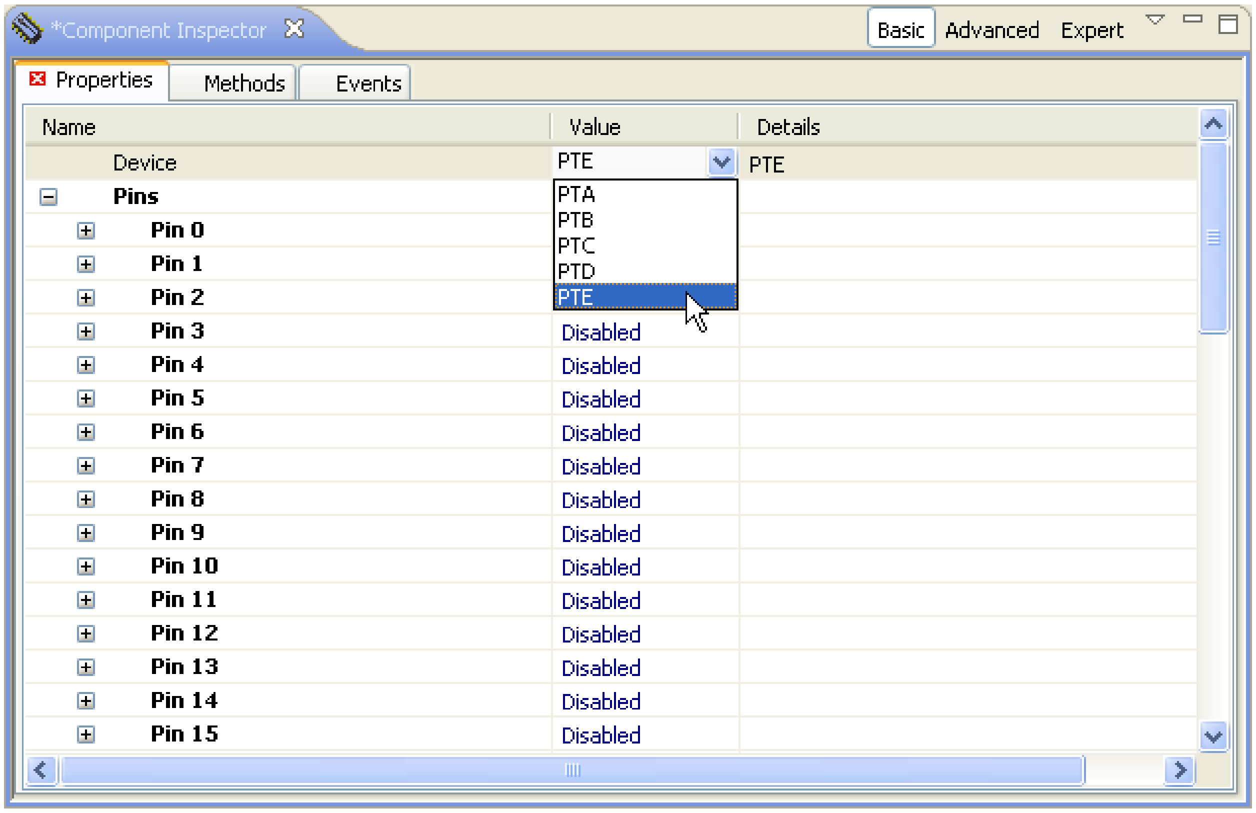
## 6 Add and Configure Init_GPIO

SW2 does not have a pull up on the board, so you will turn on the internal pull up on the port. Because SW2 is connected to PTE, you will use that port.

Go to the Components Library. In the Categories tab, look for CPU Internal Peripherals->Peripheral Initialization. In the list that appears, look for `Init_GPIO`, and add it to the project. When you add it to the project, it will appear as `GPIO3:Init_GPIO`.

Then examine this component in the Component Inspector.

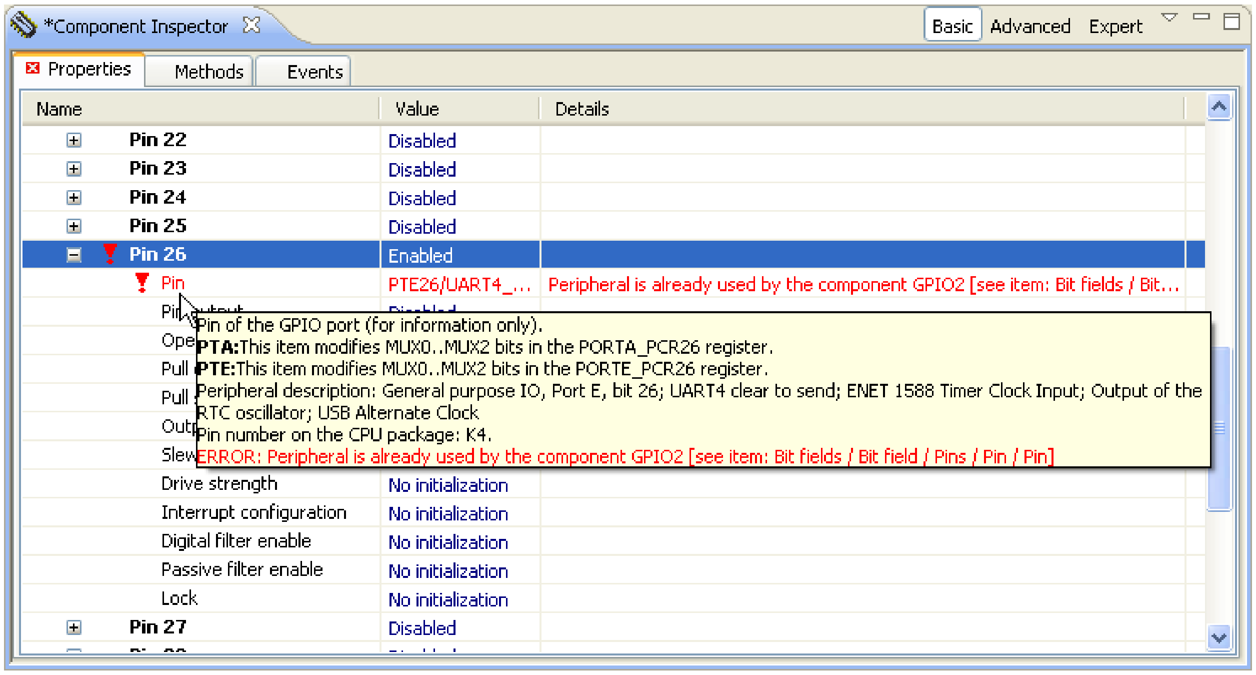
At the very top is the Device property. Set this to PTE, as shown here.



| Name | Value | Details |
|---|---|---|
| Device | PTE | PTE |
| Pins | | |
| Pin 0 | PTA | |
| Pin 1 | PTB / PTC | |
| Pin 2 | PTD / PTE | |
| Pin 3 | Disabled | |
| Pin 4 | Disabled | |
| Pin 5 | Disabled | |
| Pin 6 | Disabled | |
| Pin 7 | Disabled | |
| Pin 8 | Disabled | |
| Pin 9 | Disabled | |
| Pin 10 | Disabled | |
| Pin 11 | Disabled | |
| Pin 12 | Disabled | |
| Pin 13 | Disabled | |
| Pin 14 | Disabled | |
| Pin 15 | Disabled | |

The Inspector lists all the pins associated with this component. You can expand a pin and set various properties for each pin. In this step you will work on **Pin 26**.

Scroll to **Pin 26**, and enable the pin. When you do, a warning appears as shown below. This pin is already in use by GPIO2 which you configured in the previous step.



| Name | Value | Details |
|---|---|---|
| Pin 22 | Disabled | |
| Pin 23 | Disabled | |
| Pin 24 | Disabled | |
| Pin 25 | Disabled | |
| Pin 26 | Enabled | |
| Pin | PTE26/UART4_... | Peripheral is already used by the component GPIO2 [see item: Bit fields / Bit... |
| Drive strength | No initialization | |
| Interrupt configuration | No initialization | |
| Digital filter enable | No initialization | |
| Passive filter enable | No initialization | |
| Lock | No initialization | |
| Pin 27 | Disabled | |

Pin of the GPIO port (for information only).
**PTA:**This item modifies MUX0..MUX2 bits in the PORTA_PCR26 register.
**PTE:**This item modifies MUX0..MUX2 bits in the PORTE_PCR26 register.
Peripheral description: General purpose IO, Port E, bit 26; UART4 clear to send; ENET 1588 Timer Clock Input; Output of the RTC oscillator; USB Alternate Clock
Pin number on the CPU package: K4.
ERROR: Peripheral is already used by the component GPIO2 [see item: Bit fields / Bit field / Pins / Pin / Pin]

Note that the error statement tells you precisely what's going on and where the conflict arises. In this case, you *want* to use the same pin, so this is not an error. You will solve this a little later in this step by enabling pin sharing. For now, continue with pin configuration.

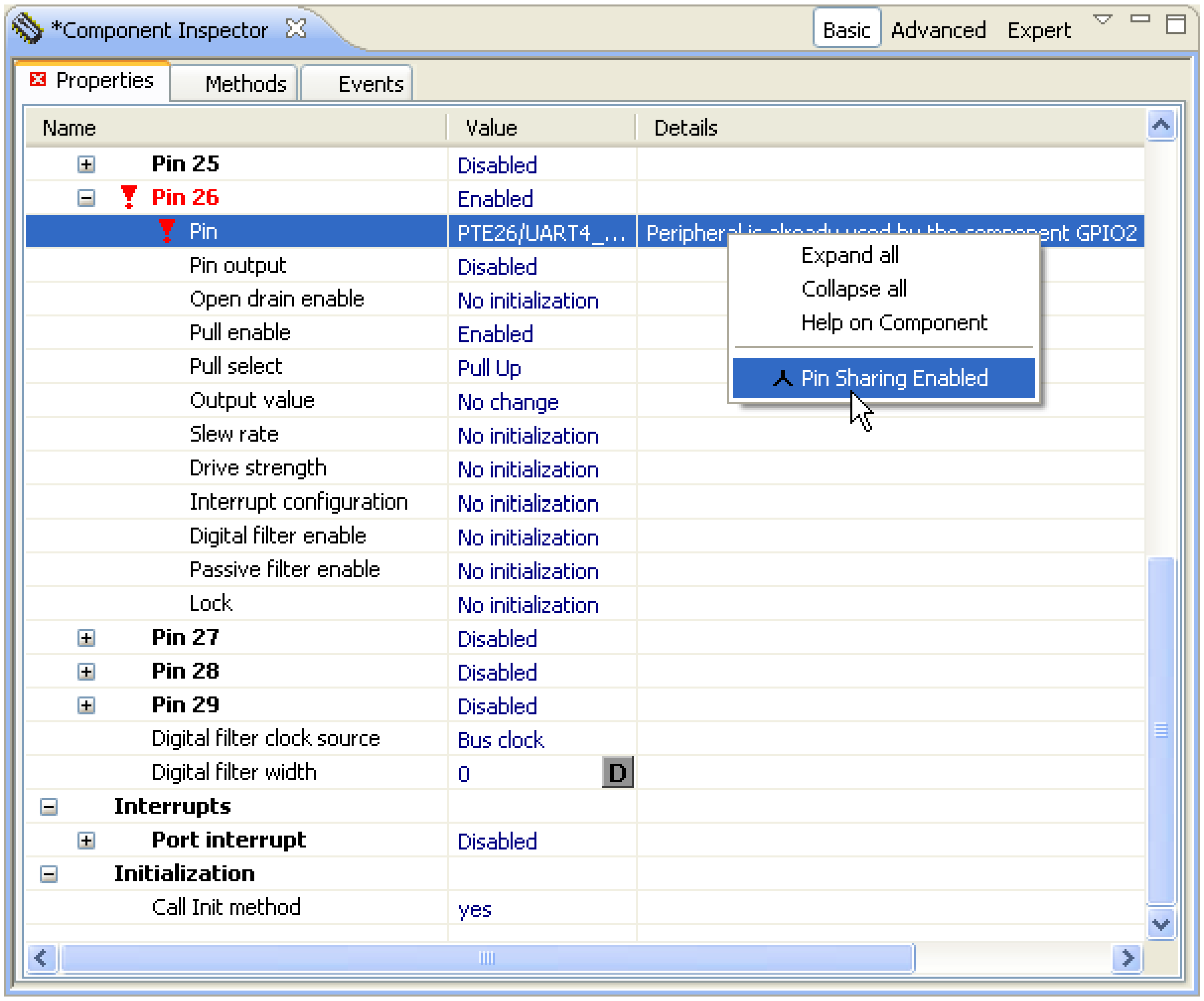
In the Component Inspector, expand **Pin 26** and set these properties to these values:

| Property | Value | Note |
|---|---|---|
| Pull enable | Enabled | |
| Pull select | Pull up | |

All other pin values use default settings, either no change or no initialization.

Scroll to the end of the Properties for this component, you will see the Interrupts property for the component. Set the property **Port Interrupt** to Disabled.

Finally, resolve the pin sharing conflict. On the **Pin** property of pin 26, right click and choose Pin Sharing Enabled. Essentially this tells Processor Expert that you know about this and to ignore the potential problem, you are explicitly sharing this pin.


*Component Inspector — Basic | Advanced | Expert

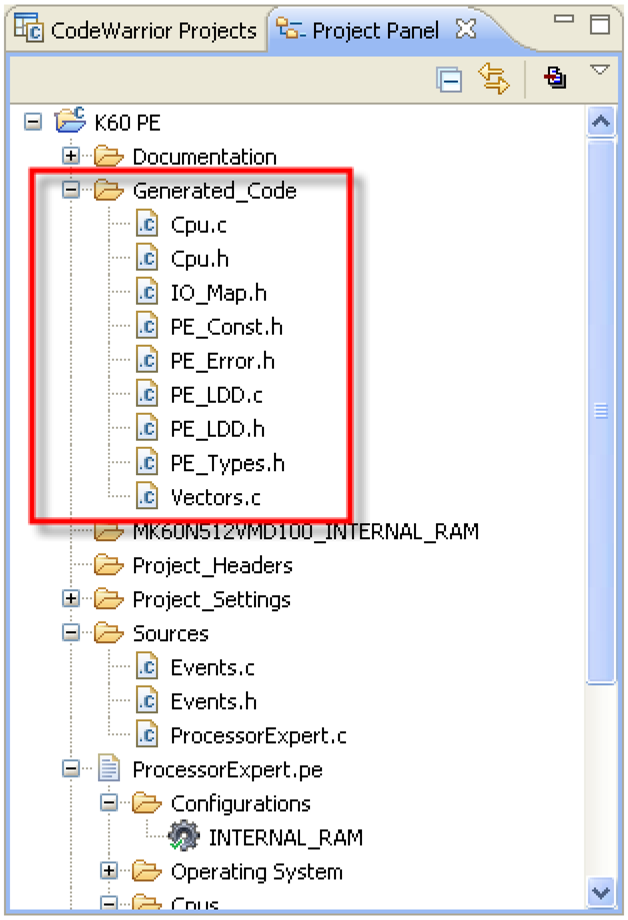
Properties | Methods | Events

| Name | Value | Details |
|---|---|---|
| Pin 25 | Disabled | |
| Pin 26 | Enabled | |
| Pin | PTE26/UART4_... | Peripheral is already used by the component GPIO2 |
| Pin output | Disabled | |
| Open drain enable | No initialization | |
| Pull enable | Enabled | |
| Pull select | Pull Up | |
| Output value | No change | |
| Slew rate | No initialization | |
| Drive strength | No initialization | |
| Interrupt configuration | No initialization | |
| Digital filter enable | No initialization | |
| Passive filter enable | No initialization | |
| Lock | No initialization | |
| Pin 27 | Disabled | |
| Pin 28 | Disabled | |
| Pin 29 | Disabled | |
| Digital filter clock source | Bus clock | |
| Digital filter width | 0 | |
| Interrupts | | |
| Port interrupt | Disabled | |
| Initialization | | |
| Call Init method | yes | |

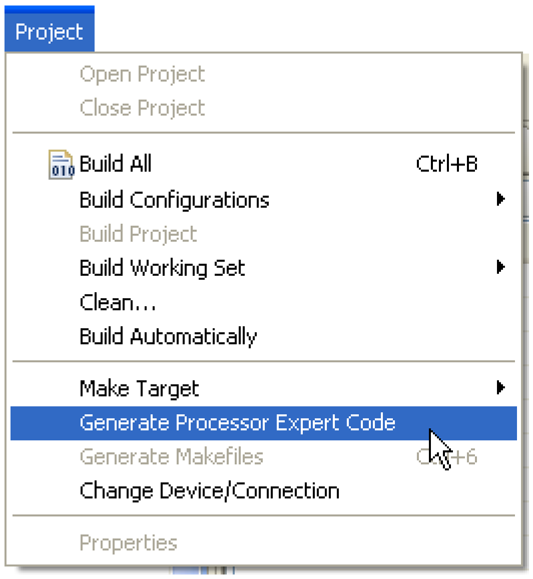
When complete, the error goes away and the other properties for the component and the pin should look like the image above.

## 7 Generate Code

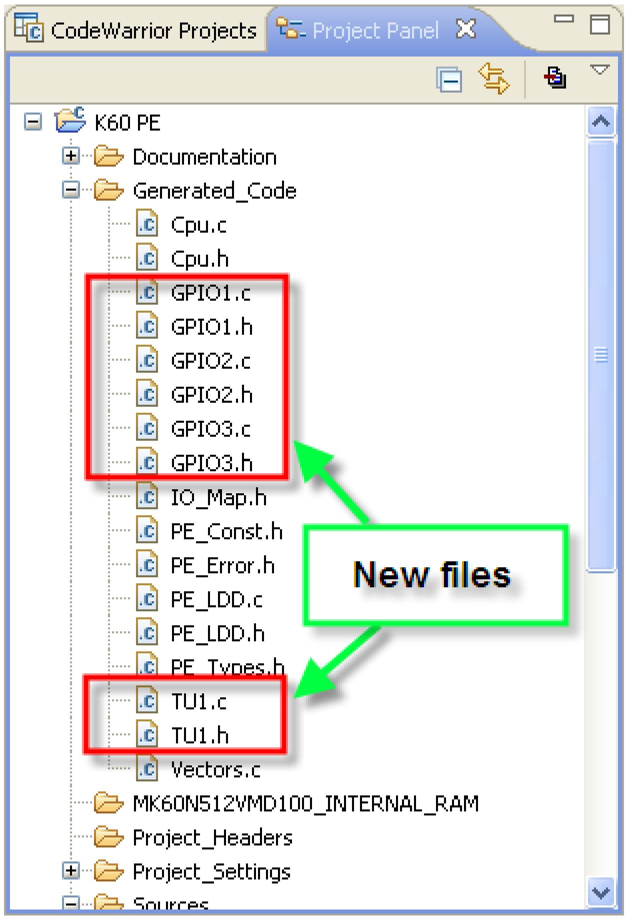
Before you generate code, expand the Generated Code directory in the Projects view, as shown here, just to see what is already in this folder.



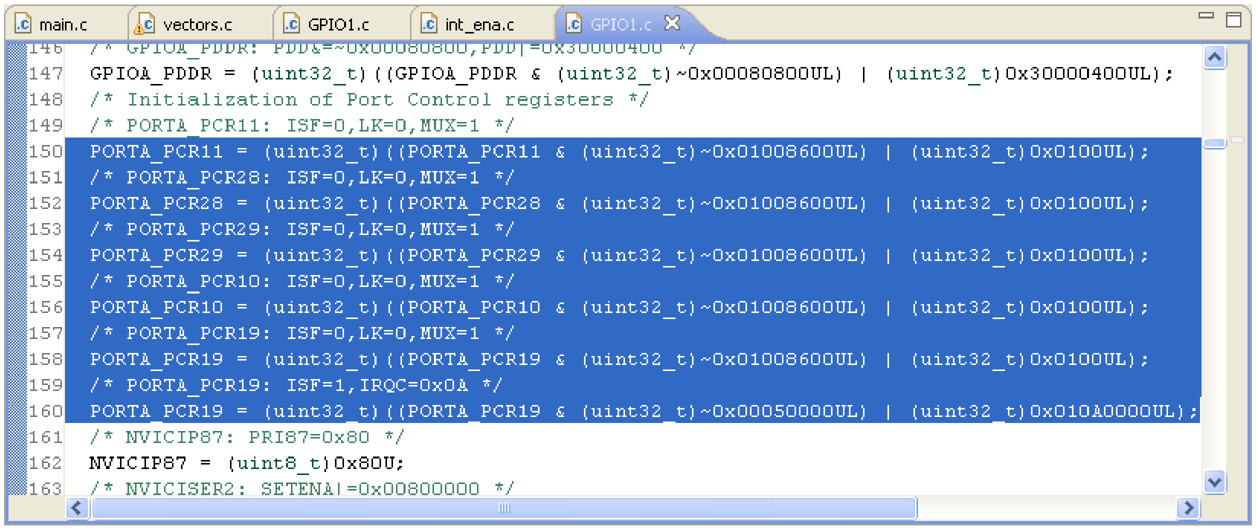
Now generate code. Go to the Project menu and choose Generate Processor Expert Code.



When you do, a progress dialog will appear. There should be no errors. When complete, new source files will appear in the Generated Code directory.



Feel free to open and explore these files. For example, here is the code that sets up the pins driving the LEDs, from `GPIO1.c`



```
/* GPIOA_PDDR: PDD&=~0x00080800,PDD|=0x30000400 */
GPIOA_PDDR = (uint32_t)((GPIOA_PDDR & (uint32_t)~0x00080800UL) | (uint32_t)0x30000400UL);
/* Initialization of Port Control registers */
/* PORTA_PCR11: ISF=0,LK=0,MUX=1 */
PORTA_PCR11 = (uint32_t)((PORTA_PCR11 & (uint32_t)~0x01008600UL) | (uint32_t)0x0100UL);
/* PORTA_PCR28: ISF=0,LK=0,MUX=1 */
PORTA_PCR28 = (uint32_t)((PORTA_PCR28 & (uint32_t)~0x01008600UL) | (uint32_t)0x0100UL);
/* PORTA_PCR29: ISF=0,LK=0,MUX=1 */
PORTA_PCR29 = (uint32_t)((PORTA_PCR29 & (uint32_t)~0x01008600UL) | (uint32_t)0x0100UL);
/* PORTA_PCR10: ISF=0,LK=0,MUX=1 */
PORTA_PCR10 = (uint32_t)((PORTA_PCR10 & (uint32_t)~0x01008600UL) | (uint32_t)0x0100UL);
/* PORTA_PCR19: ISF=0,LK=0,MUX=1 */
PORTA_PCR19 = (uint32_t)((PORTA_PCR19 & (uint32_t)~0x01008600UL) | (uint32_t)0x0100UL);
/* PORTA_PCR19: ISF=1,IRQC=0x0A */
PORTA_PCR19 = (uint32_t)((PORTA_PCR19 & (uint32_t)~0x00050000UL) | (uint32_t)0x010A0000UL);
/* NVICIP87: PRI87=0x80 */
NVICIP87 = (uint8_t)0x80U;
/* NVICISER2: SETENA|=0x00800000 */
```

You do not need to modify these files in any way, nor should you. You will need to write some code, but you will do that in the files in the Sources directory.

## 8 Instantiate Components (`ProcessorExpert.c`)

As noted in the previous step, you do need to write code, but you do that in files in the **Sources** directory. In this step you modify the code in the file `ProcessorExpert.c`. Note that this file is created automatically for you. It contains standard `#include` statements and function stubs. Places where you add code are clearly delineated.

In the project window, double click `ProcessorExpert.c` to open the file in the editor. Look for the `main()` function and add the required code in the correct places. Add the bold red statements to the code.

First, define three variables that will hold pointers to the data structures for the components. This happens globally, just before the code for the `main()` function, not inside the function. These variables will be used in `events.c`, and are not local to `main()`. At the time of this writing, this is at line 32 of the file.

```
/* User includes (#include below this line is not maintained by Processor Expert)
*/

LDD_TDeviceData *Led1Data;
LDD_TDeviceData *SW2Data;
LDD_TDeviceData *TimerData;
```

Inside the `main()` function, create instances of these components and initialize them.

```
void main(void)
{
  …
  /* Write your code here */
  /* For example: for(;;) { } */

  Led1Data = GPIO1_Init(NULL);
  SW2Data   = GPIO2_Init(NULL);
  TimerData = TU1_Init(NULL);


  /*** Don't write any code pass this line, or it will be deleted during code
generation. ***/
```

You have now created the components inside the source code. All that remains is writing the event handling code, and then making sure it all works.

## 9 Write the Event Handling Code (`events.c`)

In the `events.c` file you add the code that executes when each switch is pressed. Based on the component configuration, Processor Expert has already created the necessary function stubs for the interrupt routines you need to write. These are in `events.c`.

There are three LEDs controlled: GPIO1 is connected to SW1 and controls LED1. It also drives the LEDs. GPIO2 is connected to SW2 and controls LED2. The timer controls the third LED. LED4 is on all the time.

In each case you are toggling the state of the LED, and GPIO1 defines the bits that handle each LED. So you will call the `GPIO1_ToggleFieldBits()` function.

Double click the `events.c` file to open it in the editor. Then look for the correct function to modify. Add the code in red.

For GPIO1

```
void GPIO1_OnPortEvent(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
      extern LDD_TDeviceData *Led1Data;
      GPIO1_ToggleFieldBits(Led1Data, LED1, 0x1);
}
```

For GPIO2

```
void GPIO2_OnPortEvent(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
      extern LDD_TDeviceData *Led1Data;
      GPIO1_ToggleFieldBits(Led1Data, LED2, 0x1);
}
```

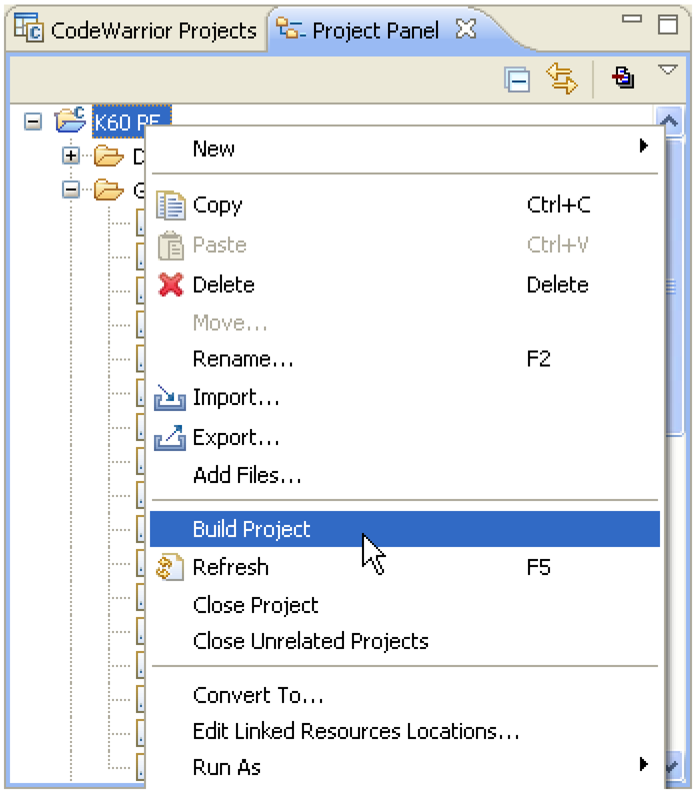
For the Timer

```
{
  /* Write your code here ... */
      extern LDD_TDeviceData *Led1Data;
      GPIO1_ToggleFieldBits(Led1Data, LED3, 0x1);

}
```

## 10 Build the Code

If you have more than one project in your project view, make sure the proper project is the focus. The most reliable way to do this is to right click the project and choose Build Project as shown below. You can also go to the Project menu and choose the same command.



If you encounter errors, look in the Problems view and resolve them. You can ignore any warnings. For example, if you accidentally defined the `LDD_TDeviceData` pointers inside `main()`, instead of before the function, the linker won't be able to find them.

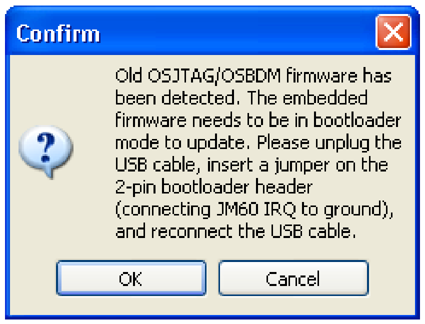
## 11 Download/Debug/Run

If the project builds correctly, it is time to download to the board and watch it work. Ensure that the USB cable that came with the board connects the board to the host computer's USB port.

There are multiple ways to issue the Debug command You can right click the project in the projects view and choose Debug As->CodeWarrior Download. Alternatively, you can go to the Run menu and choose Debug (F11). . If you have multiple projects in the projects view, make sure the correct one has focus.

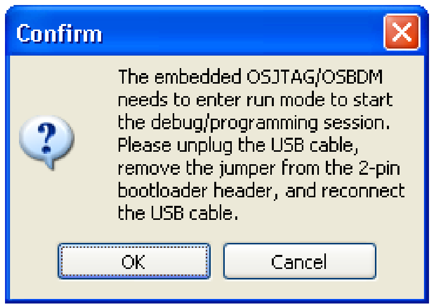
If you see a dialog asking for which configuration to use, choose the one for Internal RAM. If you followed instructions carefully, there will not be a Flash configuration.

Firmware may change after the boards have been manufactured and shipped. As a result, you may encounter this alert when you attempt to download software to the board:


Confirm

Old OSJTAG/OSBDM firmware has been detected. The embedded firmware needs to be in bootloader mode to update. Please unplug the USB cable, insert a jumper on the 2-pin bootloader header (connecting JM60 IRQ to ground), and reconnect the USB cable.

Follow the instructions carefully. Unplug the USB cable. As it comes from the factory, the K60 board has a free jumper on the board. Look for the two pins labeled JM60 Boot and put a jumper on those pins. Then reconnect the USB cable and click OK. The new firmware will download. A new dialog will appear when the process is complete.


Confirm

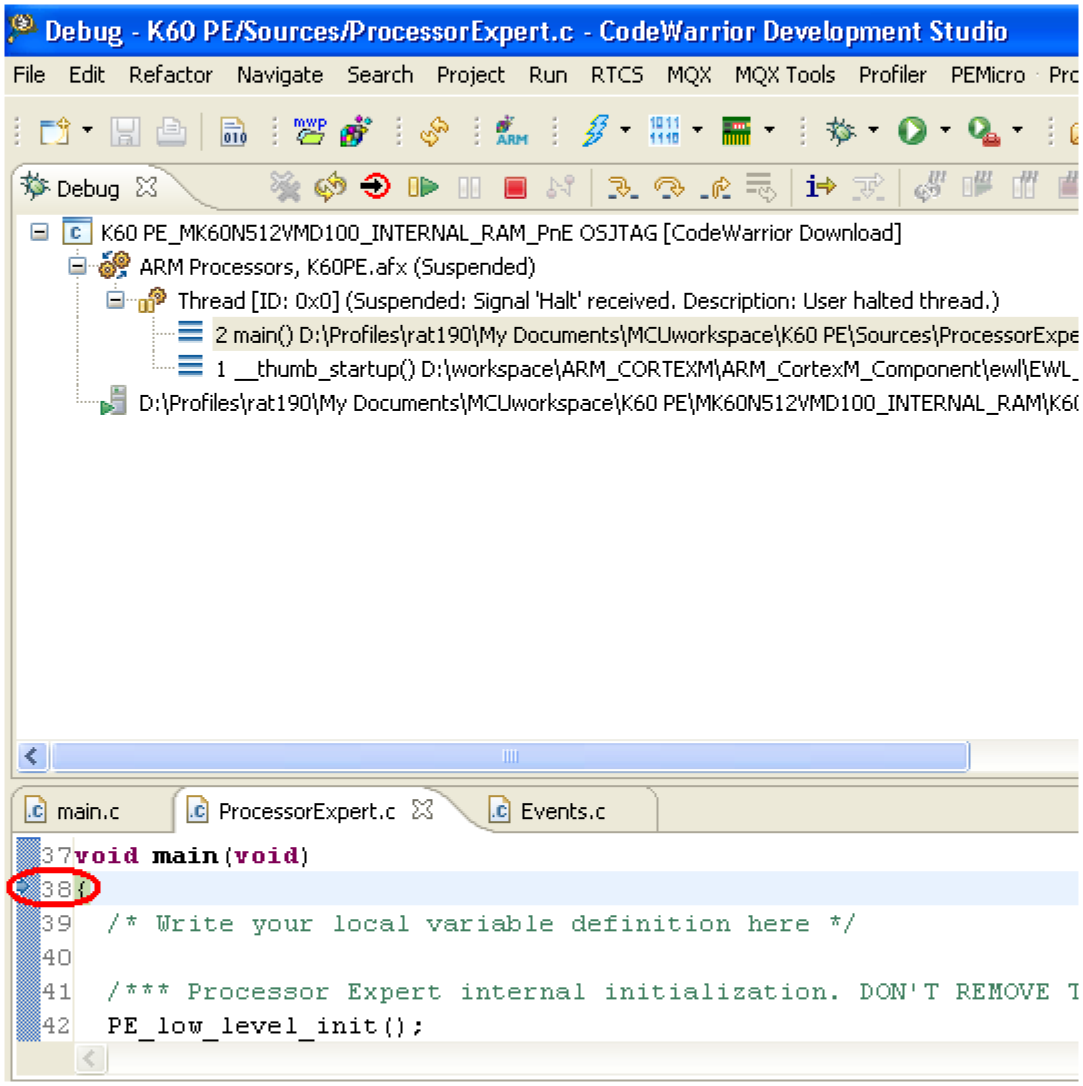
The embedded OSJTAG/OSBDM needs to enter run mode to start the debug/programming session. Please unplug the USB cable, remove the jumper from the 2-pin bootloader header, and reconnect the USB cable.

Unplug the cable, remove the jumper, and reconnect the cable. Then click OK. (You can store the jumper on the board, just set it so that it does not connect pins.)

You may or may not encounter the firmware issue, or the multiple configurations issue. Once resolved, you should not see them again. In an updated environment, this is what happens.

Issue a Debug command.

The project's application downloads to the board. This will take a few moments. The code stops at the first line of `main()`. The program counter arrow is a bit subtle, so it is circled in the screenshot below.


Debug - K60 PE/Sources/ProcessorExpert.c - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Run RTCS MQX MQX Tools Profiler PEMicro Pro
Debug
main.c
ProcessorExpert.c
Events.c

```
K60 PE_MK60N512VMD100_INTERNAL_RAM_PnE OSJTAG [CodeWarrior Download]
  ARM Processors, K60PE.afx (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() D:\Profiles\rat190\My Documents\MCUworkspace\K60 PE\Sources\ProcessorExpe
      1 __thumb_startup() D:\workspace\ARM_CORTEXM\ARM_CortexM_Component\ewl\EWL_
  D:\Profiles\rat190\My Documents\MCUworkspace\K60 PE\MK60N512VMD100_INTERNAL_RAM\K60
```

```
void main(void)
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE T
  PE_low_level_init();
```

Click the Resume button and the code runs.

The blue LED should be on and stay on. The green LED should be flashing about once per second.

Press and hold SW2. The yellow LED should light as long as you hold down the switch. When you release the switch, the LED turns off.

Press SW1. The orange LED should toggle on or off.

If the LEDs do not behave as expected, have fun debugging! For example, if an LED does not light at all, check the Processor Expert configuration – did you set the associated pin to be an output signal?

Even if everything is working, feel free to set breakpoints in the code to study how it all works.

Click the Pause button to stop execution. Click the Terminate button to end debugging.

## 12 Conclusion

You have successfully built a complete application from scratch, configured drivers, and built all the necessary initialization code. Along the way you have seen how the Processor Expert tool, a genuine expert system for driver configuration, warns you of potential difficulties based on its comprehensive knowledge of SOC components. As well, the user interface enables you to configure a driver quickly and easily. The tool handles most of the low-level work cleanly and reliably, and creates function stubs for the ISRs you need, based on the driver configuration you specified.

Have fun experimenting. Modify the behavior of the switches, or the timing rate. The pads around the LED are touch sensitive! You could write code so that a touch on the blue LED toggles it on or off, or touch the green LED to change the flashing rate.

Congratulations and good luck in your programming endeavors.

## 13 Revision History

**Table 1. Revision History**

| Rev. Number | Date | Substantive Change |
|---|---|---|
| 0.1 | 04 April 2011 | Original Draft – Work in Process |
| 1.0 | 19 May 2011 | Major rewrite and expansion, put in proper template |