Freescale Semiconductor
Application Note

Document Number: AN3467
Rev. 0, 05/2007

Using Processor Expert with Flexis™ Microcontrollers

by: Bruno Castelucci / Paulo Knirsch
Field Application Engineers
Latin America Team, Brazil
Freescale Semiconductors

This application note uses Processor Expert embedded beans and QE128 Flexis microcontrollers (MCU) to demonstrate integrated peripheral initialization and functionality on both an 8-bit and 32-bit QE128. Processor Expert is a tool that helps reduce development time for embedded software creation and markets products faster. This document provides a basic easy to use overview of Processor Expert.

The Freescale Controller Continuum provides stepwise compatibility for an easy migration path, from the ultra-low-end RS08 to our highest-performance ColdFire® V4 devices.

CodeWarrior™ development tool helps you reuse and speed the application development through the board portfolio of the Controller Continuum.

Flexis™ Series of Microcontrollers is an 8-bit to 32-bit Connection Point on the Freescale Controller Continuum. Pin-to-pin compatibility between many Flexis devices allows controller exchanges without board redesign. Microcontrollers S08 through ColdFire

Contents

freescale™ semiconductor

V1 (CFV1) share a common set of peripherals and development tools to deliver the ultimate in migration flexibility.

Flexis -- Is a single development tool to ease migration between 8-bit (S08) and 32-bit (CFV1). And a common peripheral set to preserve and maximize the use of the software and hardware migration.

CodeWarrior provides a developed environment for Flexis devices that re-target your application code to a seamless new Core (CFV1 or S08), taking you to the next step of an 8-bit to 32-bit compatibility.

1 Processor Expert and Embedded Beans

This section provides an overview of Processor Expert and embedded bean basics.

Processor Expert is an optional software plug-in for Freescale CodeWarrior development tools. Processor Expert provides quick application object-oriented programming for embedded systems. MCU peripherals are configured through a graphical user interface (GUI) within CodeWarrior integrated development environment (IDE), generating the initialization and another user support code.

Figure 1 illustrates CodeWarrior IDE workspace with the enabled Processor Expert function. It shows the project manager, bean selector, error, bean inspector, and central processor unit (CPU) Processor Expert windows. Section 3.1 Creating a project with Processor Expert provides details on how to configure a CodeWarrior project to include Processor Expert.

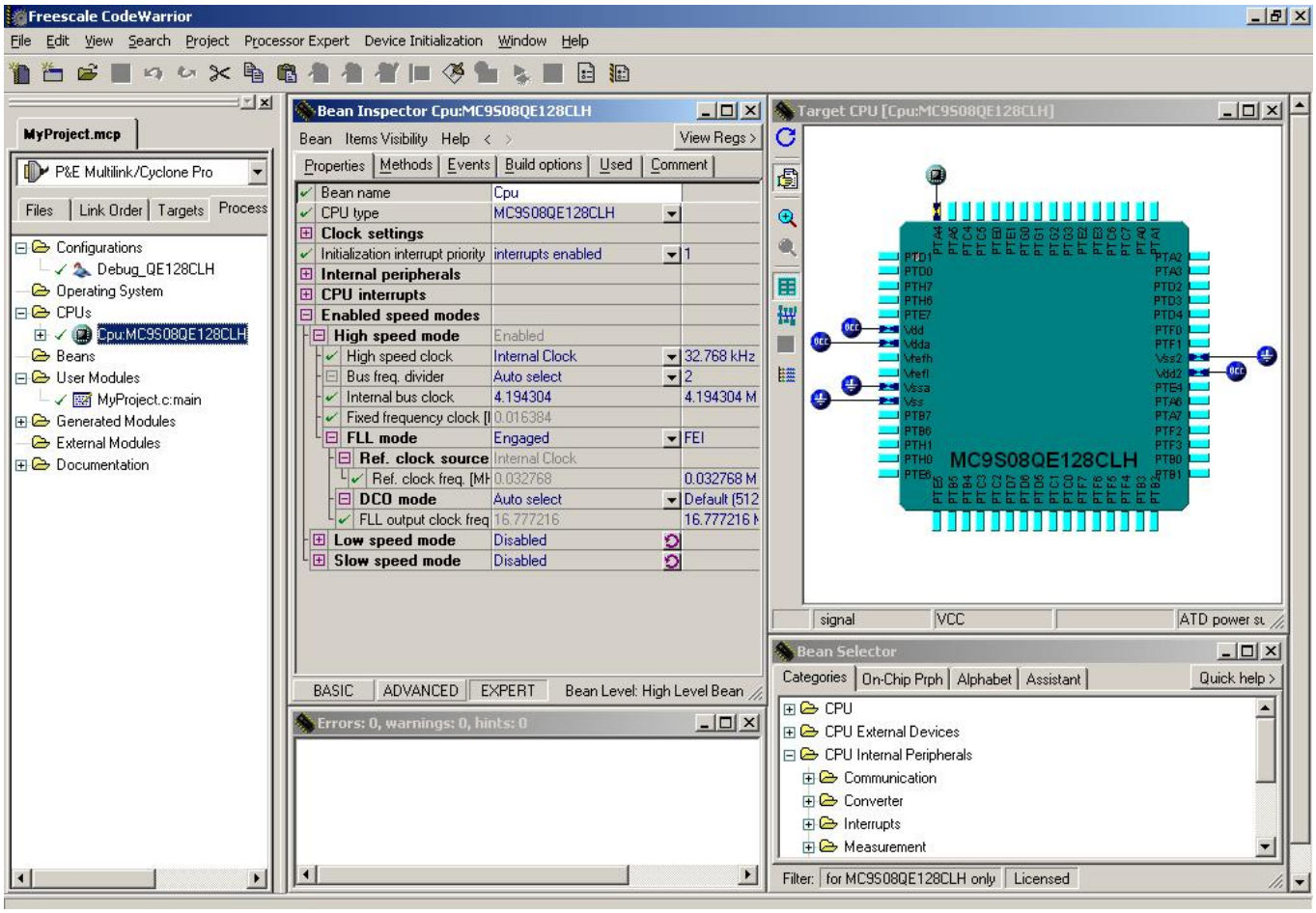

Figure 1. CodeWarrior Workspace

1.1 Processor Expert Benefits

Processor Expert uses an object-oriented application that builds methodology using embedded beans. The embedded beans read the MCU hardware and register details into an intuitive software application programmer interface (API). Embedded beans provide a software API and graphical interface to initialize the MCU.

An expert knowledge system works in the background and checks that all the MCU settings and configurations do not conflict with one another. The Processor Expert software API and the expert knowledge system enable an application developed in Processor Expert to be extremely portable among MCU processors and Freescale MCU processors. Besides the benefit of reusing Processor Expert, other benefits are:

- Easy to program and set-up CPU/MCU peripherals with a limited knowledge.
- Provides an interface to configure modules in real-world terms, such as baud rates.
- Provides ready-to-use hardware drivers for peripherals.
- Provides basic software solutions such as real time clock (RTC) functionality.
- Ability to create user-defined embedded beans.

- Design-time settings verified by the expert knowledge system.
- Allows the use of external code, libraries, and modules.

1.2 What is an Embedded Bean?

Embedded beans are ready-to-use and tested building blocks for applications. Embedded beans read embedded programming by providing a unified API across platforms and hiding the implementation details. This characteristic makes the application portable.

The embedded beans link together functionality into properties, methods, and events. More details are provided here:

- Properties - During the application design-time the embedded beans behavior is defined and then compiled. These include MCU initialization settings such as speed of serial line, time period of the periodical interrupt, or number of channels of A/D converter. Memory allocations or external crystal speed can not change during run-time.
- Methods - The embedded beans behavior can be modified during the application runtime such as receiving serial characters, changing the SCI baud rate, or driving/reading a pin value. When modifying an attribute in the run time the embedded beans provide the programmer functions to insert in the code.
- Events - These embedded beans provide function calls when important changes happen in the bean.

In this document our main purpose is to provide the basic steps with Processor Expert. Several uses of the embedded beans are described enabling users to develop real applications with this powerful tool. Properties, methods and events are explored demonstrating how they can be used in many types of applications.

2 QE Overview

The MCF51QE128 and MC9S08QE128 are the first Flexis MCUs of the Freescale Controller Continuum. These MCUs link an 8-bit and 32-bit word using a single development tool, common peripheral set and, pin-to-pin compatibility. Some target applications for these devices are:

- Battery chargers.
- Secure boot co-processors.
- Security and alarm systems.
- Electronic power meters.
- Sensing systems.
- Handheld devices.
- Health monitors.
- Small appliances.
- Human input devices.
- Smart batteries.
- Industrial controls.

- Smoke and carbon monoxide detectors.
- Lighting controls.
- Toys.
- PC peripherals.
- Watchdog co-processors.
- Personal care appliances.
- Wireless communications.
- Remote controls.
- Wireless sensor applications.

These are some characteristics of the Flexis MCUs Freescale Controller Continuum:

- 8-Bit HCS08 CPU
  - Up to 50.233 MHz at 3.6 V to 2.1 V, and 20 MHz at 2.1 V to 1.8 V both across a temperature range of -40°C to 85°C.
  - An HC08 instruction set with added background (BGND) instruction.
  - Support for up to 32 interrupt/reset sources.
- 32-Bit Version 1 ColdFire CPU
  - Up to 50.33 MHz at 3.6 V to 2.1 V, and 20 MHz at 2.1 V to 1.8 V both across a temperature range of -40 °C to 85 °C.
  - Provides 0.94 Dhrystone 2.1 million instructions per second (MIPS) per MHz performance when running from internal RAM (0.76 DMIPS/MHz from flash).
  - Implements instruction set revision C (ISA_C).
  - Support for up to 30 peripheral interrupt requests and seven software interrupts.
- On-Chip Memory
  - FLASH read/program/erase over a full range of operating voltage and temperature.
  - Random-access memory (RAM).
  - Security circuitry to prevent unauthorized access to RAM and FLASH contents.
- Power-Saving Modes
  - Two very low power stop modes, one of these allows limited use of peripherals.
  - Reduced power wait mode.
  - Peripheral clock enable register disables clocks in modules not in use, thereby reducing currents. This allows clocks to remain enabled to specific peripherals in stop3 mode.
  - A very low power external oscillator used in stop3 mode to provide accurate clock source to active peripherals.
  - Very low power real time counter used in run, wait, and stop modes with internal and external clock sources.
  - Typical wake up time of 6 µs for stop3 mode.
- Clock Source Options

  - External oscillator (XOSC) -- is a loop control Pierce oscillator using a crystal or ceramic resonator range of 31.25 kHz to 38.4 kHz or 1 MHz to 16 MHz.
  - Internal clock source (ICS) module -- contains a frequency-locked-loop (FLL) controlled by an internal or external reference. The precision trimming of an internal reference allows 0.2% resolution and 2% deviation over temperature and voltage. The ICS supports CPU frequencies from 2 MHz to 50.233 MHz.
- System Protection
  - Watchdog computer operating properly (COP) -- is a reset with option to run from dedicated 1 kHz internal clock source or bus clock.
  - Low-voltage detection with reset or interrupt, and selectable trip points.
  - Illegal opcode detection with reset.
  - FLASH block protection.
- Development Support
  - Single-wire background debug interface.
  - Multiple breakpoint capability.
  - Different trigger settings and trace capability.
- Peripherals
  - Analog to digital converter (ADC)
    - 24-channel, 12-bit resolution.
    - 2.5 µs conversion time.
    - Automatic compare function.
    - 1.7 mV/°C temperature sensor
    - Internal bandgap reference channel
    - Operation in stop3
    - Fully functional at 3.6 V to 1.8 V.
  - Analog comparator (ACMPx)
    - Two analog comparators with selectable interrupt on rising, falling, either edge of the comparator output.
    - Compare option to fixed internal bandgap reference voltage.
    - Outputs optionally routed to TPM module.
    - Operation in stop3.
  - Serial communications interface (SCIx)
    - SCI-Full duplex non-return to zero (NRZ).
    - LIN master extended break generation.
    - LIN slave extended break detection.
    - Wake up on active edge.
    - SPIx-Two serial peripheral interfaces with Full-duplex or single-wire bidirectional.
    - Double-buffered transmit and receive.

  - Master or Slave mode.
  - Either the MSB-first or the LSB-first shift.
- Inter-integrated circuits (IICx)
  - Two IICs.
  - Up to 100 kbps with maximum bus loading.
  - Multi-master operation.
  - Programmable slave address.
  - Interrupt driven byte-by-byte data transfer.
  - Supports broadcast mode and 10 bit addressing.
- Timer/pulse-width modules (TPMx)
  - One 6-channel (TPM3) and two 3-channel (TPM1 and TPM2).
  - Selectable input capture, output compare, buffered edge-/center- aligned PWM on each channel.
- Real-time counter (RTC)
  - 8-bit modules counter with binary or decimal based prescaler
  - External clock source for precise time, time-of-day, calendar and task scheduling functions.
  - Free running on-chip low power oscillator (1 kHz) for cyclic wake-up without external components.
  - Runs in all MCU modes.

- Input/Output
  - 70 general purpose input/output (GPIOs), 1 input-only and 1 output-only pin.
  - 16 keyboard interrupt (KBI) with selectable polarity.
  - Hysteresis and configurable pull up device on all input pins.
  - Configurable slew rate and drive strength on all output pins.
  - SET/CLR registers on 16 pins (PTC and PTE).
  - 16 bits of Rapid GPIO connected to the CPU's high-speed local bus with set, clear, and toggle functionality (only MCF51QE128)
- Package Options
  - 80-LQFP, 64-LQFP
  - 48-QFN, 44-QFP, 32-LQFP (only MC9S08QE)

3 Development

This next section explains how to create a new project in CodeWarrior with Processor Expert tool. This enables the procedure to configure the microcontroller CPU and the migration between the Flexis family devices. The Processor Expert and the Flexis MCUs are explained using labs divided by peripherals, or groups of peripherals. Each lab has a brief explanation of its fuctionality and a description of each peripheral it uses. For example, the first lab will use 2 peripherals (Timer and I/O) to make a LED blink.

3.1 Creating a project with Processor Expert

This section shows the procedure on how to create a new project for Code Warrior with the Processor Expert tool and gives a brief explanation on how to configure the selected microcontroller CPU.

3.1.1 Creating a new project

You can create a Code Warrior project using Processor Expert with just a few Clicks.

1. When you open CodeWarrior, the Startup dialog box, shown in Figure 2, appears. Click Create New Project, button. If the Startup dialog does not appear, go to the File menu, then click Startup Dialog....

Figure 2. Startup dialog

   Notice the Wizard Map on the left side of the Microcontrollers New Project window. This map has the steps for the project creation wizard. The first step is Device and connection.
2. For the QE family of Flexis MCUs, expand Flexis and QE Family, choose one MCU, this example uses the MC9S08QE128. See step 2 in Figure 3.
3. Choose the default connection. Click the P&E Multilink/Cyclone Pro option to build your project for the DEMOQE128. See step 3 in Figure 3.
4. Click Next. See step 4 in Figure 3.

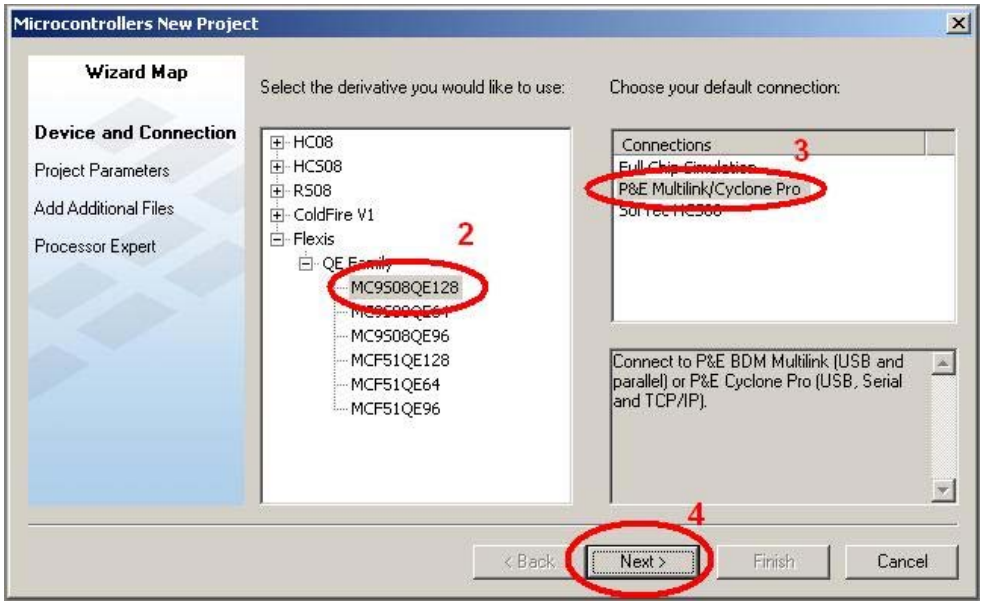

Figure 3. Microcontroller new project dialog

5. In the Project Parameters window type your project name in the corresponding field, this example uses the name MyProject. See step 5 in Figure 4.
6. Click Next. See step 6 in Figure 4.

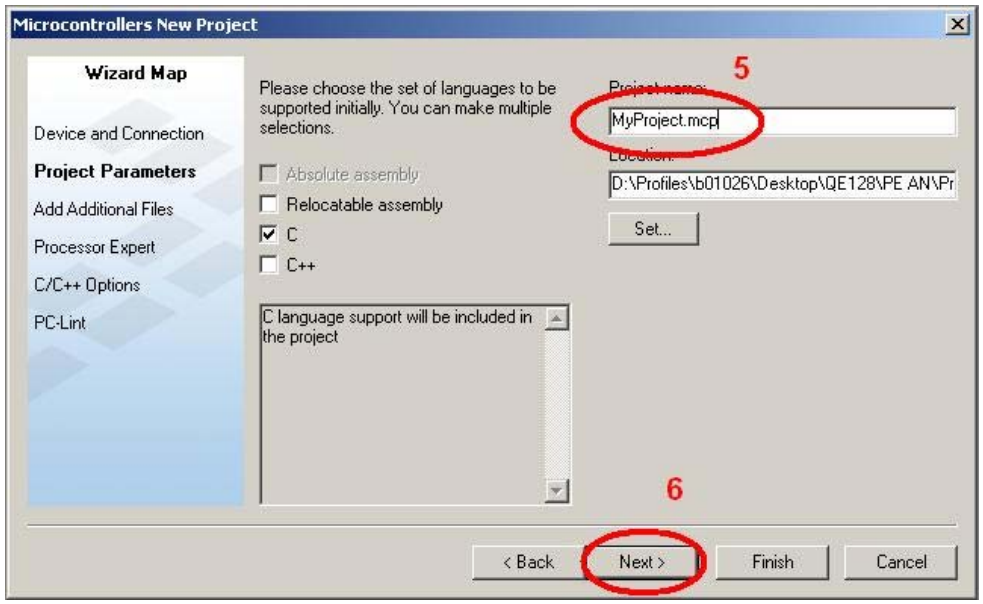

Figure 4. Microcontroller new project dialog

7. Click Next again. In the Add Additional files window, here you can add previous files that have already been created.
8. In the Processor Expert window, select the Processor Expert option. See step 8 in Figure 5.
9. Click Next. See step 9 in Figure 5.

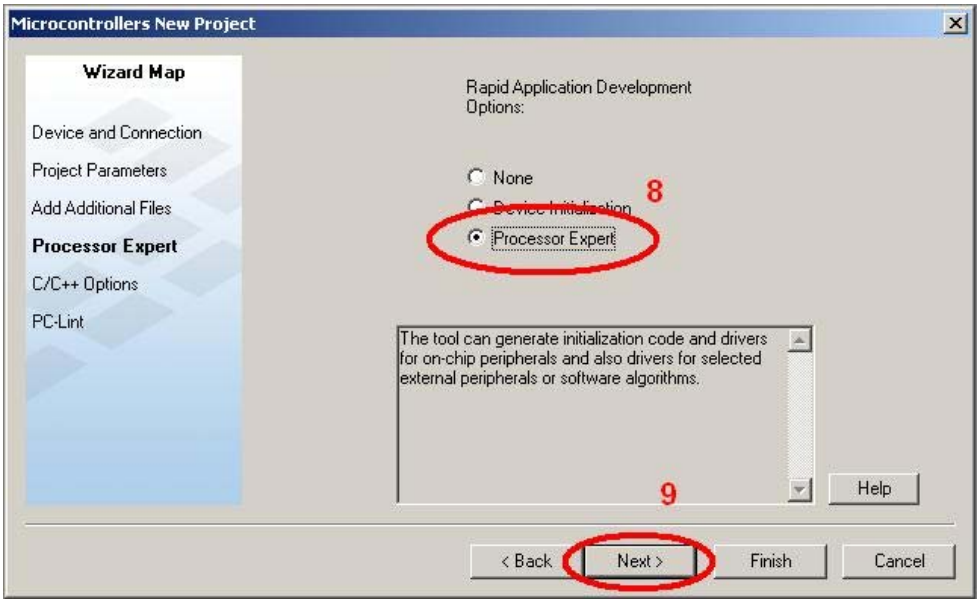

Figure 5. Microcontroller new project dialog

10. In the C/C++ Options window, click Next again, here you can choose which programming language is used to develop your project.
11. In the PC-Lint window, click the Finish button.
12. Select the MCU package you want. This example software uses the MC9S08QE128CLH 64-pins LQFP, this is used by the DEMOQE128 board. See Figure 6.
13. Click Ok.

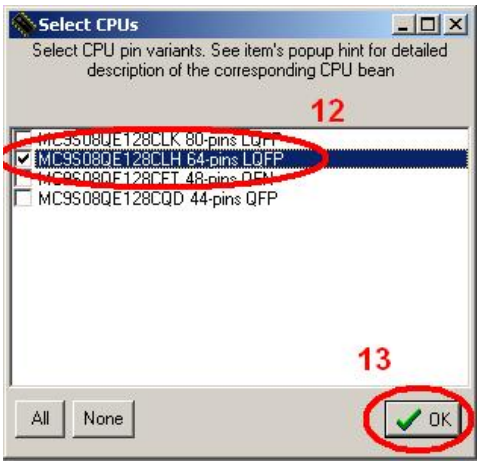

Figure 6. Select CPUs Dialog

14. In the next window you need to choose if you are creating this project for debug, release or both. For this example select only the debug option and click the Ok button. See Figure 7.

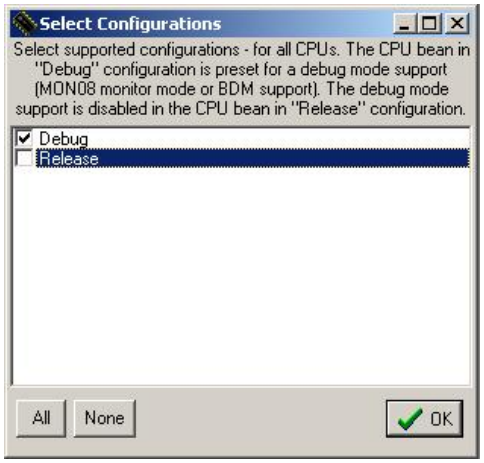

Figure 7. Select CPUs Dialog

You created a new project!

3.2 Configuring the CPU bean

The CPU bean is automatically created when beginning a project with Processor Expert. For the Flexis QE128, choose between the S08 8-bit MC9S08QE128 or the ColdFire V1 32-bit MCF51QE128. This example uses the MC9S08QE128 CPU.

Figure 8 shows on the left the Processor Expert tab. Choose the CPU bean to be configured. On the right, after the CPU bean is chosen the bean inspector window is displayed. Now configure the bean.

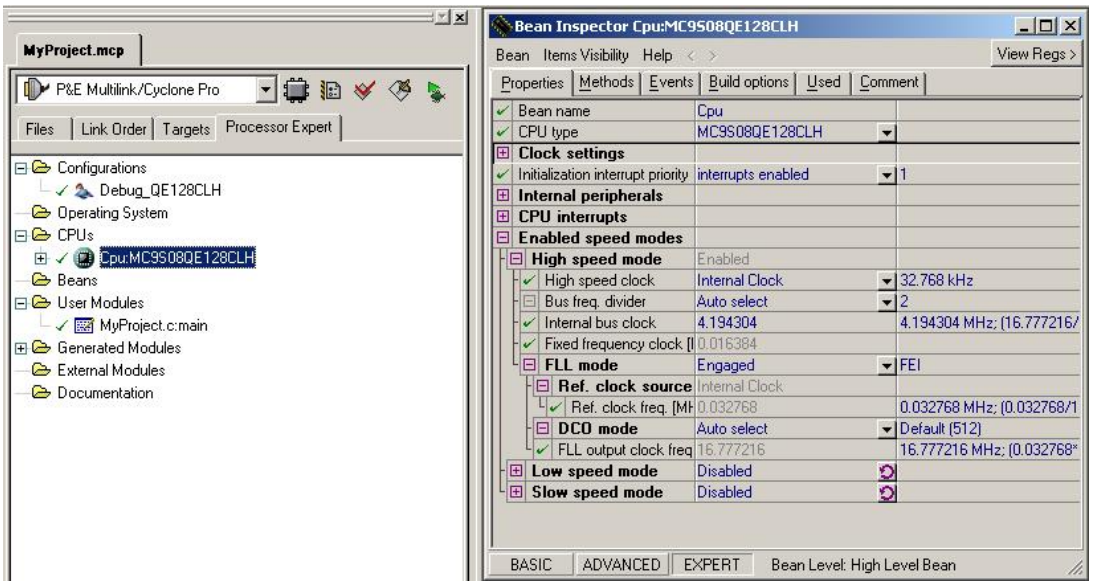

Figure 8. Bean Inspector window - CPU Bean

NOTE

Remember to always choose the Expert view at the bottom in the Bean Inspector window. This allows to see all the available configurations for the bean.

The Bean Inspector window allows the bean parameters to be configured. The Methods and Events code generation can also be enabled or disabled.

In order to configure the bean parameters, select the Properties tab and modify the properties values. Below you can find a list and description of the most important properties for the CPU MC9S08QE128 bean.

- Clock settings -- This option allows to choose an internal or external clock reference, and provides other clock configuration settings.
- Internal peripherals -- This group contains peripheral settings. Changes may be needed for the parameters to enable or disable the peripherals clock. Notice some peripherals clock are disabled by default.
- Internal bus clock -- This field specifies the Internal Bus Clock used for a project. The maximum value for the Flexis QE MCUs, is near 30 MHz.

If needed, select Methods to be created. Do this by selecting the Methods tab in the Bean Inspector window. Click the round arrow button( ). For this bean, by default, there are no Methods created. Below you can find some methods:

- EnableInt -- Enables maskable interrupts
- DisableInt -- Disables maskable interrupts
- SetWaitMode -- Sets the low power mode to Wait mode.
- SetStopMode -- Sets the low power mode to Stop mode. This method is available only if the STOP instruction is enabled. See the STOP instruction property.

Figure 9 shows the Methods tab in the Bean Inspector window.

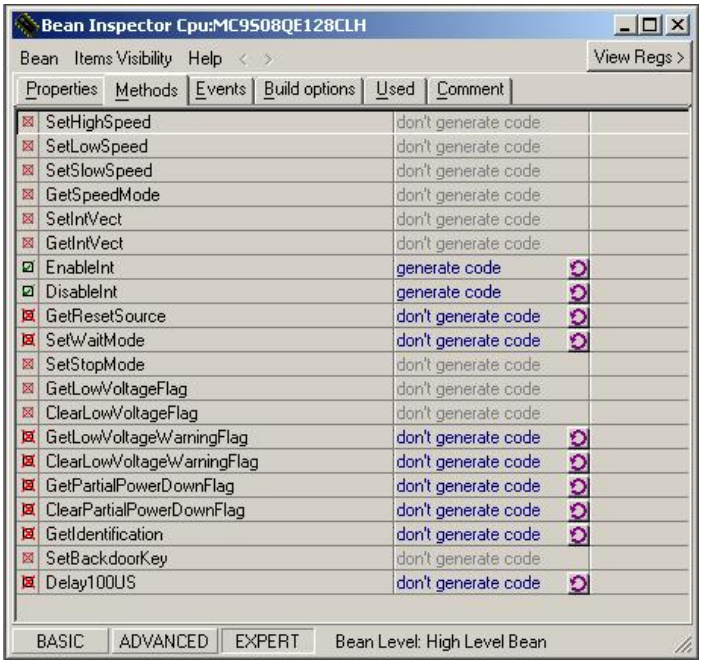

Figure 9. Bean Inspector window - Methods for CPU Bean

Finally, set the Events configuration for this bean. Select the Events tab in the Bean Inspector window. There are 4 events related to the CPU bean, Reset, Low Voltage Detection (LVD), Warning (LVW), and Software Interrupt.

3.3 Migrating CPU inside Flexis family

A project created for one of the Flexis microcontrollers can be easily migrated to other microcontroller, this means that a project initially created for an 8-bit microcontroller can be migrated to a 32-bit one, and vice versa.

This example uses the project created in Section 3.1 Creating a project with Processor Expert, initially created for the MC9S08QE128 and explains how to migrate it to the MCF51QE128.

With your project opened in CodeWarrior, click the "Change MCU/Connection..." button. See Figure 10.

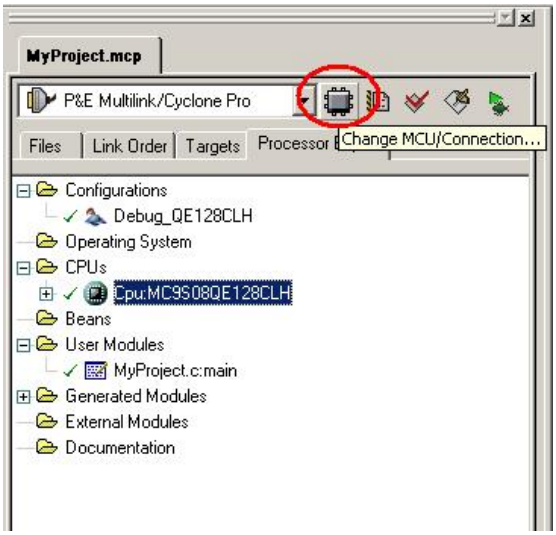

Figure 10. Migrating CPU connection

In the Device and Connection opened window, select the new MCU to migrate. Then select the default connection and click the Finish button. Follow the steps in Figure 11.

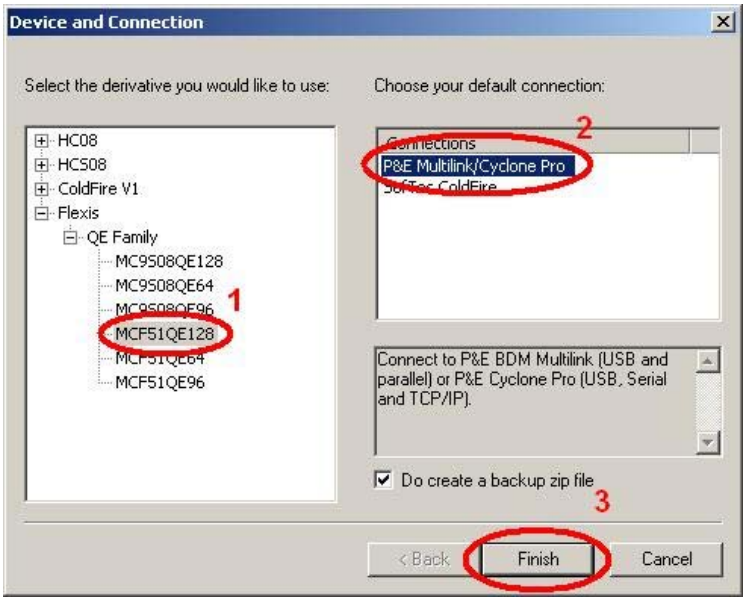

Figure 11. Migrating CPU options

Wait until CodeWarrior finishes files migration. A message may pop-up asking to add a new CPU bean, select yes in this message.

Now both CPUs are in the project window, as shown in Figure 12. Notice now the MCF51QE128 is selected.

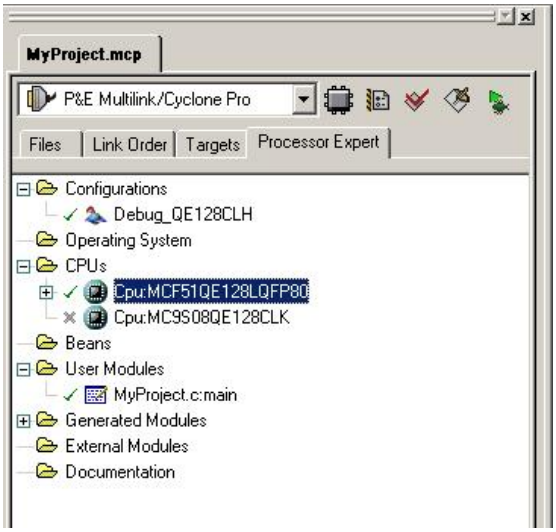

Figure 12. CPUs available

Switching is possible between CPUs by doing the same procedure again.

The project is ready to use with the new MCU!

3.4 Lab1: Timer and I/O

This lab uses a timer and an I/O port to make an LED blink. The timer generates a periodic interrupt that controls the LED state with an I/O port. Every time an interrupt occurs, the LED state changes.

For each peripheral, you will find a brief description, followed by the used beans description and configurations. Then the Lab is presented, with a configuration guide line, integrating two peripherals.

3.4.1 Timer: Peripheral Description

The timer/PWM module (TPM) is a 16-bit counter. It operates as a free-running counter and module counter. When the TPM is configured for a center-aligned PWM it also operates as an up or down counter. The TPM counter operated in a normal up-counting mode provides the timing reference for the input capture, output compare, and edge-aligned PWM functions. The timer counter module registers, TPMxMODH:TPMxMODL, and controls the module value of the counter. The values 0x0000 or 0xFFFF make the counter free running. Software reads the counter value at any time without affecting the counting sequence. Anything written to the TPMxCNT counter registers resets the counter.

The TPM has the following features:

- Each TPM can be configured for buffered, center-aligned pulse-width modulation (CPWM) on all channels.
- Clock sources independently selectable per TPMs.
- Selectable clock sources, bus clock, fixed system clock, and external pin.
- Clock prescaler taps for division by 1, 2, 4, 8, 16, 32, 64, or 128.
- 16-bit free-running and up/down CPWM count operation.

- 16-bit modulus register to control counter range.
- Timer system enable.
- One interrupt per channel, plus a terminal count interrupt for each TPM module.
- Channel features:
  - Each channel is input capture, output compare, or buffered edge-aligned PWM.
  - Rising-edge, falling-edge, or any-edge input capture trigger.
  - Set, clear, or toggle output compare action.
  - Selectable polarity on PWM outputs.

For more detailed and specific data, visit product documentation MC9S08QE128 and MCF51QE128 at www.freescale.com.

3.4.2 Timer: TimerInt Bean

For the LED blink application, use the TimerInt bean. This bean implements a periodic interrupt. The OnInterrupt event is called on periodically with a user configurable period when the bean and its events are enabled.

Figure 13 details the Bean Selector window and the TimerInt Bean. The following steps describe how to add the TimerInt bean to the project:

1. Create a project according to Section 3.1 Creating a project with Processor Expert.
2. Go to menu Processor Expert > View > Bean Selector in CodeWarrior this is the Bean Selector window (Figure 13).
3. Select the tab Categories, expand the folder CPU Internal Peripherals, and the Timer folder.
4. Inside the Timer folder are all the beans related to the Timer.
5. Double-click the TimerInt bean to add it to the project.

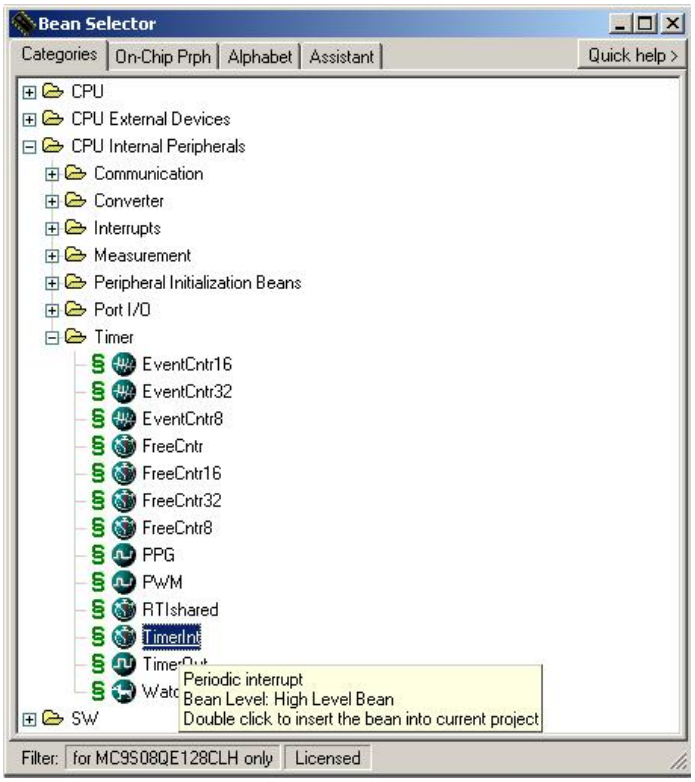

Figure 13. Bean Selector window - TimerInt Bean

These steps describe how to configure the TimerInt bean:

1. In the Processor Expert tab on the left panel, in the Beans folder, the new TimerInt bean is displayed (Figure 14).
2. Double-click the new bean, and the Bean Inspector window activates.
3. To configure the bean, select the Properties tab and modify the properties values. Below is list and description of the most important properties for the TimerInt bean.
   - Periodic Interrupt Source -- selects which hardware timer is the source for the periodic interrupt generator. The default value is the real time counter RTCmod, in this mode the selected hardware timer is a single 8-bit modulus counter. Select any of the available timers. If the TPM timer is selected choose a channel for this timer.
   - Interrupt Period -- defines the period of the Interrupt. Select the time wanted.

4. Figure 15 shows the Methods tab in the Bean Inspector window. This window by default has no Methods created. If needed select which Methods to create by selecting the Methods tab in the Bean Inspector window. Click the round arrow button( ).

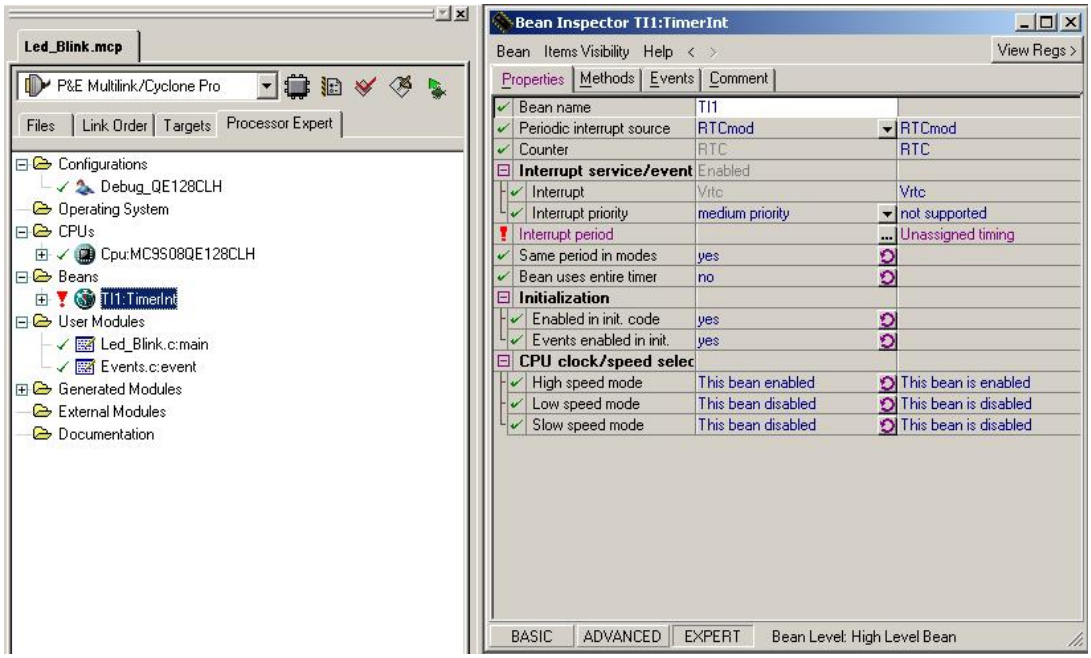

Figure 14. Bean Inspector window - TimerInt Bean

NOTE

Remember always choose the Expert mode. In this mode are all the available configurations for the bean.

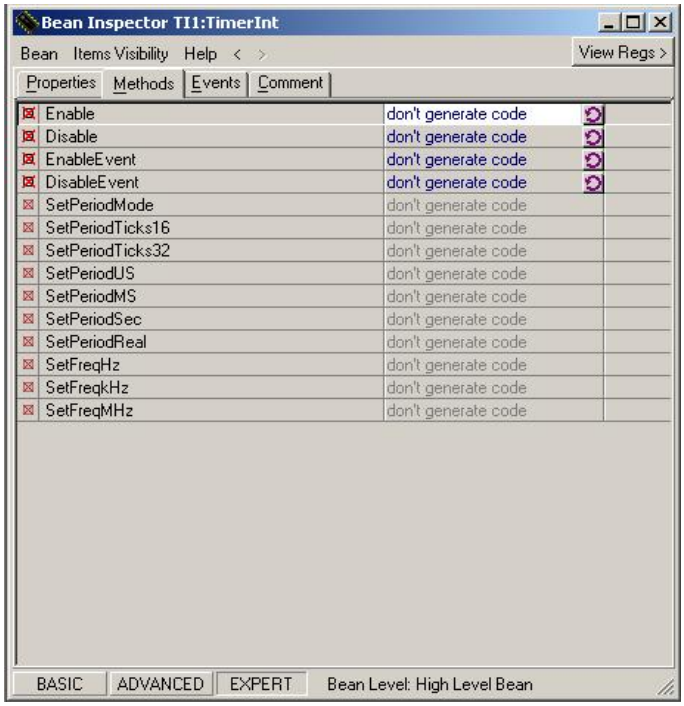

Figure 15. Bean Inspector Window - Methods for TimerInt Bean

5. Figure 16 shows the Events tab in the Bean Inspector window, select the tab to configure. For the TimerInt bean there is only one event enabled, the OnInterrupt.

Figure 16. Bean Inspector Window - Events for TimerInt Bean

6. Compile the project pressing the F7 key. Once an event is enabled, a file named Events.c is generated by the Processor Expert. This file contains all the events enabled for all the beans in the project. This file is under the User Modules folder in the left panel (Figure 17). The event OnInterrupt in the file Events.c is named TI1_OnInterrupt. The code to be executed must be placed inside the TI1_OnInterrupt function.

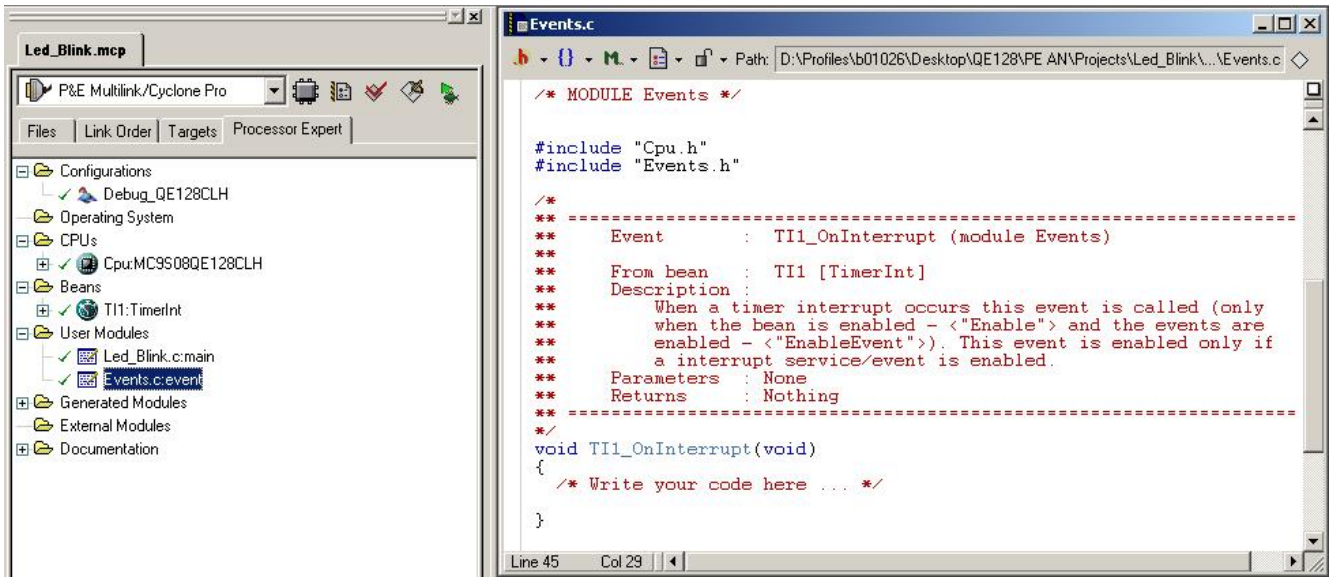

Figure 17. Events.c File for TimerInt Bean

3.4.3 I/O: Peripheral Description

General purpose inputs/outputs are commonly called I/Os. These pins are configured either as digital input or output. In most cases they are multiplexed with another peripheral, meaning that an I/O has another function other than the general digital input/output pin. For example, in the S08QE128, the pin PTA7/TPM2CH2/ADP9 can be configured as a general I/O (PTA7), as a TPM channel (TPM2CH2) or even as an A/D input (ADP9).

When a pin is configured as an I/O beyond the input/output configuration, each pin can be configured to have a slew rate control, drive strength, and internal pull-ups.

An internal pull-up device can be enabled for each port pin by setting the corresponding bit in the pull-up enable register (PTxPEn), but if the pin is configured as an output by the parallel I/O control logic or any shared peripheral function then the pull-up feature is disabled. The pull-up device is also disabled if the pin is controlled by an analog function.

Slew rate control can be enabled for each port pin by setting the corresponding bit in the slew rate register (PTxSEn). When enabled, slew rate control limits the rate at which an output can transition in order to reduce electromagnetic compatibility (EMC) emissions. Slew rate control has no effect on pins that are configured as inputs.

An output pin can be selected to have high output drive strength by setting the corresponding bit in the drive strength select register (PTxDSn). When high output drive is selected, a pin can find and consume greater current. Every I/O pin can be selected as high drive. Make sure that the total current source and absorption limits for the MCU are not exceeded. Drive strength selection is intended to affect the DC behavior of I/O pins, but the AC behavior is also affected. High output drive allows a pin to drive a greater load with the same switching speed as a low output drive enabled pin, into a smaller load. Because of this, the EMC emissions may be affected by enabling pins as high drive.

For more detailed and specific data, visit product documentation MC9S08QE128 and MCF51QE128 at www.freescale.com.

3.4.4 I/O: BitIO Bean

For the LED blink application, use the BitIO bean. This BitIO bean implements a one-bit input/output using one pin of an I/O port.

First add the BitIO bean to your project and configure it in order to use it. The following steps will show how to do that.

1. Create a project according to Section 3.1 Creating a project with Processor Expert.
2. Go to menu Processor Expert > View > Bean Selector in CodeWarrior this is the Bean Selector window (Figure 13).
3. Select the tab Categories, expand the folder CPU Internal Peripherals, and the Port I/O folder.
4. Inside the Port I/O folder are all the beans related to the I/O (Figure 18).
5. Double-click the BitIO bean to add it to your project.

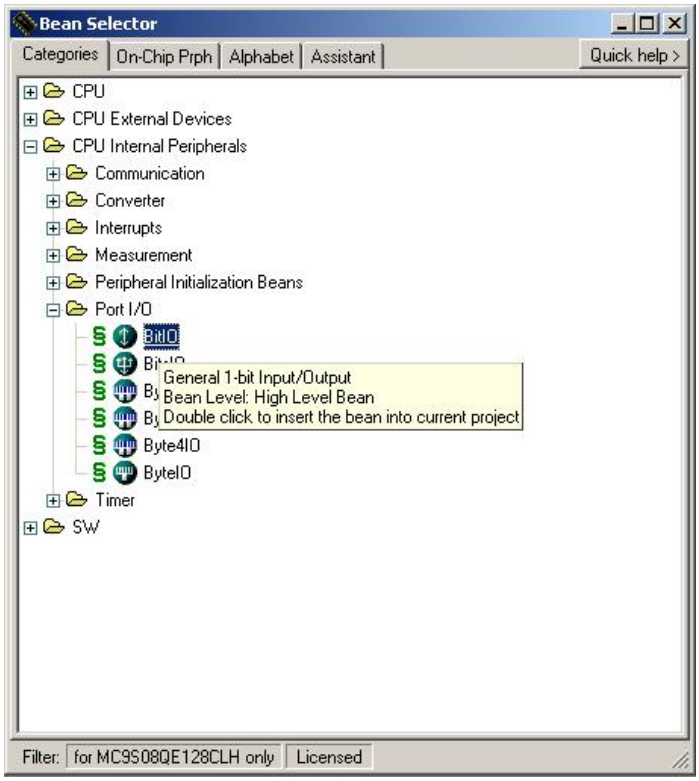

Figure 18. Bean Selector Window - BitIO Bean

These steps describe how to configure the BitIO bean:

1. In the Processor Expert tab on the left panel, in the Beans folder, the new BitIO bean is displayed (Figure 19).
2. Double-click the new bean, and the Bean Inspector window activates.

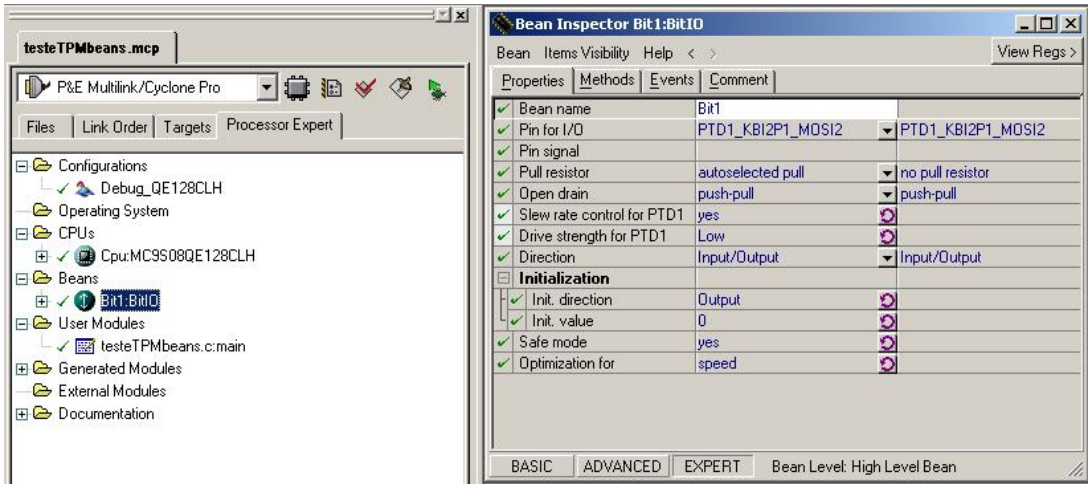

Figure 19. Bean Inspector window - BitIO Bean

NOTE

Remember always choose the Expert mode. In this mode are all the available configurations for the bean.

3. To configure the bean, select the Properties tab and modify the properties values. Below is a list and description of the most important properties for the BitIO bean.
   - Pin for I/O -- MCU pin used by this bean.
   - Pull resistor -- Setting of the pull resistor (only for input mode). There are 6 options:
     - no pull resistor
     - pull up resistor
     - pull down resistor
     - pull up or no pull resistor
     - pull down or no pull resistor
     - autos elected pull resistor
   - Slew rate control for pin -- Determines the slew rate control for the selected pin, if enabled. Slew rate control limits the rate at which an output can transition in order to reduce EMC emissions.
   - Drive strength for pin -- Determines the Drive Strength control for the selected pin. When high drive is selected a pin is capable of sourcing and consuming greater current or driving a greater load with the same switching speed as low drive. EMC emissions may be affected.
   - Direction -- Direction of the bean: input, output, input/output. The direction of the pin can be switched in run time - see SetDir, SetInput and SetOutput methods.
4. If needed select which Methods to create by selecting the Methods tab in the Bean Inspector window (Figure 20). Click the round arrow button( ). For this bean, the methods that should be used are:
   - SetInput -- Sets a pin direction to input, this is available only if the direction = input/output.
   - SetOutput -- Sets a pin direction to output, this is available only if the direction = input/output.

— GetVal -- Returns the input/output value. If the direction is input then the input value of the pin is read and returned. If the direction is output then the last written value is returned.

— PutVal -- The specified output value is set.
  – If the direction is in input, the bean saves the value to a memory or a register.
  – If the direction is output, it writes the value to the pin.

— ClrVal -- Clears the output value when set to zero. It is equivalent to the PutVal(FALSE). This method is available only if the direction = output or input/output.

— SetVal -- Sets the output value to one. It is equivalent to the PutVal(TRUE). This method is available only if the direction = output or input/output.

— NegVal -- Negates the output value inverting it. It is equivalent to the PutVal(!GetVal()). This method is available only if the direction = output or input/output.

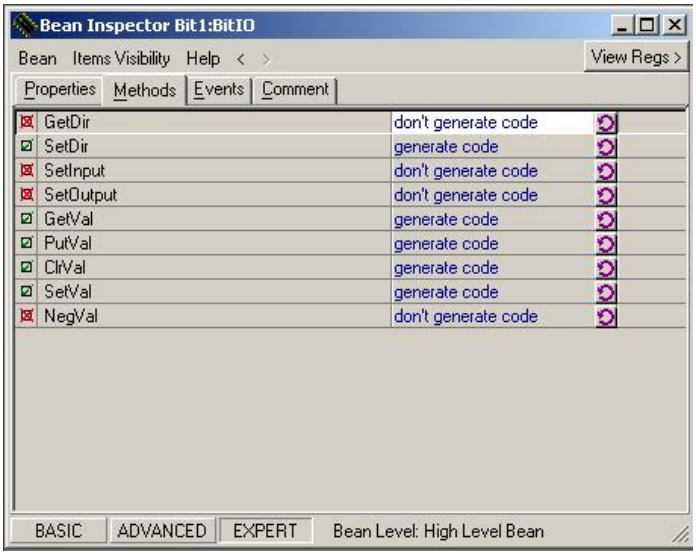

Figure 20. Bean Inspector window - Methods for BitIO Bean

5. There are no Events for this bean.

3.4.5 Lab1: LED Blink - Project configuration

This project is implemented for the DEMOQE128 Board.

For the LED blink application, use the TimerInt bean and the BitIO bean. The TimerInt bean generates an interrupt with a 2 Hz frequency, which is a 500 ms period. The BitIO bean controls the MCU PTC0 pin connected to a LED (LED0) on the board.

The LED state is switched with the BitIO bean method NegVal on the TimerInt, the event OnInterrupt.

Figure 21 shows the Bean Inspector window for TimerInt Bean with the properties values for the LED Blink project. The following steps describe how to configure the LED Blink project:

1. Create a project according to Section 3.1 Creating a project with Processor Expert. Create it with the CPU configured to run with Internal Clock at the default speed.

2. The TimerInt bean and the BitIO bean must be added to your project. Refer to Section 3.4.2 Timer: TimerInt Bean and Section 3.4.4 I/O: BitIO Bean.
3. Once the TimerInt bean is inserted to the project, it appears on the left panel in the Processor Expert tab, in the Beans folder (Figure 17).
4. Configure this bean to generate an event every 500 ms (freq = 2 Hz).
5. Double-click the TimerInt bean, and the Bean Inspector window activates.
6. In the Bean Inspector window configure the following:
   — Let the Periodic interrupt source stay in its default mode (RTCmod).
   — Set the Interrupt period to 500 ms.

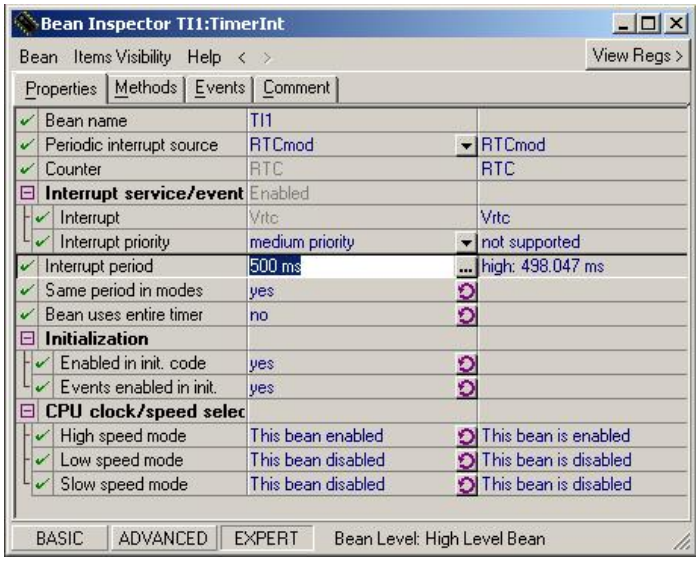

Figure 21. Bean Inspector window - TimerInt configuration on LED_Blink project

7. Configure the TimerInt bean to control the pin PTC0, and set it as an output pin.
8. Double-click the BitIO bean, and the Bean Inspector window activates.
9. Figure 22 shows the Bean Inspector window for BitIO Bean with the properties values for the LED Blink project. In the Bean Inspector window configure the following:
   — Set the Pin for I/O value to PTCO.
   — Set the Direction to Output.
10. The LED Blink project uses the TimerInt event OnInterrupt. This event is configured by default.

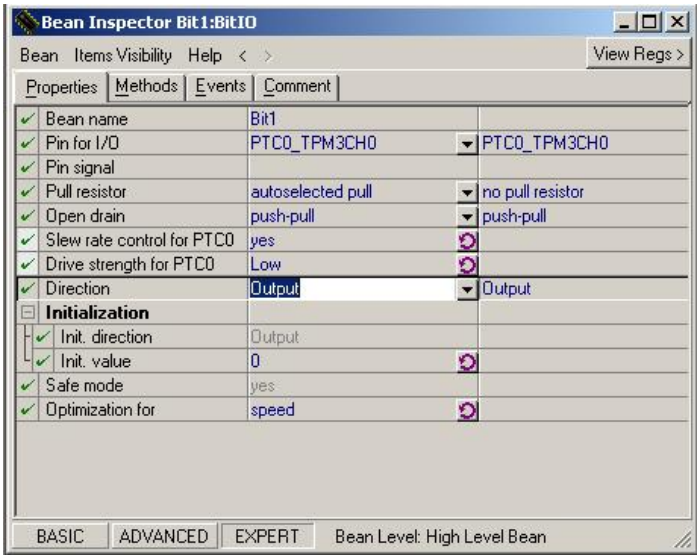

Figure 22. Bean Inspector window - BitIO configuration on LED_Blink project

11. This project uses the NegVal method from the BitIO bean to invert the LED state. It is not generated by default. To configure it, go to the Methods tab in the Bean Inspector window and click the round arrow button ( ) for the NegVal method. Figure 23 shows the Methods tab with the changes proposed for the LED Blink project.

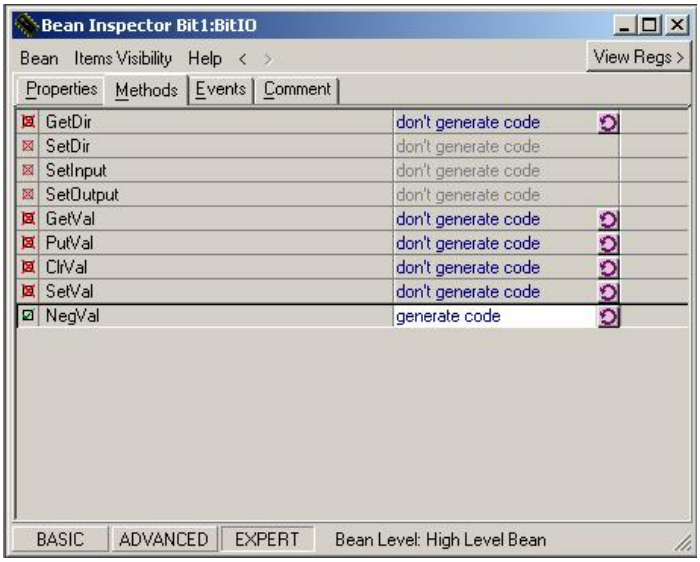

Figure 23. Bean Inspector Window - BitIO Methods Configuration on LED_Blink Project

12. Now the TimerInt and BitIO beans are created.
13. Every time the TimerInt happens use the method and events of these beans to make the LED change.
14. Compile the project pressing the F7 key.

15. Edit the function executed on the OnInterrupt event. In the file Events.c there is a function named TI1_OnInterrupt. This file is in the User Modules folder on the left panel. The code to be executed must be placed inside the TI1_OnInterrupt function.
16. If needed drag-and-drop the NegVal method inside the TI1_OnInterrupt function. The NegVal method is inside the BitIO bean, as shown in Figure 24.

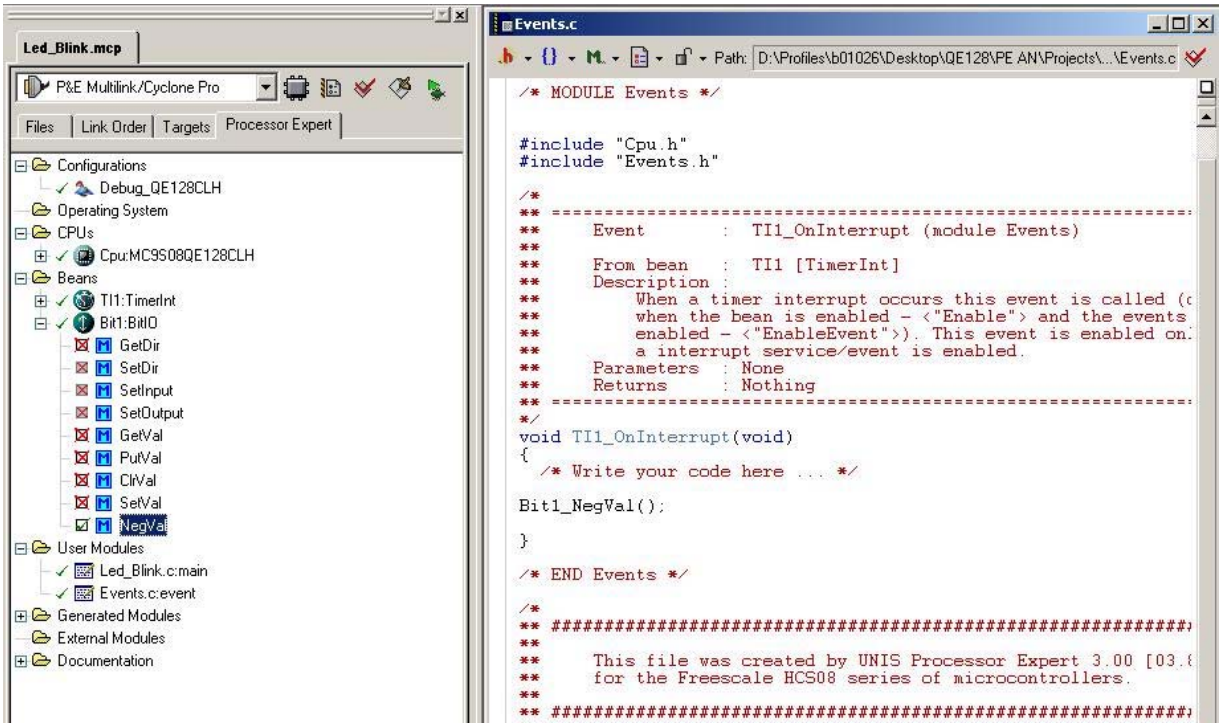

Figure 24. Events.c file -TI1_OnInterrupt Function on LED_Blink Project

17. The LED Blink project is ready.
18. To program the board, run and debug the project, refer to Section 3.7 Running and Debugging the project.
19. Test the project on the DEMOQE128 board. Connect the board to the PC and press the debug button as shown in Figure 25.

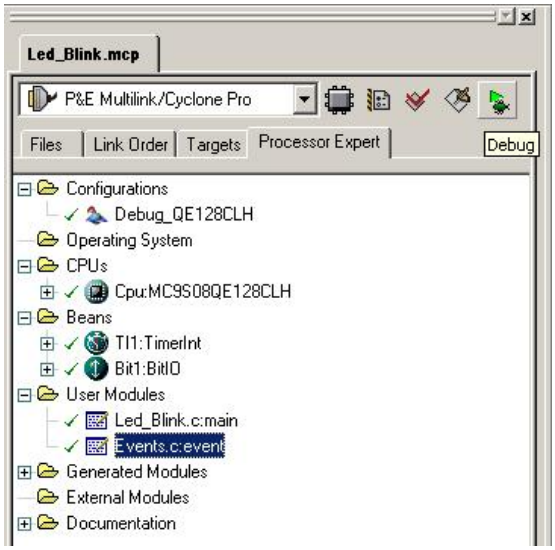

Figure 25. Debug - LED Blink Project

3.5 Lab2: ADC - Analog to Digital converter

This lab uses an ADC channel and an I/O port to make the LED (switch) change its state. This depends on the potentiometer resistance value. The ADC reads the voltage value generated by the potentiometer. When the value is more than half, the LED is ON. When the value is less than half, the LED is OFF.

The I/O peripheral is explained in Section 3.2.3. For the ADC, used related beans description and configuration there is a brief description. The Lab is then presented with a configuration guide line integrating the peripherals beans.

3.5.1 ADC: Peripheral Description

The analog-to-digital converter (ADC) is a converter that changes an analog voltage to a digital value. The 12-bit ADC is a successful design that operates within an integrated MCU system-on-chip.

Features of the ADC module include:

- Linear approximation algorithm with 12 bits resolution.
- Up to 28 analog inputs.
- Output formatted in 12, 10 or 8-bit right-justified format.
- Single or continuous conversion and an automatic return to idle after single conversion.
- Configurable sample time and conversion speed/power.
- Conversion complete, flag and interrupt.
- Input clock selectable, up to four sources.
- Operation in wait or stop3 modes for lower noise operation.
- Asynchronous clock source for low noise operation.
- Selectable asynchronous hardware conversion trigger.
- Automatic compare with interrupt for less-than, greater-than or equal-to programmable value.

For more detailed and specific data, visit product documentation MC9S08QE128 and MCF51QE128 at www.freescale.com.

3.5.2 ADC: ADC Bean

For the LED ADC application use the ADC bean. This bean implements the internal ADC peripheral functionality. Properties of the ADC bean are defined to provide an initialization code and selectable methods for the bean runtime API. The properties define all settings for the ADC operation mode.To use the ADC bean, you first need to add it to your project and then configure it, the following steps will show how to do that.

Adding an ADC bean the project:

1. Create a project according to Section 3.1 Creating a project with Processor Expert.
2. Go to menu Processor Expert > View > Bean Selector in Code Warrior and add the ADC bean (Figure 26).
3. Select the tab Categories, expand the folder CPU Internal Peripherals, the Converter, and the ADC folders.
4. Inside the ADC folder are all the beans related to the ADC peripheral.
5. Double-click the ADC bean to add it to your project.

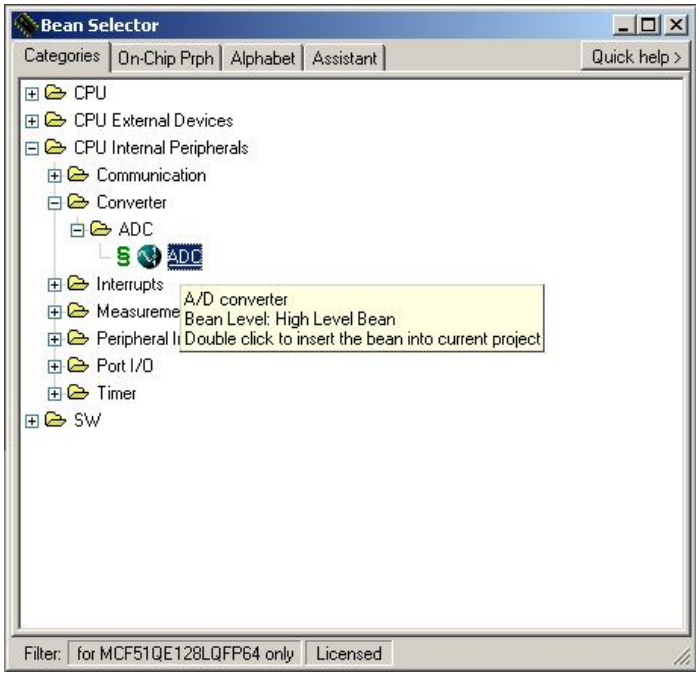

Figure 26. Bean Selector Window - ADC Bean

These steps describe how to configure the TimerInt bean:

1. In the Processor Expert tab on the left panel, in the Beans folder, the new ADC bean is displayed (Figure 27).
2. Double-click the new bean, and the Bean Inspector window activates. There you can configure the bean parameters, and to enable/disable the Methods and Events creation.

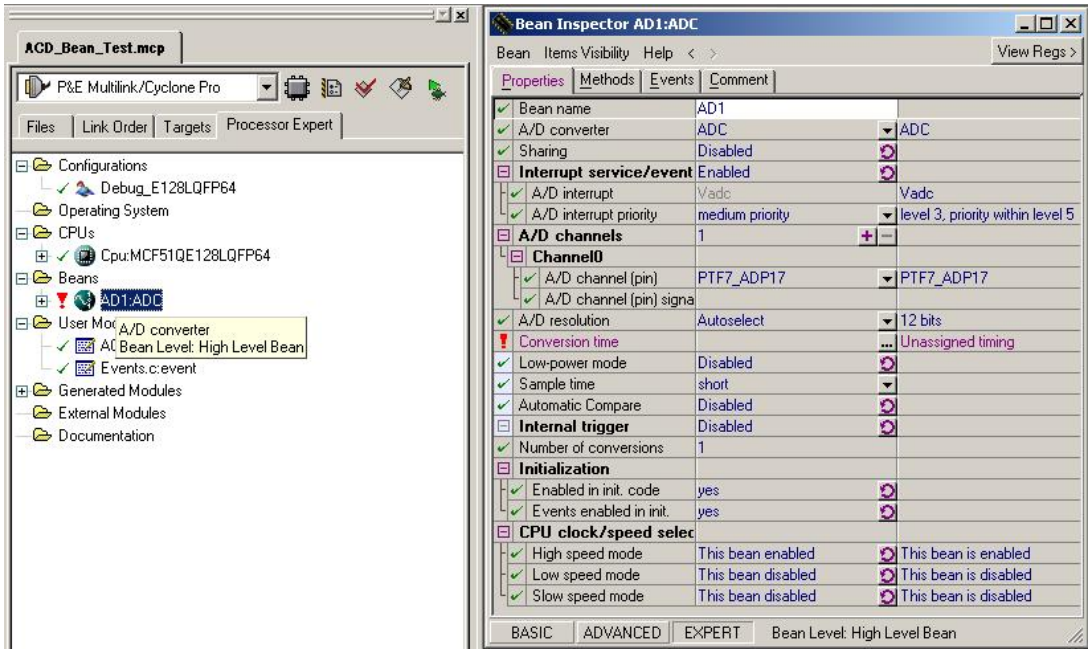

Figure 27. Bean Inspector window - ADC Bean

NOTE

Remember always choose the Expert mode. In this mode are all the available configurations for the bean.

3. To configure the ADC bean, select the Properties tab and modify the properties values. Below is a list and description of the most important properties for the ADC bean.
   - Interrupt service/event -- If this property is set to Enabled, the bean functionality depends on the interrupt service and does not operate if the CPU interrupts are disabled.
   - A/D channels -- List of the pins used by the A/D converter. You may add/delete a pin item with the +/- buttons, and select a pin for each item with the roll-down menu. Each item of the channel list has the following fields:
     - Channel0 -- Number of the channel.
     - A/D channel -- Selected A/D channel or pin name.
   - A/D resolution -- Maximum data resolution required by application 8, 10 or 12-bit.
   - Conversion time -- Time of one conversion. It is necessary to type both a value and a unit. The setting may be made with the help of the Timing dialog box. It opens when clicking on the button (...).
   - Sample time -- Select length of the sample time in units of ADC conversion clock cycles. This selection affects the total conversion time and the A/D conversion accuracy. A longer sample time results in better accuracy.
   - Number of conversions -- Number of conversions for one measurement to calculate an average value.

4. Notice there is an error in the Conversion Time field, the default value is empty this field is mandatory. Complete this field with any value, click the "°¦" button to see all the possible values. The error should now disappear.
5. If needed select the Methods to create by selecting the Methods tab (Figure 28) in the Bean Inspector window. Click the round arrow button ( ). For this bean, the methods that can be used are:
   - Start -- This method starts continuous conversion on all channels that are set in the bean inspector. When the measurements on all channels have finished the OnEnd event may be invoked.
   - Stop -- This method stops the continuous measurement.
   - MeasureChan -- This method performs measurements on one channel.
   - GetValue -- Returns the last measured values for all channels. The format and width of the value is a native format of the A/D converter.
   - GetChanValue -- Returns the last measured value of the required channel. The format and width of the value is a native format of the A/D converter.
   - GetChanValue16 -- This method returns the last measured value of the specified channel in a 16-bit word, justified to the left (even if the AD resolution is less than 16-bit). The user code dependency on AD resolution is eliminated.

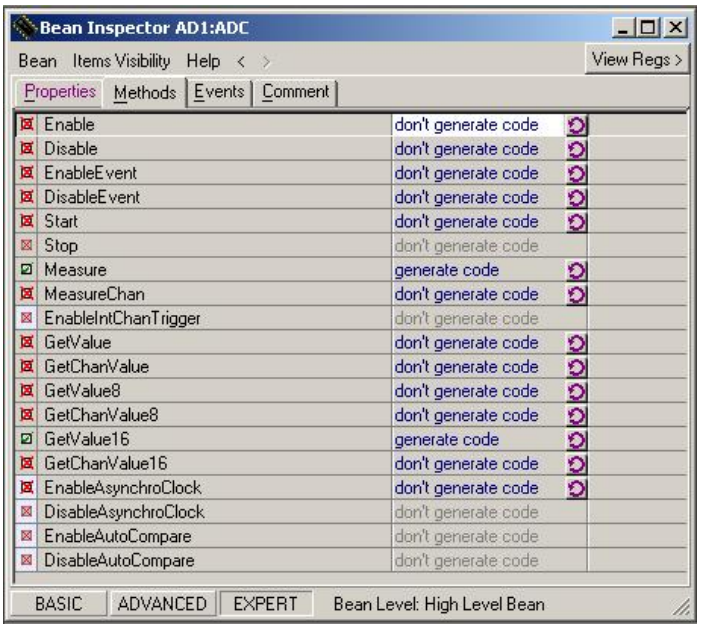

Figure 28. Bean Inspector window - Methods for ADC Bean

6. Set the Events configuration for this bean. Select the Events tab (Figure 29) in the Bean Inspector window. For the ADC bean the event OnEnd is very important. Once this event is enabled, every time the ADC completes a conversion, an event occurs.

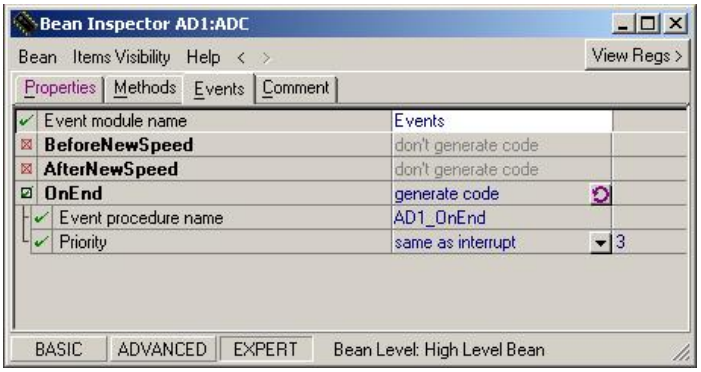

Figure 29. Bean Inspector Window - Events for ADC Bean

7. Compile the project pressing the F7 key.
8. Edit the function executed on the OnEnd event. In the file Events.c there is a function named AD1_OnEnd. This file is in the User Modules folder on the left panel. The code to be executed must be placed inside the AD1_OnEnd function (Figure 30).

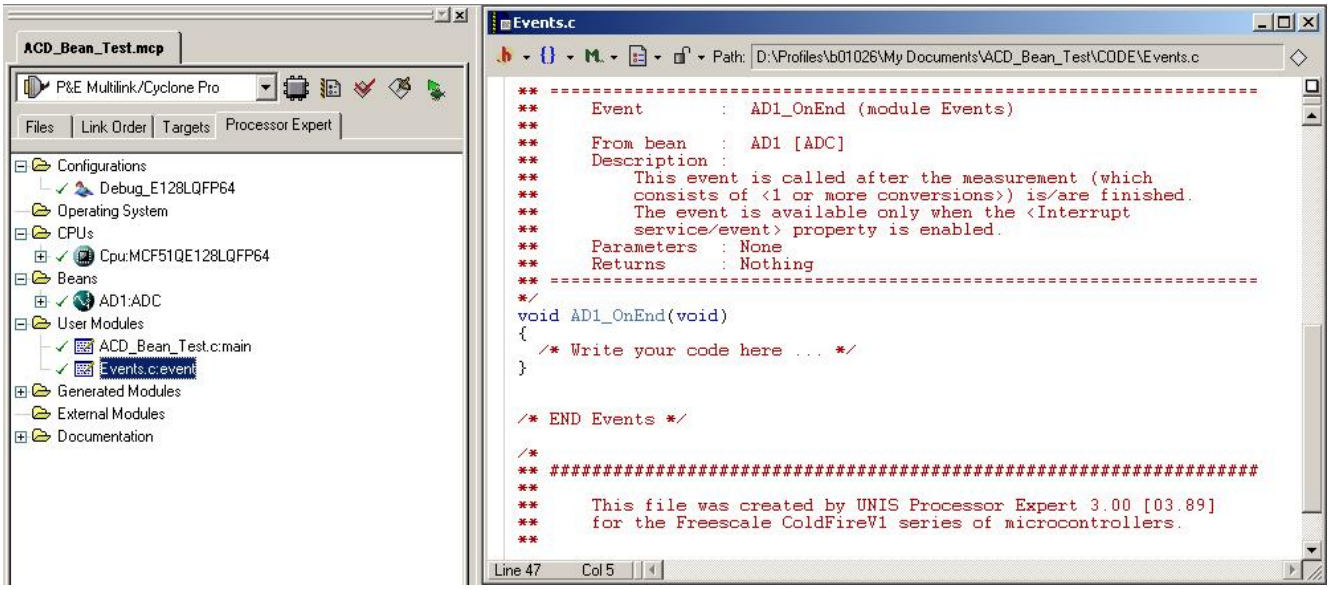

Figure 30. Events.c File for ADC Bean

3.5.3 Lab2: LED ADC - Project configuration

This project is implemented for the DEMOQE128 Board.

For the LED ADC application, use the ADC bean and the BitIO bean. The ADC reads the voltage value generated by the potentiometer. It is connected to the port PTA0 and if the value is more than half of the maximum value read by the potentiometer, the LED (PTC0) is ON. If the read value is less than half, the LED is OFF.

Figure 31 shows the Bean Inspector window for ADC Bean with the properties values for the LED ADC project. The following steps describe how to configure the LED Blink project:

1. Create a project according to Section 3.1 Creating a project with Processor Expert. Create it with the CPU configured to run with the Internal Clock at the default speed.
2. The ADC bean and the BitIO bean must be added to your project. Refer to Section 3.5.2 ADC: ADC Bean and Section 3.4.4 I/O: BitIO Bean.
3. Once the ADC bean is inserted to the project, it appears on the left panel in the Processor Expert tab, in the Beans folder.
4. Configure this bean to generate conversions with the PTA0 port voltage value.
5. Double-click the ADC bean, and the Bean Inspector window activates.
6. In the Bean Inspector window configure the following:
   - Select the PTA0 in the A/D channel field.
   - Select the value 5.484 µs in the field Conversion Time by clicking the "..." button.
   - In the Methods tab, enable the "Start" method by clicking the round arrow button ( ).

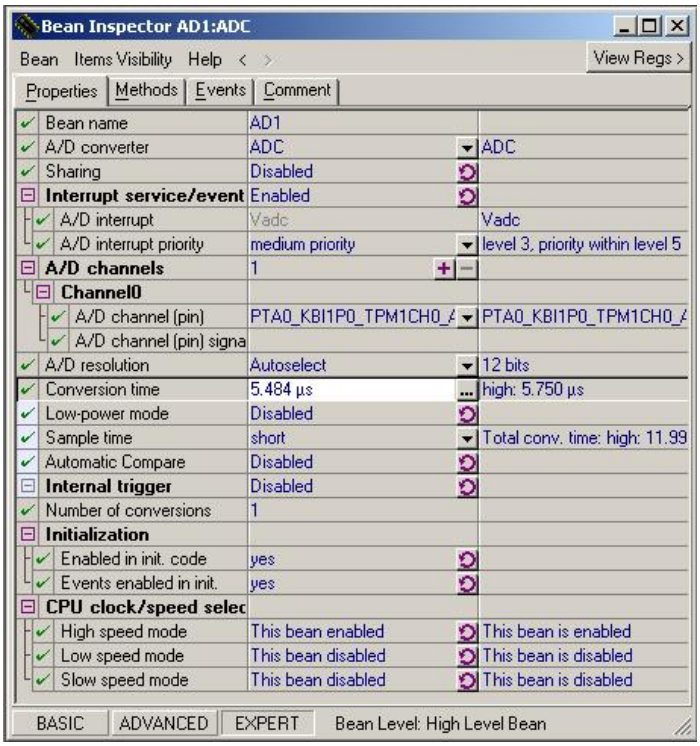

Figure 31. Bean Inspector Window - ADC Configuration on LED_ADC Project

7. The LED ADC project uses the ADC event OnEnd. This event is configured by default.
8. Configure the BitIO bean to control the pin PTC0, and set it as an output pin.
9. Double-click the BitIO bean, and the Bean Inspector window activates (Figure 32).
10. In the Bean Inspector window configure the following:

— Set the Pin for I/O value to PTC0. This pin is multiplexed with other peripherals, its name has also the KBI, TPM and AD functions.

— Set the Direction to Output.

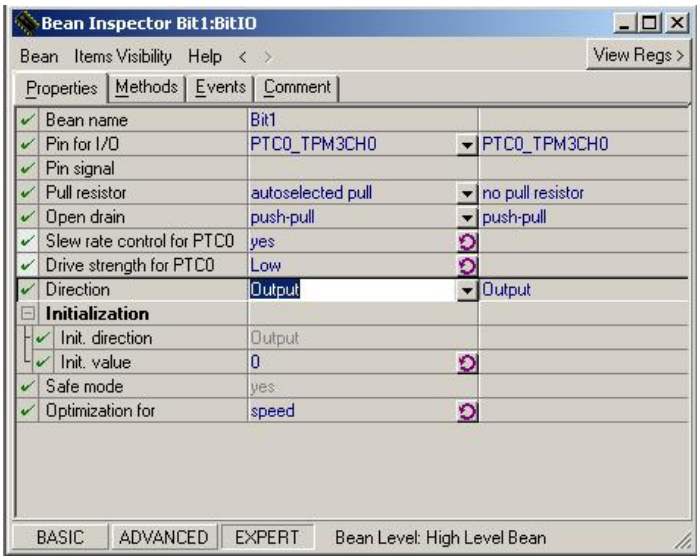

Figure 32. Bean Inspector Window - BitIO Configuration on LED_ADC Project

11. Integrate the ADC and BitIO beans created in order to make the LED state change. The LED state changes every time the value read from the ADC varies.
12. Edit the function executed on the OnEnd event. In the file Events.c find a function named OnEnd. This file is under the User Modules folder on the left panel. The code to be executed when the interrupt generated by the ADC bean occurs, must be placed inside the OnEnd function.
13. Use the AD1_GetValue16 method to get the value from the ADC.
14. Write a small code that compares the value from the ADC with half of the maximum value. Use the AD1_GetValue16 method, this method always returns values between 0x0000 and 0xFFFF. The half value is 0x7FFF. To use the AD1_GetValue16 method, pass a parameter with a pointer to the place where the ADC value is stored for future use. In this example create a word type variable (16-bits) named AD_Result.
15. Use an if...then...else statement to compare the value.
16. Use the BitIO methods Bit1_ClrVal() and Bit1_SetVal() to turn the LED ON and OFF.
17. The code created must be placed inside the OnEnd function.
18. The final code is shown in Figure 33.

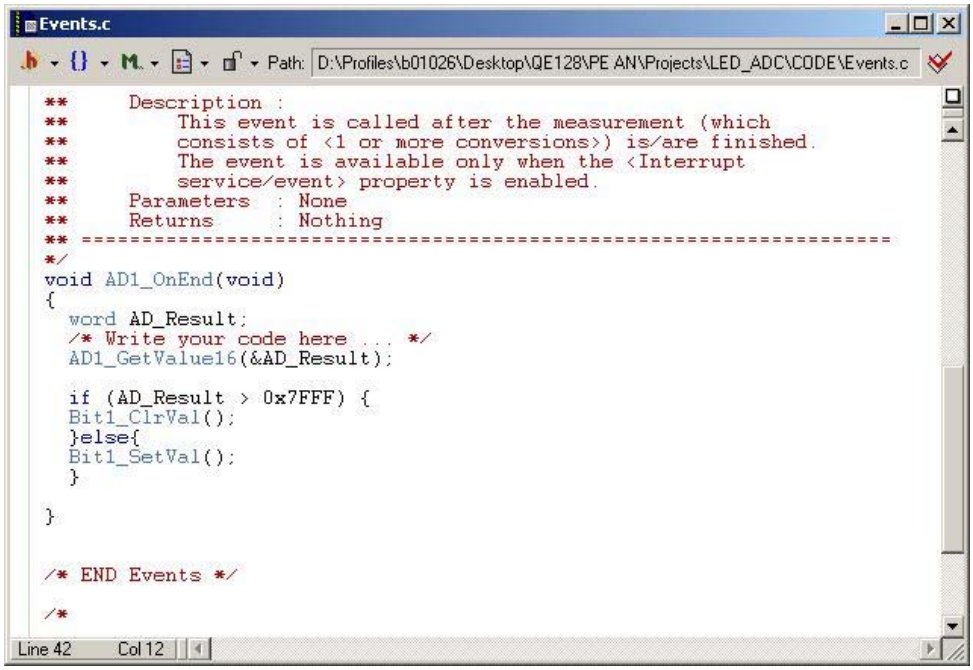

Figure 33. Events.c file -OnEnd Function on LED_ADC Project

19. By configuring the OnEnd event, the LED is controlled when the ADC finishes a conversion. The final step is to start the ADC conversions.
20. This project uses the continuous conversion mode. Every time an ADC conversion ends, another one begins. To start the ADC conversions in continuous mode, use the ADC Bean Start method. Place this method in the beginning of the code, inside the main() function.
21. The main() function is inside the LED_ADC.c file.
22. Add the Start method inside the main() function, before the infinite loop for(;;){}.
23. This code is shown in Figure 34.

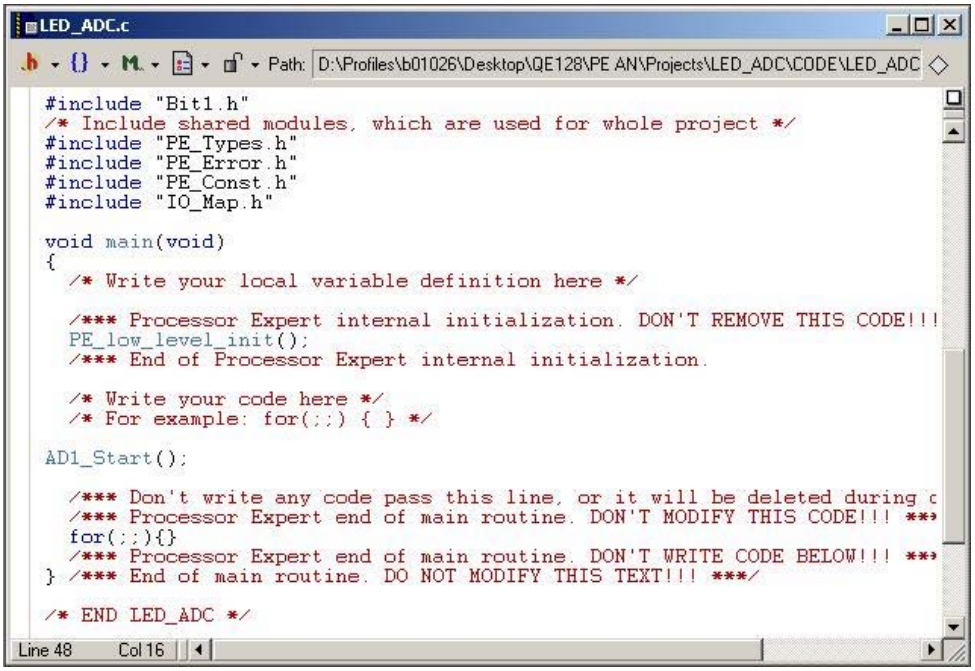

Figure 34. LED_ADC.c File -main Function on LED_ADC Project

24. The LED ADC project is ready.
25. To program the board, run and debug the project, refer to Section 3.7 Running and Debugging the project.
26. Test the project on the DEMOQE128 board. Change the potentiometer to see the LED0 turn ON and OFF.

3.6 Lab3: PWM - Pulse Width Modulation

This lab uses a PWM to control the brightness of the LED. The PWM duty-cycle is controlled by a potentiometer. Applying a voltage in an ADC channel, the ADC converts this value to digital to control the PWM duty-cycle.

The PWM uses a TPM timer. The peripheral has been explained in Section 3.2.1. The PWM bean is explained with details in the next section. This lab also uses an ADC to read the potentiometer position value. The ADC peripheral and the ADC bean are explained in Section 3.3.1.

For more detailed and specific data, visit product documentation MC9S08QE128 and MCF51QE128 at www.freescale.com.

3.6.1 Timer: PWM Bean

For the PWM_Lab application, use the PWM bean. This bean implements a PWM that generates a signal with a variable duty and fixed frequency.

To use the PWM bean, first add it to your project and then configure it. The following steps will show how to add a PWM bean to the project:

1. Create a project according to Section 3.1 Creating a project with Processor Expert.
2. Go to menu Processor Expert > View > Bean Selector in Code Warrior and add the PWM bean (Figure 35).
3. Select the tab Categories, expand the folder CPU Internal Peripherals, and the Timer folder.
4. Double-click the PWM bean to add it to your project.

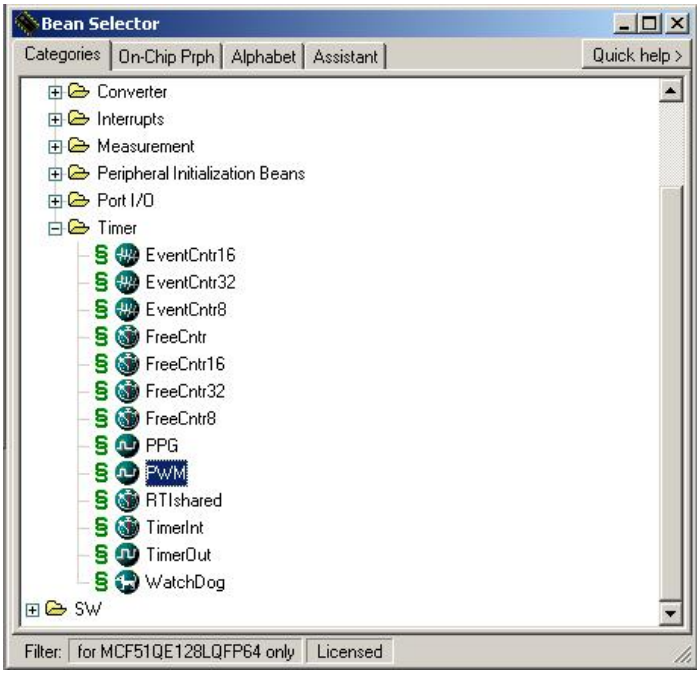

Figure 35. Bean Selector Window - PWM Bean

These steps describe how to configure the PWM bean.

1. The PWM bean is displayed in the Processor Expert tab on the left panel, in the Beans folder (Figure 36).
2. Double-click the created bean. The Bean Inspector window activates. There you can configure the bean parameters, to enable/disable the Methods and Events created.

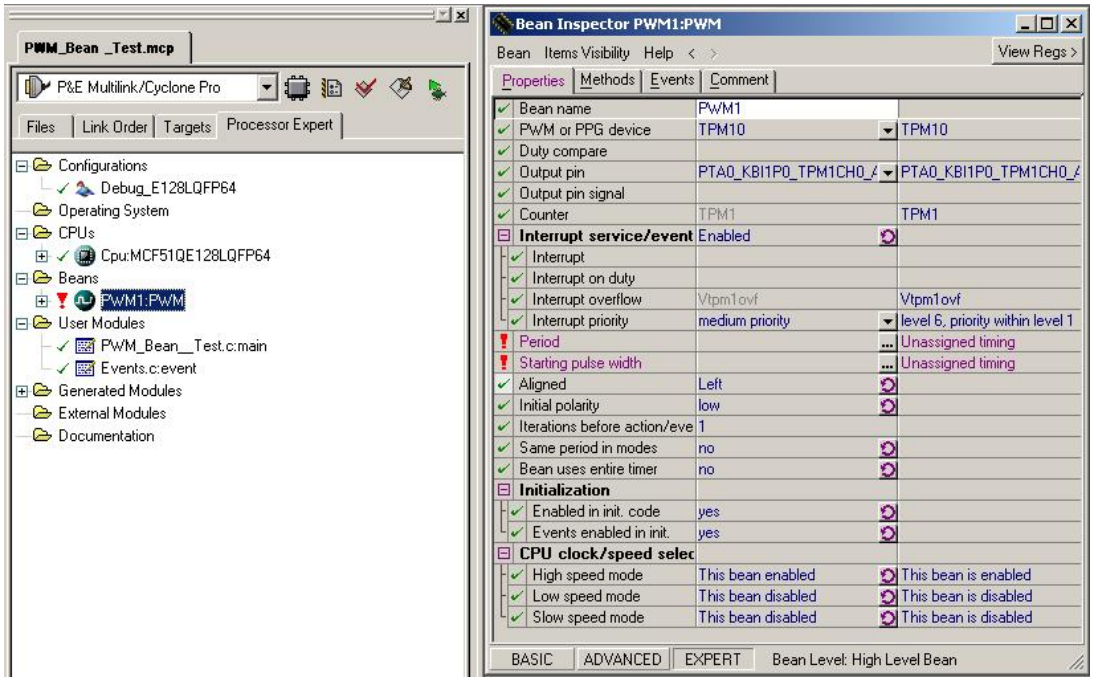

Figure 36. Bean Inspector Window - PWM Bean

NOTE

Remember always choose the Expert mode. In this mode are all the available configurations for the bean.

3. To configure the PWM bean. Select the Properties tab and modify the properties values. Below is a list and description of the most important properties for the PWM bean.
   - PWM or PPG device -- Pulse Width Modulation compare device or Programmable Pulse Generation period compare/reload device.
   - Output pin -- Pin used for output of the generated signal.
   - Prescaler -- Prescaler selected for the timer.
   - Period - Period of the output signal. It is necessary to specify both a value and a unit. The setting can be made with the help of the Timing dialog box that opens when clicking on the button (...).
   - Starting pulse width -- Starting pulse width specifies the length of time the output signal spends in the active level during the output cycle. The active level is defined in the Initial polarity. It is necessary to specify both a value and a unit (see Timing Setting Syntax WHERE). The setting can be made with the help of the Timing dialog box that opens when clicking on the button (...).
   - Initial polarity -- Initial polarity of the output signal: 0 = Low, 1 = High. It is possible to change it at runtime using methods: ClrValue and SetValue.
4. Configure the Period and Starting Pulse Width fields. Complete these fields with values, click the (...) button to see all the possible values.
5. Select which Methods to create by selecting the Methods tab (Figure 37) in the Bean Inspector window. Click the round arrow button ( ). For this bean, the methods that used are:

— Enable -- Starts the signal generation.

— Disable -- Stops the signal generation and events calling.

— SetRatio16 -- This method sets a new duty-cycle ratio. Ratio is expressed as a 16-bit unsigned integer number. 0 - FFFF value is proportional to ratio 0 - 100%.

NOTE

Calculated duty depends on the timer possibilities and on the selected period.

— SetDutyUS -- This method sets the new duty value of the output signal. The duty is expressed in microseconds as a 16-bit unsigned integer number.

— SetValue -- Sets the output pin value. This method only works when the timer is disabled.

— ClrValue -- Clears the output pin value. This method only works when the timer is disabled.

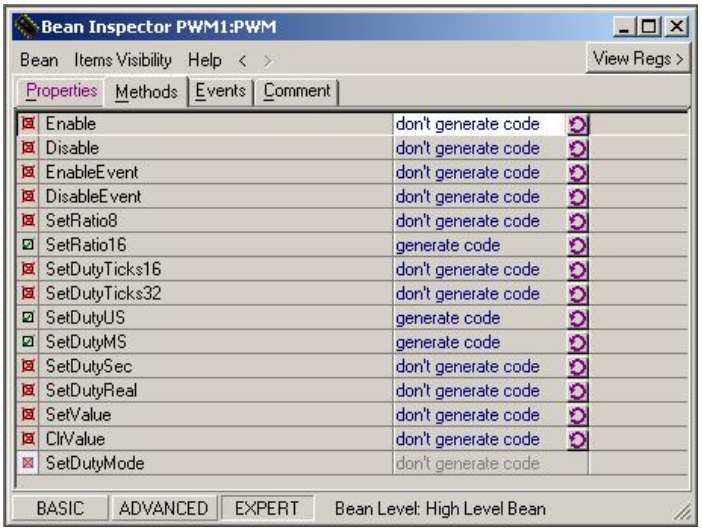

Figure 37. Bean Inspector Window - Methods for PWM Bean

6. Set the events configuration for this bean. Select the Events tab in the Bean Inspector window (Figure 38). For the PWM bean the event OnEnd is very important. Once this event is enabled, the PWM completes a cycle.

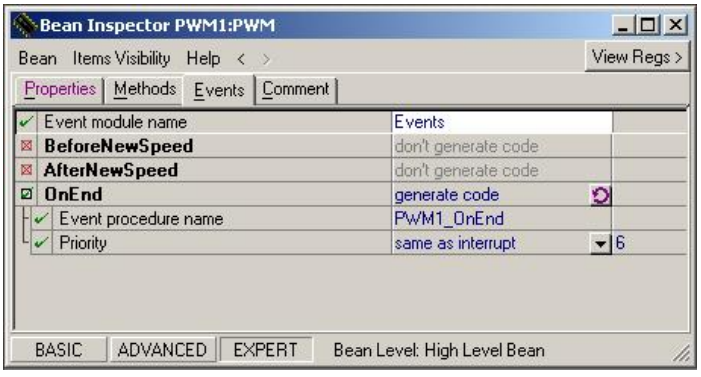

Figure 38. Bean Inspector Window - Events for PWM Bean

7. Compile the project by pressing the F7 key.
8. Edit the function executed on the OnEnd event. In the file Events.c there is a function named PWM1_OnEnd. This file is in the User Modules folder on the left panel. The code to be executed must be placed inside the PWM1_OnEnd function (Figure 39).

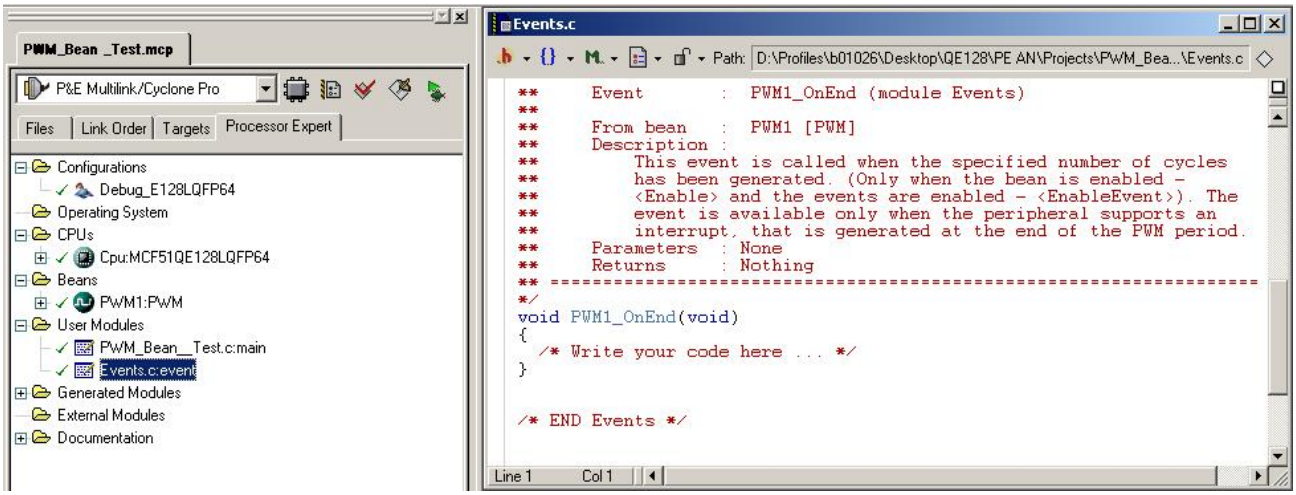

Figure 39. Events.c file for PWM Bean

3.6.2 Lab3: PWM Lab - Project configuration

This project is implemented for the DEMOQE128 Board.

Use the ADC bean and the PWM bean for the PWM Lab application. The ADC bean reads the voltage value generated by the potentiometer connected to the port. The PWM bean generates a PWM wave which its duty-cycle is controlled by the ADC read value. The PWM output controls a LED. The LED brightness changes according to the PWM duty-cycle applied.

The following steps describe how to configure the LED Blink project:

1. Create a project according to Section 3.1 Creating a project with Processor Expert. Create it with the CPU configured to run with the Internal Clock at the default speed.
2. The ADC bean and PWM bean must be added to the project. Refer to Section 3.5.3 Lab2: LED ADC - Project configuration and Section 3.6.1 Timer: PWM Bean.
3. Once the ADC bean is displayed on the left panel in the Processor Expert tab, in the Beans folder (Figure 40).
4. Configure this bean to make conversions with the PTA0 port voltage value.
5. Double-click the TimerInt bean and the Bean Inspector window activates.
6. In the Bean Inspector window configure the following:
   - Select the PTA0 in the A/D channel field.
   - Select the value 5.484 µs in the field Conversion Time by clicking the ("...") button.

Figure 40 shows the Bean Inspector window for the ADC Bean with the properties values of the PWM_Lab project.

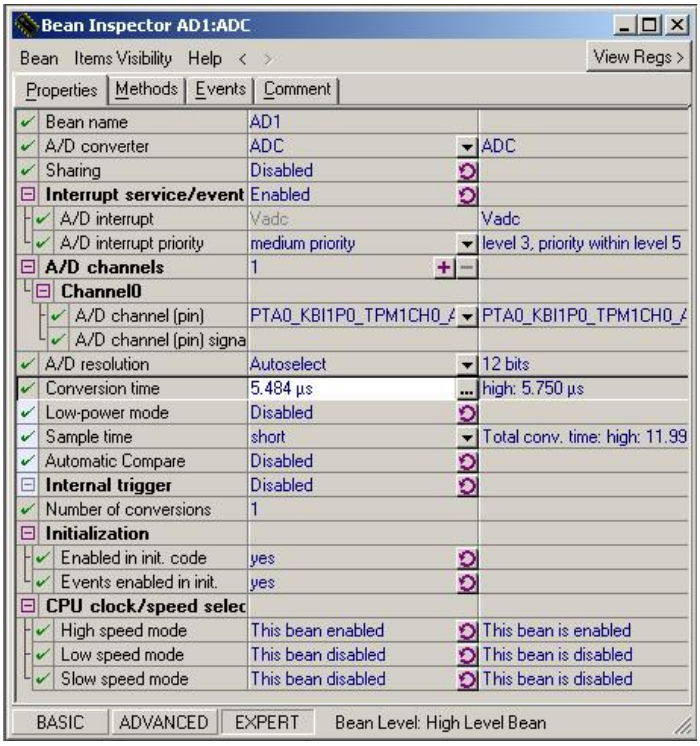

Figure 40. Bean Inspector Window - ADC Configuration on PWM Lab Project

7. The PWM_Lab project uses the ADC Event OnEnd.
8. The ADC bean is inserted and configured. The PWM bean is configured.
9. The PWM bean is displayed on the left panel, in the Processor Expert tab, in the Beans folder.
10. Configure this bean to control the pin PTC0, this pin is associated with TPM3 Channel 0 (TPM30). Also, configure the PWM period to 10 ms and the initial value to 0.
11. Double-click the PWM bean and the Bean Inspector window is activated.
12. In the Bean Inspector window configure the following (Figure 41):
    - Select the PWM or PPG device as TPM30 in order to select the Channel connected to the pin PTC0.
    - Set the Period value to 10 ms.
    - Set the Starting Pulse Width to 0.

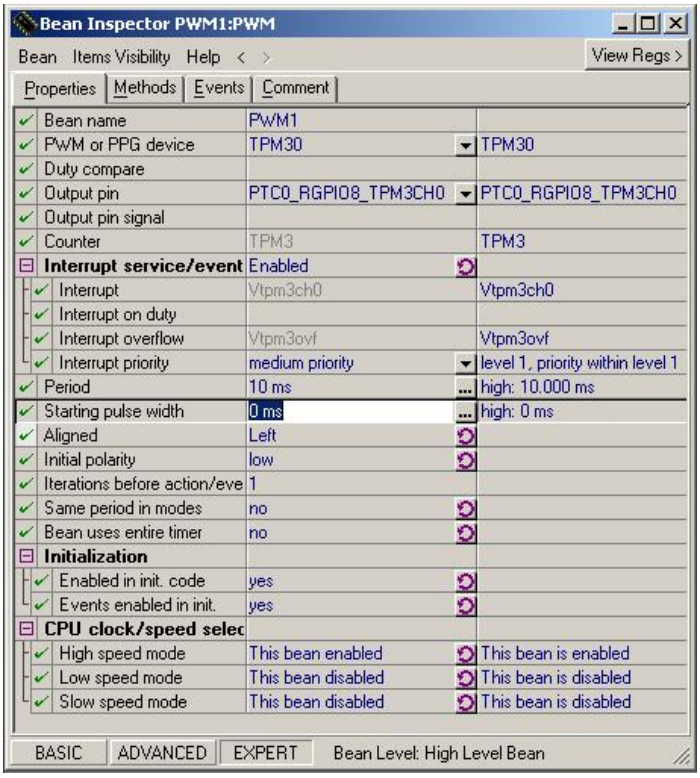

Figure 41. Bean Inspector Window - PWM Configuration on PWM Lab Project

13. The PWM_Lab project uses the PWM Event OnEnd. This event is configured by default.
14. Integrate the PWM and ADC beans created, in order to make the LED brightness change. This depends on the value read from ADC. Edit the two functions executed on the OnEnd event of the ADC and PWM. In the file Events.c find two functions named OnEnd, one is ADC1_OnEnd and the other PWM1_OnEnd. This file is in the User Modules folder on the left panel. The ADC1_OnEnd event occurs when an ADC conversion is finished, and the PWM1_OnEnd occurs when a PWM cycle ends. In this application, every 10 ms.
15. Configure the PWM1_OnEnd event to start an ADC read, and the ADC1_OnEnd to set the duty-cycle with the value.
16. Use the AD1_Measure to start a read in the ADC.
17. Use the AD1_GetValue16 method to get the read value from the ADC.
18. Use the PWM1_SetRatio16 to set the PWM duty-cycle.
19. Create a word variable to store the ADC read value.
20. The final code is shown in Figure 42.

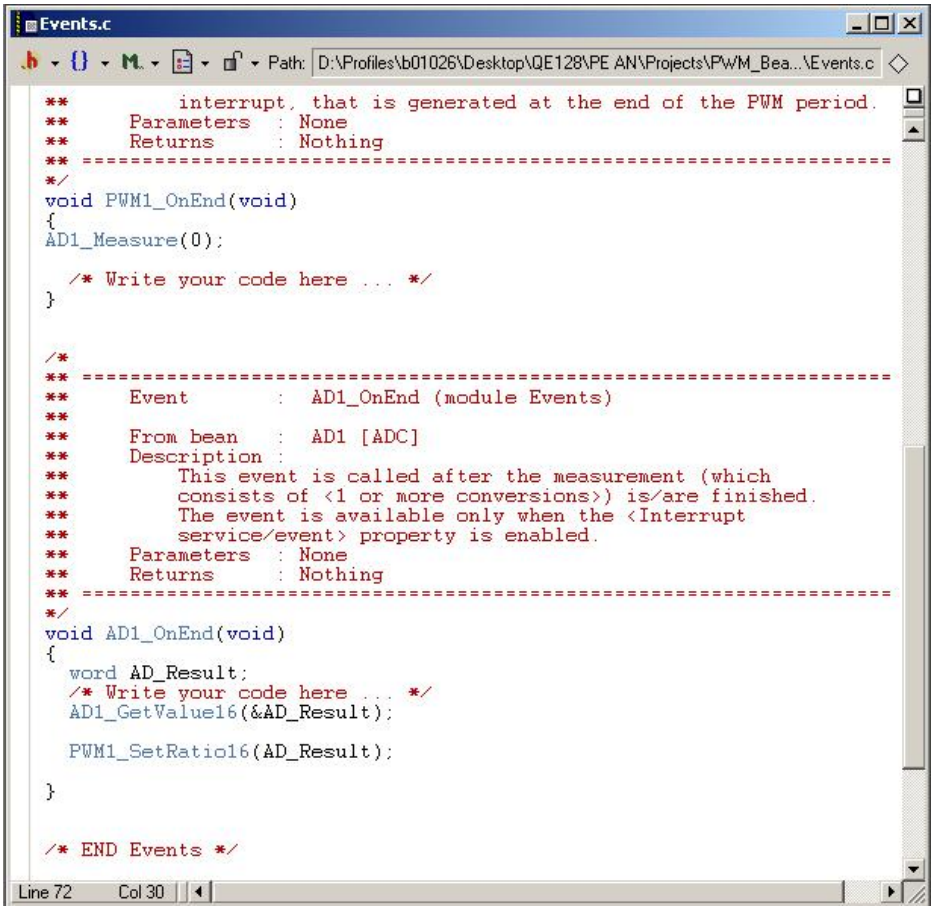

Figure 42. Events.c file -OnEnd Functions on PWM_Lab Project

21. The PWM Lab project is ready.
22. To program the board, run and debug the project. Refer to Section 3.7 Running and Debugging the project.
23. Test the project on the DEMOQE128. Change the potentiometer to see the LED0 brightness change.

3.7 Running and Debugging the project

This section explains how to run the project on the board and the basic debug functions. Once a project is created click the Debug button in the project tab inside CodeWarrior. The Connection Manager window opens (Figure 43). This window presents the Interface for connection with the board, in case that the board is connected to the PC, its description is shown in the field Port.

Check if the board is correctly connected and click the Connect button.

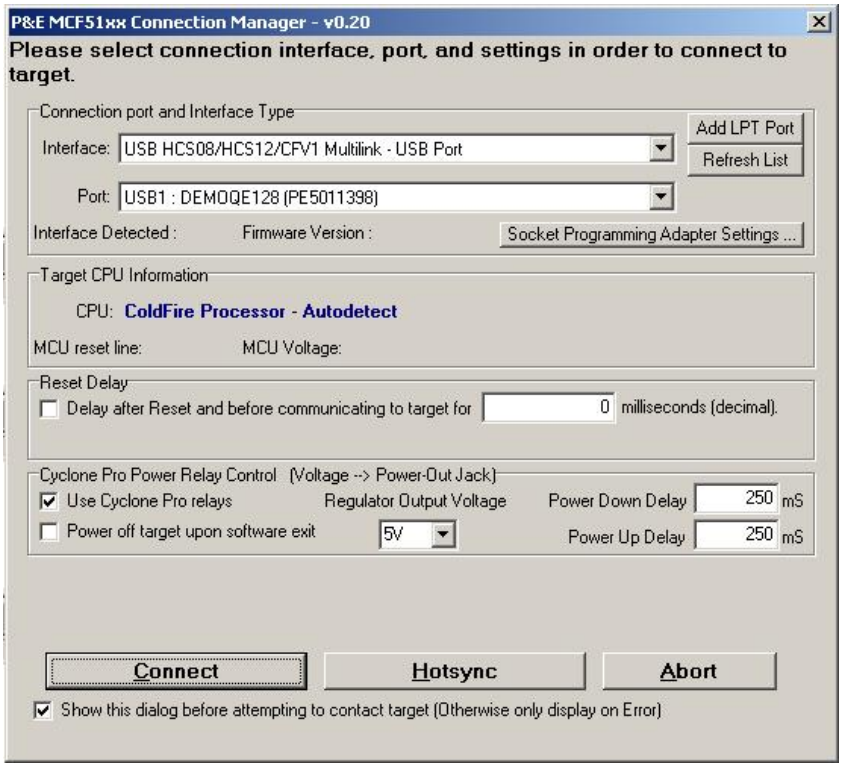

Figure 43. Connection Manager Window

A message asking to erase the Flash might pop-up, click ok to erase the flash and program it with the project.

After the MCU is erased and re-programed, the True-Time Simulator & Real-Time Debugger window opens (Figure 44). In this window, debug your software. There are some windows inside the main window, like the corresponding code in Assembly, variables values, full memory map, CPU register values and command log lines, in these windows see the actual execution point of the code.

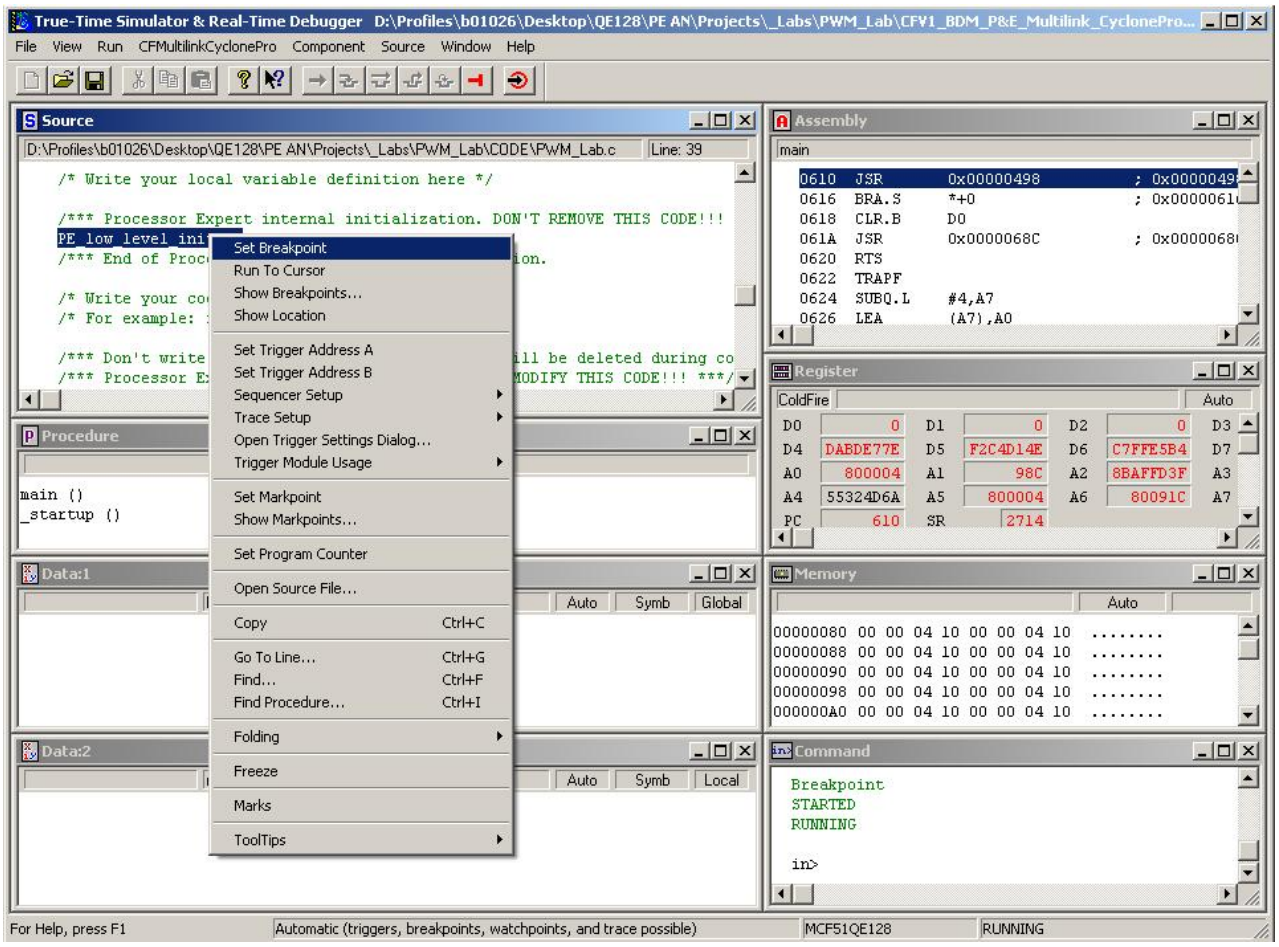

Figure 44. Real-Time Debugger Window

In order to debug the code, use the following commands: Run( ), Single Step ( ), Step Over ( ), Step Out ( ), Assembly Step( ), Halt( ) and Reset Target( ).

Place Breakpoints in the code to better debug it. To do this right-click the code wanted and select the Set Breakpoint option.

4 Conclusion

The costs of development using MCUs are increasingly affected by the software content of the development projects and the complexity of modern MCUs. Part of these costs are often reflected in the learning curve associated with new development tools. This additional cost is important enough for developers to reject the most suitable MCU for an application. It would take too much effort and time to learn a new set of tools. Software engineers need a high-performance development environment designed to use all of the capabilities of the MCU architecture, while simultaneously minimizing the underlying complexity. It is critical that MCU software development tools provide the same level of ease-of-use and extensibility as mainstream microprocessor tools, in order to minimize the new product introduction cycle. CodeWarrior development tools, with Processor Expert, provide an integrated suite of development tools. These tools are designed to work together to dramatically simplify applications software development.

Together, the Freescale Flexis MCUs, CodeWarrior development tools and Processor Expert take users to a new level of MCU-based development. Users can easily develop and migrate their applications between the 8-bit and 32-bit devices seamlessly.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3467
Rev. 0
05/2007

freescale™
semiconductor