Freescale Semiconductor
Application Note

Document Number:AN4769
Rev 0, 08/2013

Integrating a Processor Expert Driver (LDD) into a Non-Processor Expert Project

by: Chris Brown

Contents

1 Introduction: Integrating a Processor Expert driver into a project....1
2 Processor Expert basics....2
2.1 CPU components....2
2.2 Embedded components....2
3 Processor Expert files....2
4 General procedure for integrating a Processor Expert driver into your project…..............3
5 Example driver integration....4
6 Conclusion....26
7 References....26
8 Revision history....27

1 Introduction: Integrating a Processor Expert driver into a project

There are many reasons for adding a single Processor Expert (PEx) driver into an application.

- Lack of time to develop drivers for the product.
- Complications involved in the module which requires drivers
- Future plans to integrate an operating system into the application

Whatever the reason for adding drivers to a project, integrating PEx drivers into an application, that is not a dedicated PEx project, can be a non-trivial task.

This application note discusses the process and techniques for integrating a single Processor Expert driver into a "baremetal" project. An example of such integration is also included for reference. The example application was built and tested using IAR 6.50.6 Embedded Workbench and targets TWR-K60N512 tower board. The PEx driver files were created using Processor Expert Driver Suite v10.0.2.

2 Processor Expert basics

Before discussing how to integrate a PEx driver into an application, the users must first have a basic understanding of Processor Expert and the files it generates. Processor Expert is a development system to create, configure, optimize, migrate, and deliver software components which can generate source code for Freescale silicon. Users interact with a graphical interface to configure the desired components for their project. Processor Expert is available as part of the CodeWarrior tool suite or as an Eclipse-based plug-in feature for installation into an independent Eclipse environment.

With any Processor Expert project, there are essentially two different types of components to will deal with: CPU components and embedded components. These two components are examined in the following subsections.

2.1 CPU components

CPU components specify the device which the project targets such as MKL25Z128VLK4, MK60N512VMD100, and MK40X256VMD100. There can be only one active CPU component in a project. Within the CPU component, the user configures the core oscillator, internal oscillators, interrupt settings, watchdog settings, power mode settings, and other global settings that the project may require. MCU modules associated with the CPU component are as follows:

- Multipurpose Clock Generator (MCG)
- System Integration Module (SIM)
- Power Management Controller (PMC)
- Oscillator (OSC)
- Low-Leakage Wake-Up Unit (LLWU)
- Watchdog (WDOG) (if applicable)

NOTE

This is not an all inclusive list. Some pins or other peripherals may be configured through the code generated by the CPU component.

2.2 Embedded components

Embedded Components comprise all of the MCU modules that are not configured by the CPU component. In general, only the peripheral modules are included in this category. There are three different types of embedded components:

- High/low level components
- Logical Device Driver (LDD) components
- Peripheral Initialization components

This application note examines the integration of the LDD component into a project as this component is more powerful than the peripheral initialization component and the building block of the high/low level components.

3 Processor Expert files

As discussed in Processor Expert basics, Processor Expert generates the files necessary to create an application. Therefore, it is necessary to understand the files generated and their specific function, before knowing which files are necessary or unnecessary to include in a project. This table presents the contents of each of the files generated by Processor Expert.

Table 1. Processor Expert file descriptions

File	Contents
CPU.c	Contains standard functions that configure the CPU such as memory, MCG, SIM
CPU.h	Contains definitions, structure declarations, and function declarations required by CPU.c.
IO_Map.h	Device-specific header file that defines registers and bitmasks
PE_Const.h	Defines masks for causes of reset and low-voltage detect
PE_Error.h	Defines PE-specific error masks
PE_LDD.c	Contains LDD-specific structure declarations and general functions that LDDs may require.
PE_Types.h	Defines standard types and macros common to LDDs
Vectors.c	Defines the reset and interrupt vectors
*_PDD.c	"Physical Device Driver" that contains module-specific definitions required by the LDDs

4 General procedure for integrating a Processor Expert driver into your project

The following steps provide a general procedure for integrating a PEx Logical Device Driver into your project. It is important to remember that every driver may not integrate exactly the same way into every project.

NOTE

The following instructions assume the user has a basic knowledge of PEx. A more detailed example is included for user convenience and for those that are new to Processor Expert.

1. Create a Processor Expert project.
   - Be sure to select the appropriate device when creating the project.
   - It is recommended to save the PEx project in the same folder as the tool chain project (or the folder where you plan to save the tool chain project) with a descriptive name.
2. Configure the PEx project.
   - Add the desired component to the project and configure it as desired.
   - As a general rule, the clock configurations should be set up exactly as they would be in your application. This is required only if the driver being created depends on the clock configurations of the part, such as a timer or a communications module.
3. Generate the code.
   - When generating the code, do not use the automated scripts. These scripts are useful when creating a full Processor Expert project, but can make things complicated when trying to use just one driver.
4. Add the generated code to the bare-metal project.
   - The following files will be required when integrating the newly created PEx driver:
     - Events.c
     - Events.h
     - PE_LDD.c
     - PE_LDD.h
     - PE_Types.h

  - <Component Name>.c
  - <Component Name>.h
- The associated Physical Device Driver (PDD) header file will also need to be included in the project. For example, if an ADC LDD was added to the project, the compiler tool that you using must be able to locate the file ADC_PDD.h.

NOTE

The PDD files are with your PEx build, located at <PEx Root Dir>\eclipse \ProcessorExpert\lib\Kinetis\pdd\inc.

5. Modify the driver files that have been added to the project.
   - In general, include files should be included through one common include file as much as possible. This avoids duplicated definitions of variables and constants.
   - Each driver file may be slightly different. So the required include files may change slightly for each driver.
   - None of the files added to the project will need to include Cpu.h, PE_Types.h, IO_Map.h, or PE_Const.h. These files are either not needed or the information they contain should already be included in the project.
   - All the functions in PE_LDD.c should all be removed except for the LDD_DeviceData definition (the driver functions will need this definition).
   - It may be necessary to add some definitions from some of the aforementioned files to the common files. These may differ from driver to driver.

5 Example driver integration

Now consider the example of adding an I2C LDD to an example from the Kinetis K60 sample code (KINETIS512_SC.zip package). For this example, a custom project has been created (named PEx_Drv_Int_Training) which is based on the Hello World project from the KINETIS512_SC code package. This project already includes code specific to using the I2C_LDD for convenience.

1. Create the Processor Expert project.
   a. Choose Start > All Programs > Freescale Processor Expert > PExDrv v10.2 >eclipse, to open Processor Expert 10.2.
   b. Select an empty folder for your workspace.
   c. Choose File > New > Processor Expert Project.

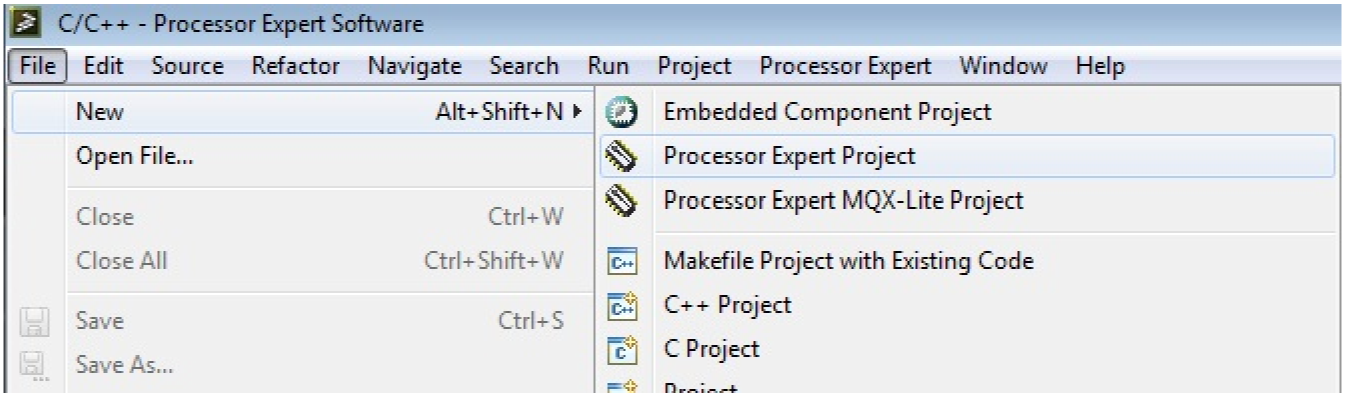
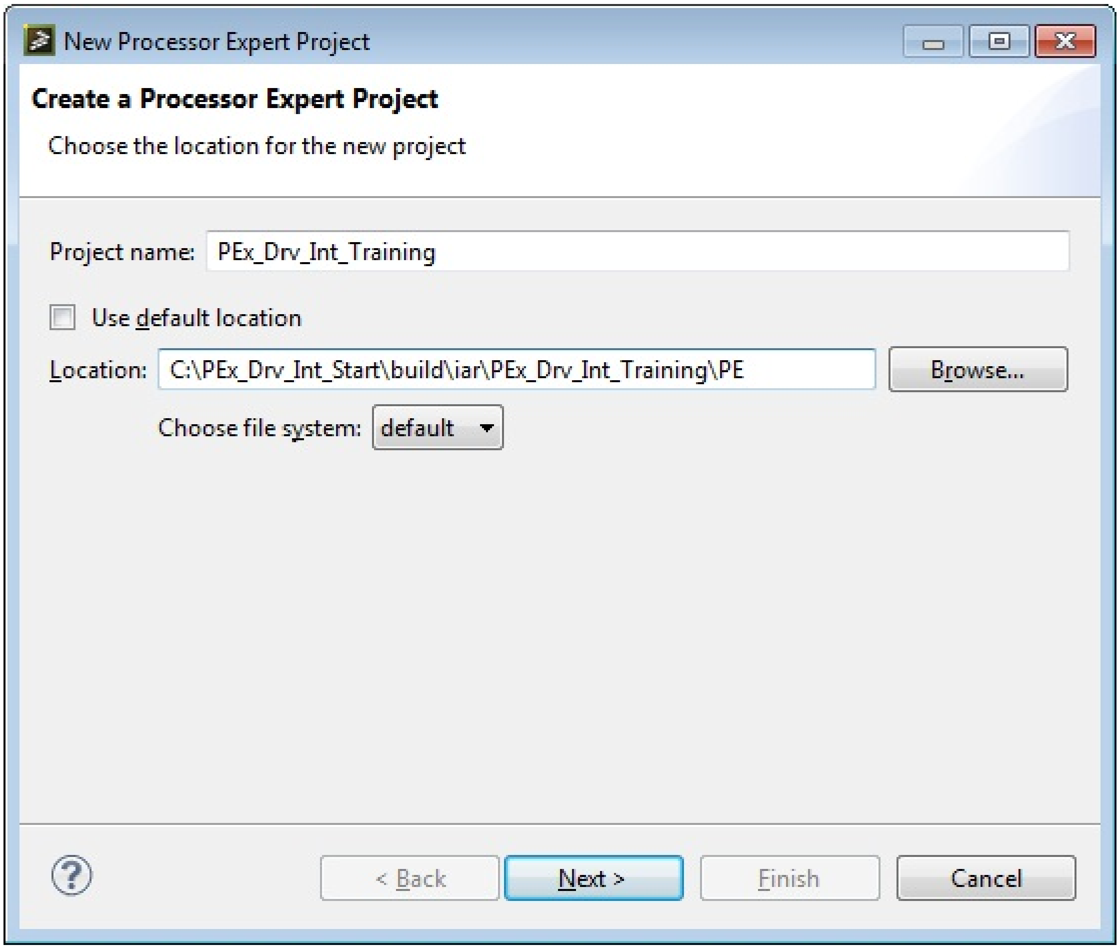


Figure 1. New Processor Expert project creation

   d. Deselect the "Use default location" option and set the Location and Project name as shown in the following figure (your PEx project should be stored in ..\build\iar\PEx_Drv_Int_Training\PE):



New Processor Expert Project

Create a Processor Expert Project

Choose the location for the new project

Project name: PEx_Drv_Int_Training

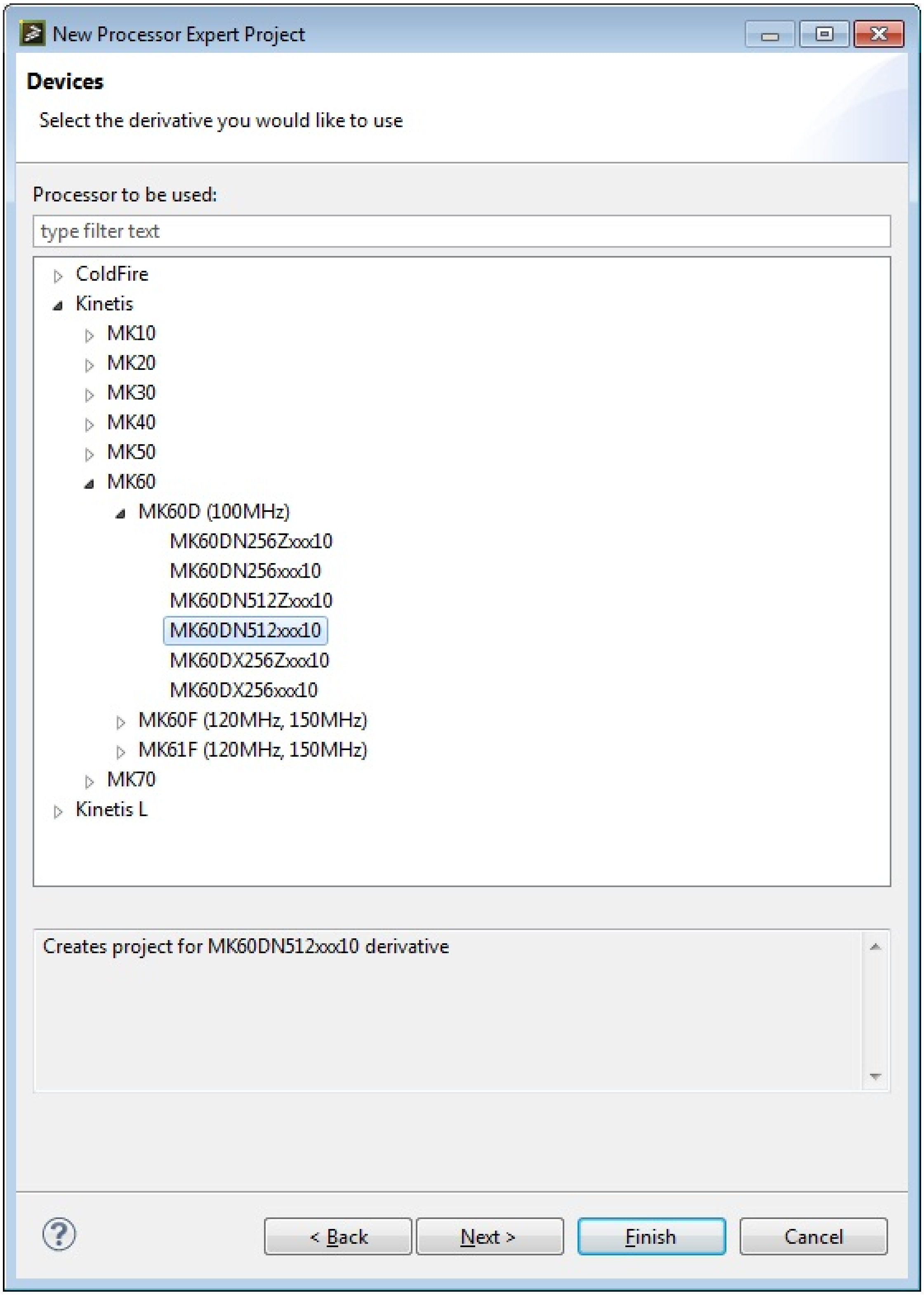
□ Use default location

Location: C:\PEx_Drv_Int_Start\build\iar\PEx_Drv_Int_Training\PE

Choose file system: default

Figure 2. Storing Processor Expert project

e. Select the appropriate device in the PE device selection dialog, as shown in the following figure.



New Processor Expert Project

Devices

Select the derivative you would like to use

Processor to be used:

type filter text

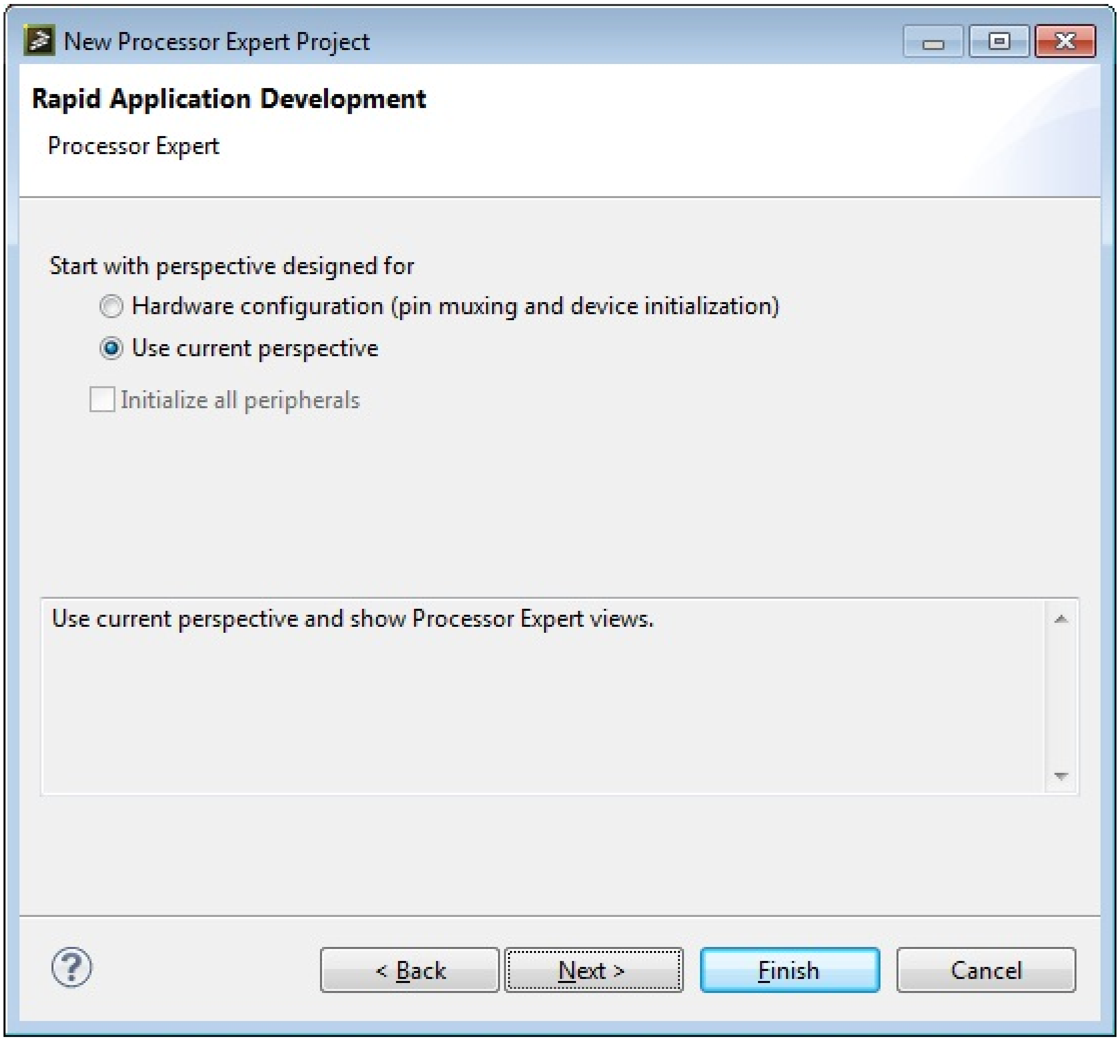
- ColdFire
- Kinetis
  - MK10
  - MK20
  - MK30
  - MK40
  - MK50
  - MK60
    - MK60D (100MHz)
      - MK60DN256Zxxx10
      - MK60DN256xxx10
      - MK60DN512Zxxx10
      - MK60DN512xxx10
      - MK60DX256Zxxx10
      - MK60DX256xxx10
    - MK60F (120MHz, 150MHz)
    - MK61F (120MHz, 150MHz)
  - MK70
- Kinetis L

Creates project for MK60DN512xxx10 derivative

Figure 3. PEx device selection dialog

f. In the PE tool selection dialog:

- For "Start with perspective designed for" select the "Use current perspective" option.



Rapid Application Development

Processor Expert

Start with perspective designed for

○ Hardware configuration (pin muxing and device initialization)

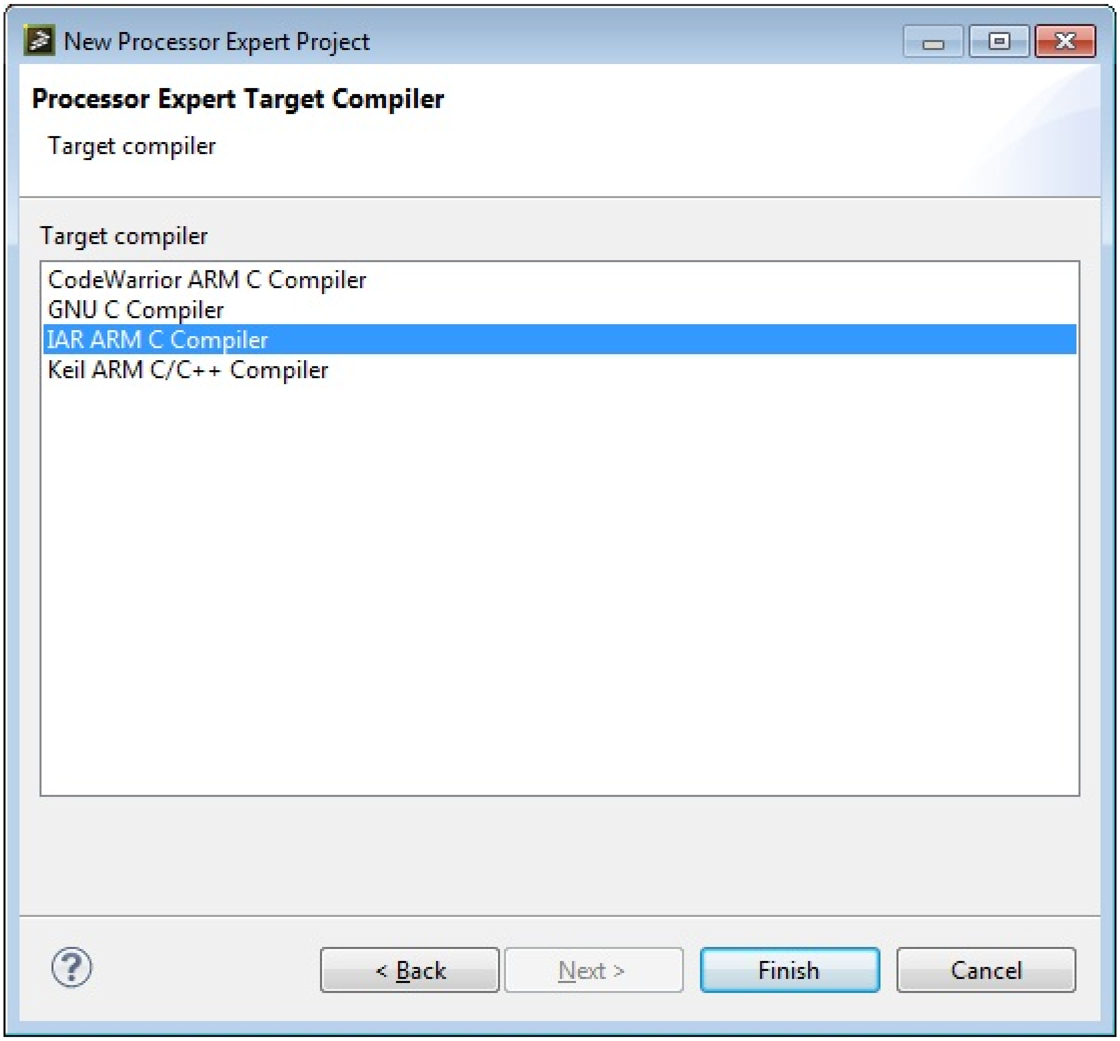
● Use current perspective

□ Initialize all peripherals

Use current perspective and show Processor Expert views.

Figure 4. PEx tool selection dialog

g. Select IAR ARM C Compiler for the Target compiler in the PE tool chain selection dialog, as shown in the following figure.



New Processor Expert Project

Processor Expert Target Compiler

Target compiler

Target compiler

- CodeWarrior ARM C Compiler
- GNU C Compiler
- IAR ARM C Compiler
- Keil ARM C/C++ Compiler

Figure 5. Tool chain selection dialog

h. Click Finish.

2. Configure PEx Project to the same clock configurations the application will use. The desired clock configuration for this application is: core frequency of 96 MHz, bus frequency of 48 MHz, and flash clock frequency of 24 MHz using a 50 MHz external clock.

   a. Select the Expert visibility tab in PEx.

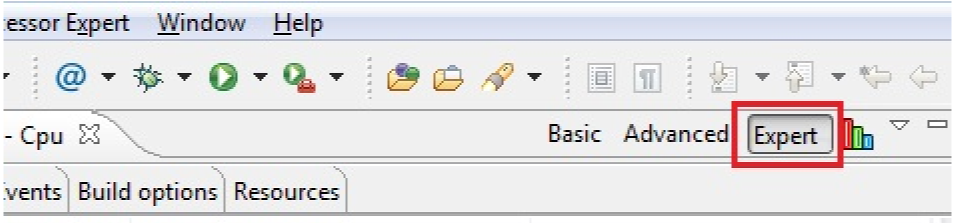
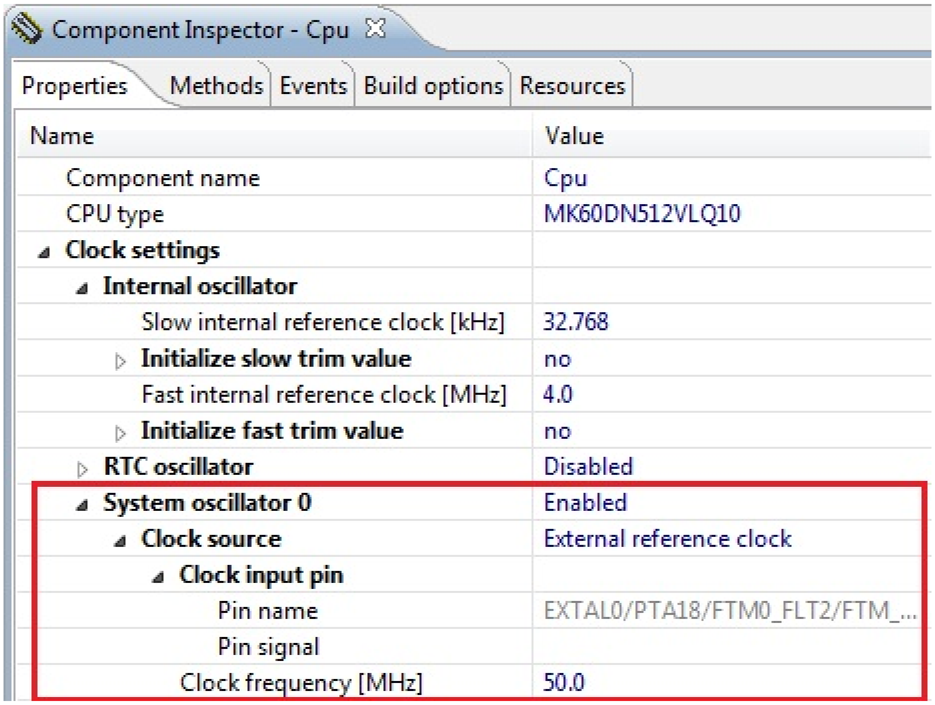


Figure 6. Setting Expert visibility in PEx

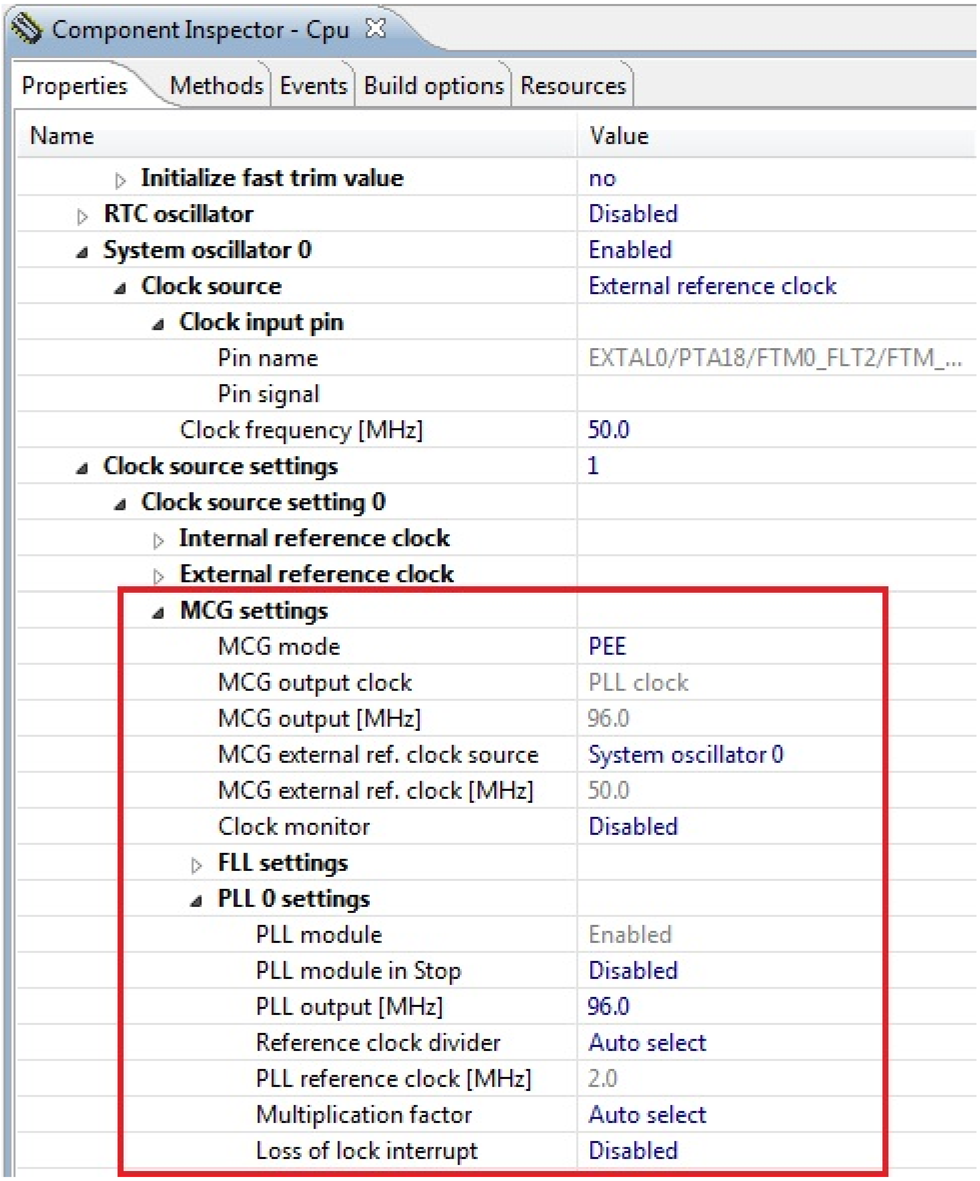
   b. In the Component Inspector - Cpu window, select the Properties tab and enable the system oscillator and its settings.



Name	Value
Component name	Cpu
CPU type	MK60DN512VLQ10
◢ **Clock settings**	
◢ **Internal oscillator**	
Slow internal reference clock [kHz]	32.768
▷ **Initialize slow trim value**	no
Fast internal reference clock [MHz]	4.0
▷ **Initialize fast trim value**	no
▷ **RTC oscillator**	Disabled
◢ **System oscillator 0**	Enabled
◢ **Clock source**	External reference clock
◢ **Clock input pin**	
Pin name	EXTAL0/PTA18/FTM0_FLT2/FTM_...
Pin signal	
Clock frequency [MHz]	50.0

Figure 7. CPU component oscillator configuration section

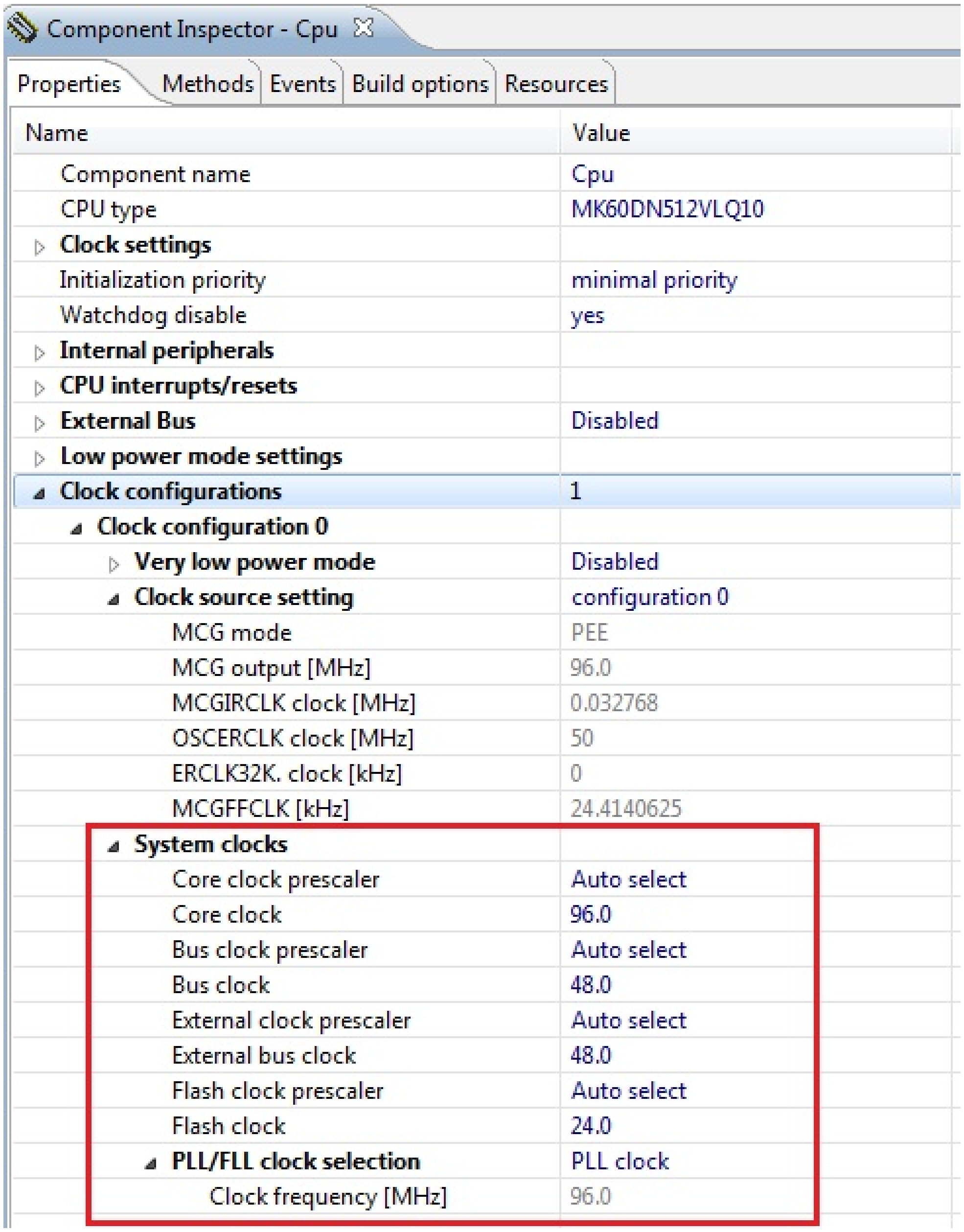
c. For MCG settings, select PEE mode as the MCG mode, and configure the PLL settings as shown in this figure.



Name	Value
Initialize fast trim value	no
RTC oscillator	Disabled
System oscillator 0	Enabled
Clock source	External reference clock
Clock input pin	
Pin name	EXTAL0/PTA18/FTM0_FLT2/FTM_...
Pin signal	
Clock frequency [MHz]	50.0
Clock source settings	1
Clock source setting 0	
Internal reference clock	
External reference clock	
MCG settings	
MCG mode	PEE
MCG output clock	PLL clock
MCG output [MHz]	96.0
MCG external ref. clock source	System oscillator 0
MCG external ref. clock [MHz]	50.0
Clock monitor	Disabled
FLL settings	
PLL 0 settings	
PLL module	Enabled
PLL module in Stop	Disabled
PLL output [MHz]	96.0
Reference clock divider	Auto select
PLL reference clock [MHz]	2.0
Multiplication factor	Auto select
Loss of lock interrupt	Disabled

Figure 8. MCG settings in CPU component oscillator configuration section

d. Configure the System clocks in the clock configuration as shown in this figure.



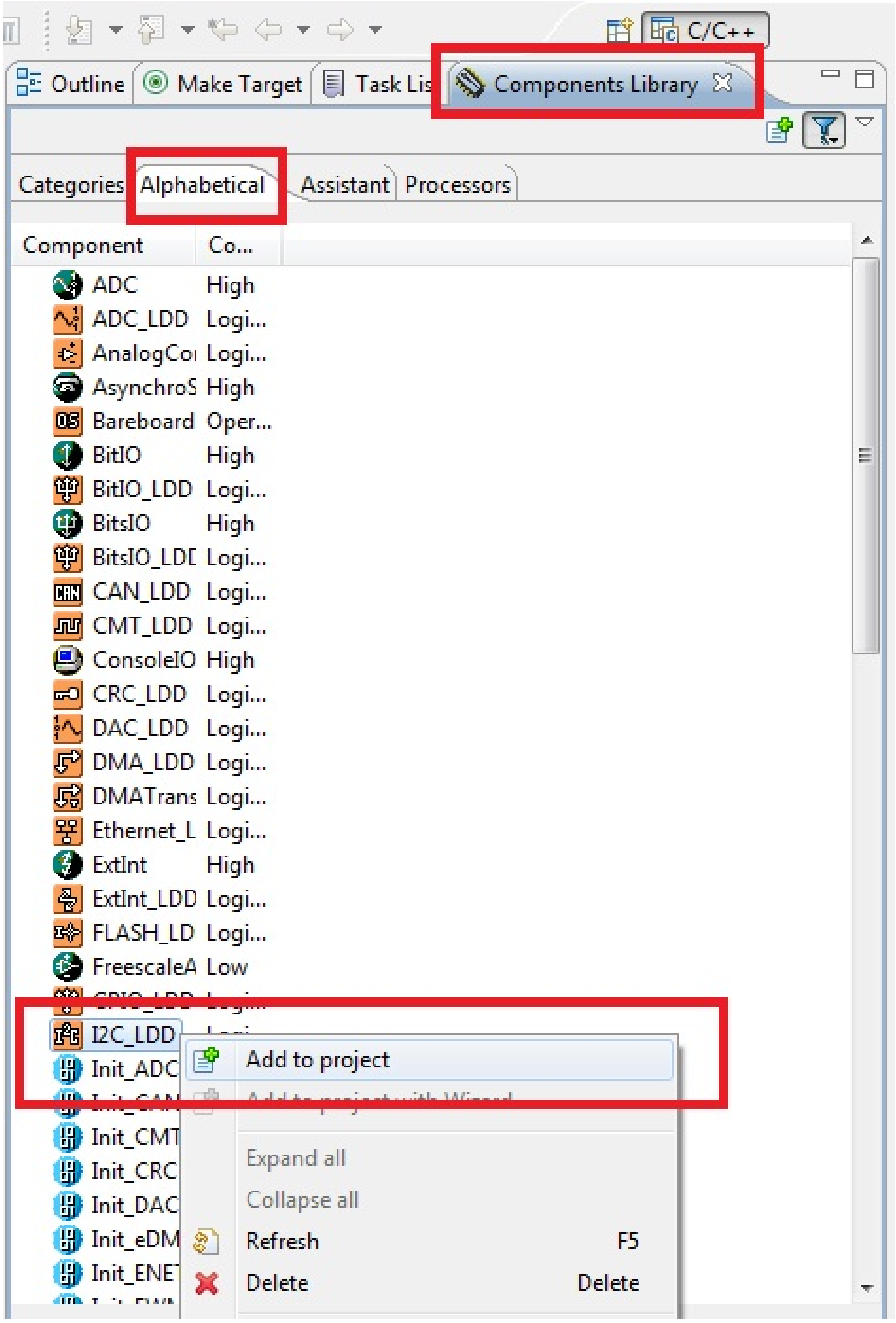
Component Inspector - Cpu

Properties | Methods | Events | Build options | Resources

Name	Value
Component name	Cpu
CPU type	MK60DN512VLQ10
▷ **Clock settings**	
Initialization priority	minimal priority
Watchdog disable	yes
▷ **Internal peripherals**	
▷ **CPU interrupts/resets**	
▷ **External Bus**	Disabled
▷ **Low power mode settings**	
◢ **Clock configurations**	1
◢ **Clock configuration 0**	
▷ **Very low power mode**	Disabled
◢ **Clock source setting**	configuration 0
MCG mode	PEE
MCG output [MHz]	96.0
MCGIRCLK clock [MHz]	0.032768
OSCERCLK clock [MHz]	50
ERCLK32K. clock [kHz]	0
MCGFFCLK [kHz]	24.4140625
◢ **System clocks**	
Core clock prescaler	Auto select
Core clock	96.0
Bus clock prescaler	Auto select
Bus clock	48.0
External clock prescaler	Auto select
External bus clock	48.0
Flash clock prescaler	Auto select
Flash clock	24.0
◢ **PLL/FLL clock selection**	PLL clock
Clock frequency [MHz]	96.0

Figure 9. CPU component system clocks configuration

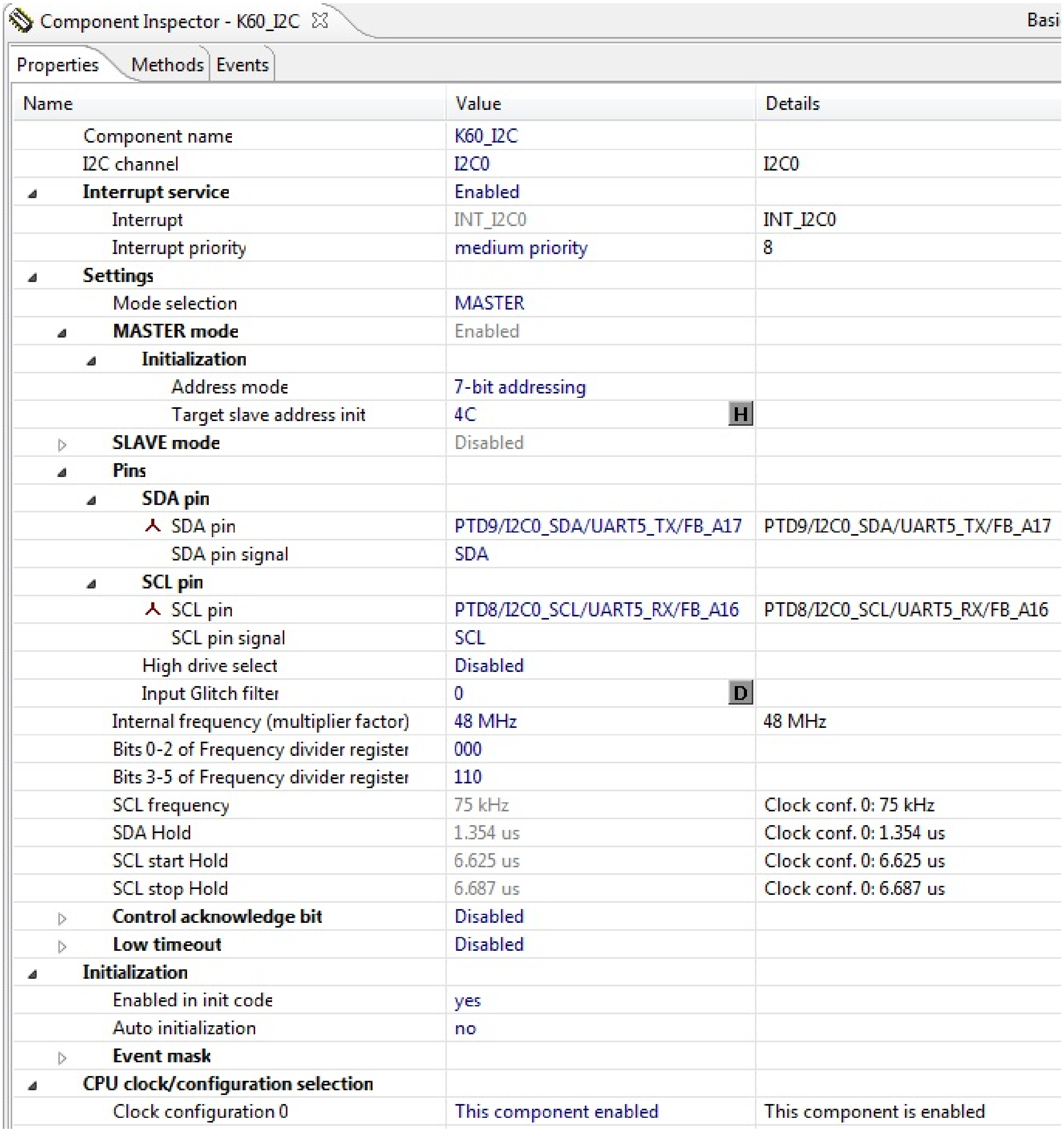
3. Add the I2C_LDD component to the project.
   a. Right-click the I2C_LDD component and select Add to Project.



Component	Co...
ADC	High
ADC_LDD	Logi...
AnalogCo	Logi...
AsynchroS	High
Bareboard	Oper...
BitIO	High
BitIO_LDD	Logi...
BitsIO	High
BitsIO_LDI	Logi...
CAN_LDD	Logi...
CMT_LDD	Logi...
ConsoleIO	High
CRC_LDD	Logi...
DAC_LDD	Logi...
DMA_LDD	Logi...
DMATrans	Logi...
Ethernet_L	Logi...
ExtInt	High
ExtInt_LDD	Logi...
FLASH_LD	Logi...
FreescaleA	Low
GPIO_LDD	Logi...
I2C_LDD	Logi...
Init_ADC	
Init_CAN	
Init_CMT	
Init_CRC	
Init_DAC	
Init_eDM	
Init_ENE	

Figure 10. PEx Component insertion

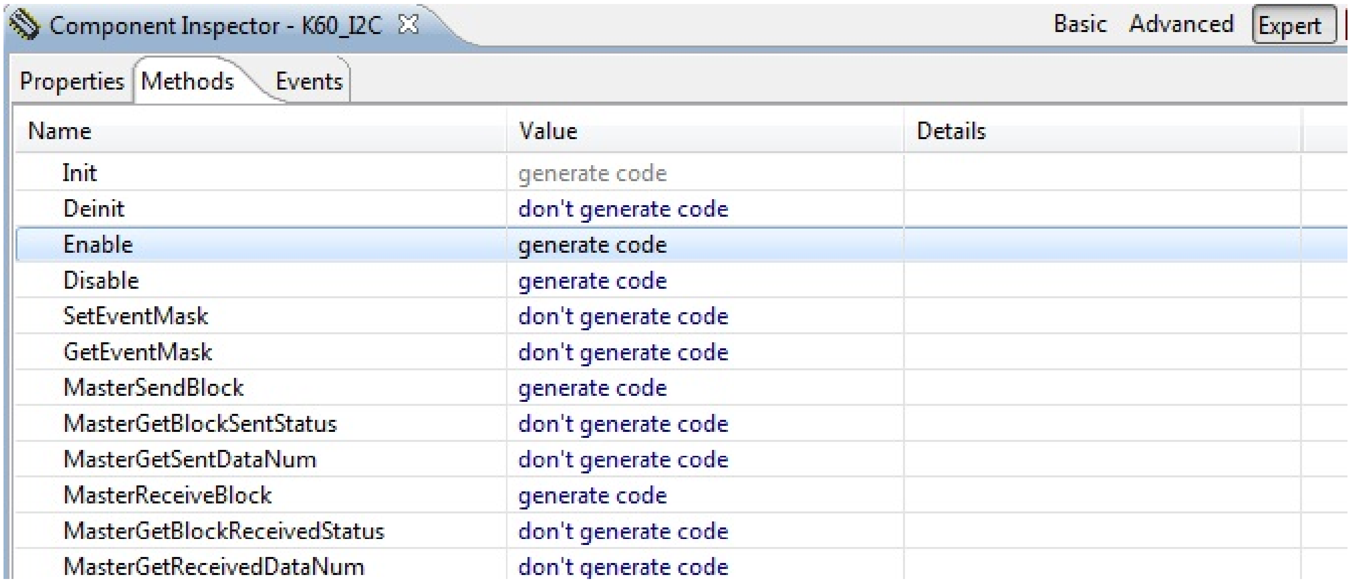
4. Configure the I2C component.
   a. For this application note, the I2C component should be configured as shown in the following figure.



Name	Value	Details
Component name	K60_I2C	
I2C channel	I2C0	I2C0
Interrupt service	Enabled	
Interrupt	INT_I2C0	INT_I2C0
Interrupt priority	medium priority	8
Settings		
Mode selection	MASTER	
MASTER mode	Enabled	
Initialization		
Address mode	7-bit addressing	
Target slave address init	4C H	
SLAVE mode	Disabled	
Pins		
SDA pin		
SDA pin	PTD9/I2C0_SDA/UART5_TX/FB_A17	PTD9/I2C0_SDA/UART5_TX/FB_A17
SDA pin signal	SDA	
SCL pin		
SCL pin	PTD8/I2C0_SCL/UART5_RX/FB_A16	PTD8/I2C0_SCL/UART5_RX/FB_A16
SCL pin signal	SCL	
High drive select	Disabled	
Input Glitch filter	0 D	
Internal frequency (multiplier factor)	48 MHz	48 MHz
Bits 0-2 of Frequency divider register	000	
Bits 3-5 of Frequency divider register	110	
SCL frequency	75 kHz	Clock conf. 0: 75 kHz
SDA Hold	1.354 us	Clock conf. 0: 1.354 us
SCL start Hold	6.625 us	Clock conf. 0: 6.625 us
SCL stop Hold	6.687 us	Clock conf. 0: 6.687 us
Control acknowledge bit	Disabled	
Low timeout	Disabled	
Initialization		
Enabled in init code	yes	
Auto initialization	no	
Event mask		
CPU clock/configuration selection		
Clock configuration 0	This component enabled	This component is enabled

Figure 11. I2C component configuration

b. Also remember to enable the "Enable" and "Disable" methods for this component.



Name	Value	Details
Init	generate code	
Deinit	don't generate code	
Enable	generate code	
Disable	generate code	
SetEventMask	don't generate code	
GetEventMask	don't generate code	
MasterSendBlock	generate code	
MasterGetBlockSentStatus	don't generate code	
MasterGetSentDataNum	don't generate code	
MasterReceiveBlock	generate code	
MasterGetBlockReceivedStatus	don't generate code	
MasterGetReceivedDataNum	don't generate code	

Figure 12. I2C component methods

5. Generate the code.
   a. Now, generate the code by clicking the Generate Code button on the right-side corner of the Components window, (as shown in the following figure), or choose Project > Generate Processor Expert Code.

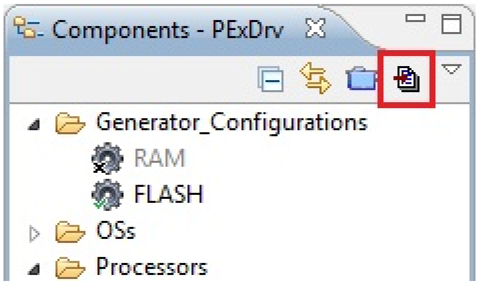


Figure 13. Code generation button

6. Open the PEx_Drv_Int_Training workspace located at ...build\iar\PEx_Drv_Int_Training.
7. Select the PEx_Drv_Int_Training_k60_tower_project and select the Flash_256KB_PFLASH target.

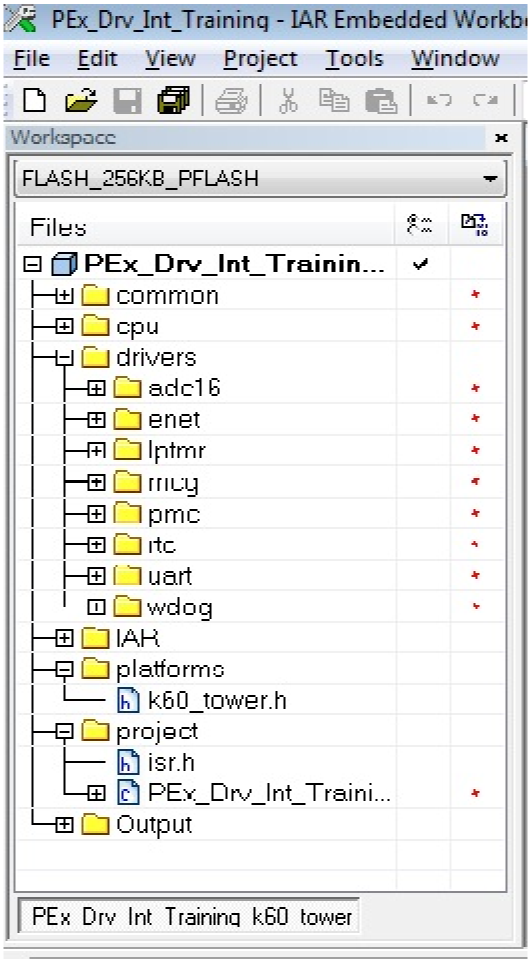


Figure 14. Project selection IAR IDE

8. Add the generated code to the Project.
    a. Right-click the drivers folder and choose Add > Add Group.

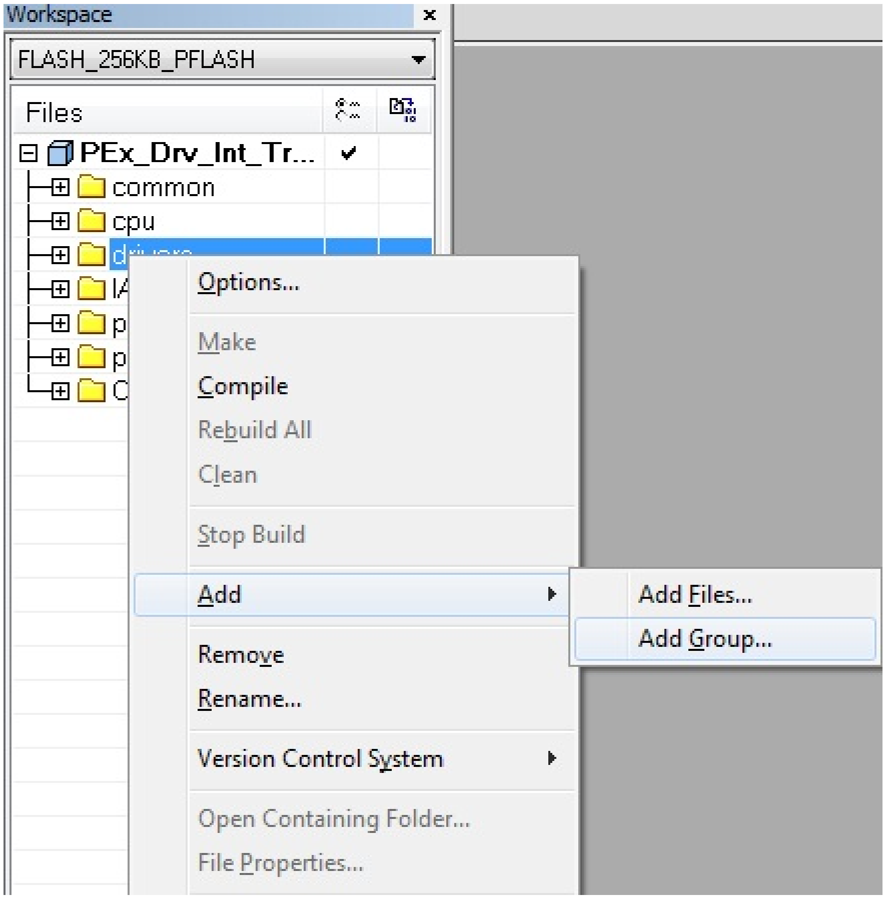
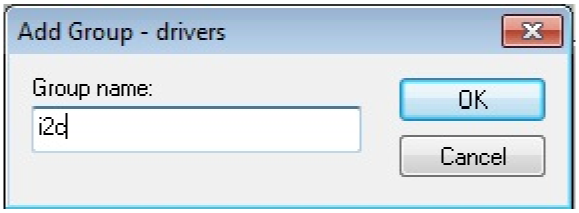


Figure 15. Add group selection in IAR

b. Name the group appropriately.



Group name: i2c

Figure 16. Add group dialog box in IAR

c. Right-click the folder you just created and choose Add > Add Files.

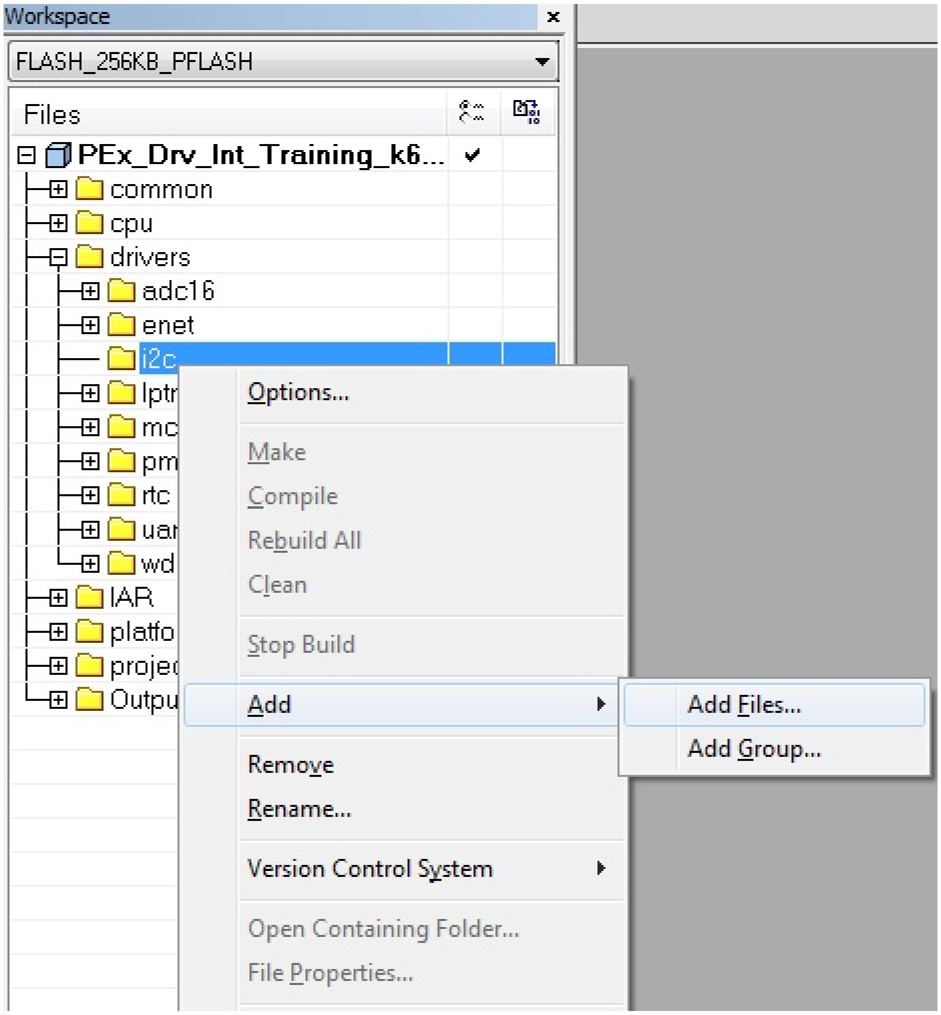
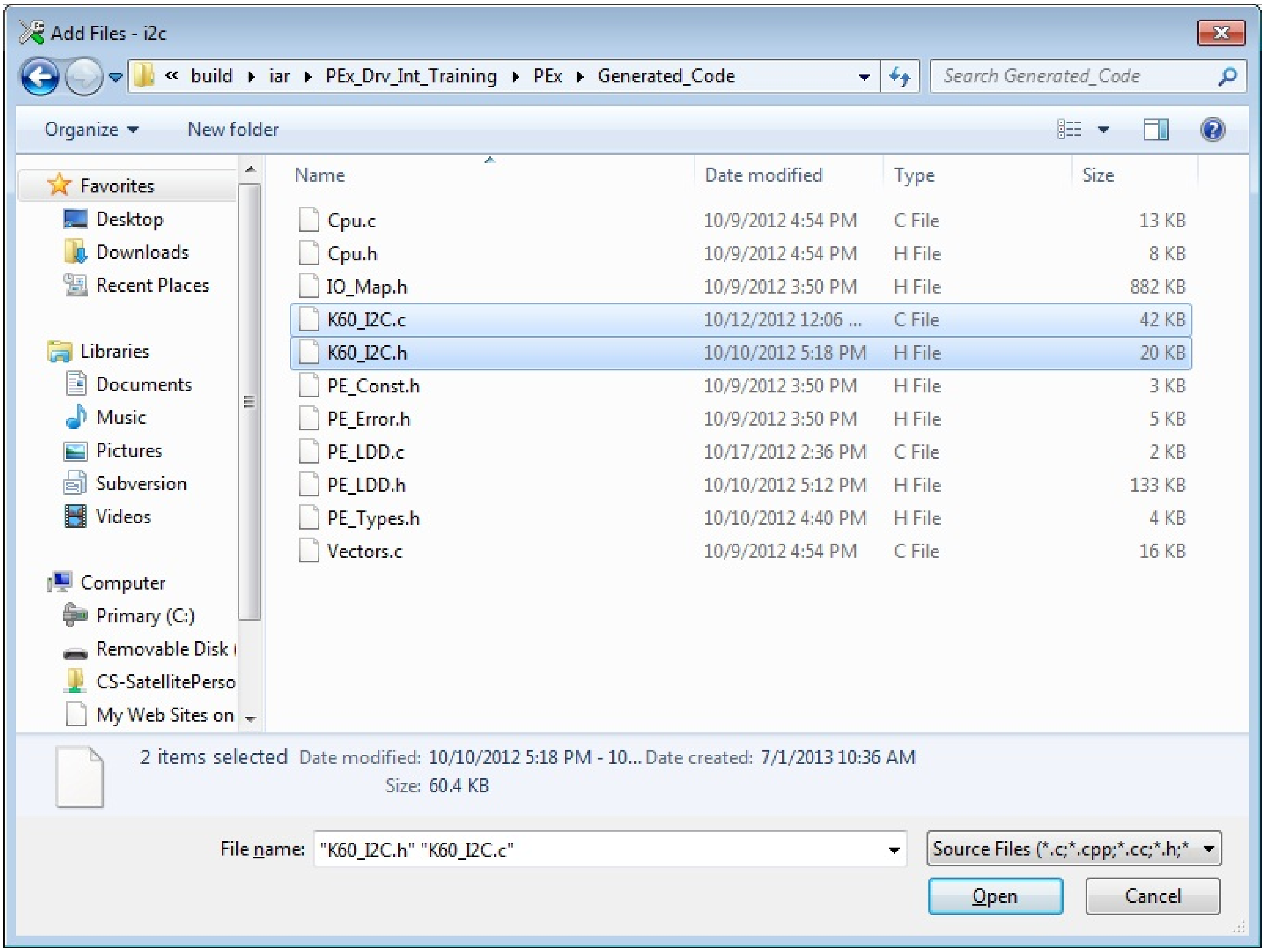


Figure 17. Adding files in IAR

d. Point IAR to the driver files that were generated (K60_I2C.c and K60_I2C.h) located at ...build\iar \PEx_Drv_Int_Training\PE\Generated_Code.



Name	Date modified	Type	Size
Cpu.c	10/9/2012 4:54 PM	C File	13 KB
Cpu.h	10/9/2012 4:54 PM	H File	8 KB
IO_Map.h	10/9/2012 3:50 PM	H File	882 KB
K60_I2C.c	10/12/2012 12:06 ...	C File	42 KB
K60_I2C.h	10/10/2012 5:18 PM	H File	20 KB
PE_Const.h	10/9/2012 3:50 PM	H File	3 KB
PE_Error.h	10/9/2012 3:50 PM	H File	5 KB
PE_LDD.c	10/17/2012 2:36 PM	C File	2 KB
PE_LDD.h	10/10/2012 5:12 PM	H File	133 KB
PE_Types.h	10/10/2012 4:40 PM	H File	4 KB
Vectors.c	10/9/2012 4:54 PM	C File	16 KB

2 items selected Date modified: 10/10/2012 5:18 PM - 10... Date created: 7/1/2013 10:36 AM
Size: 60.4 KB

File name: "K60_I2C.h" "K60_I2C.c"

Source Files (*.c;*.cpp;*.cc;*.h;*

Figure 18. Add Files dialog in IAR

e. Right-click the project folder and choose Add > Add Files.

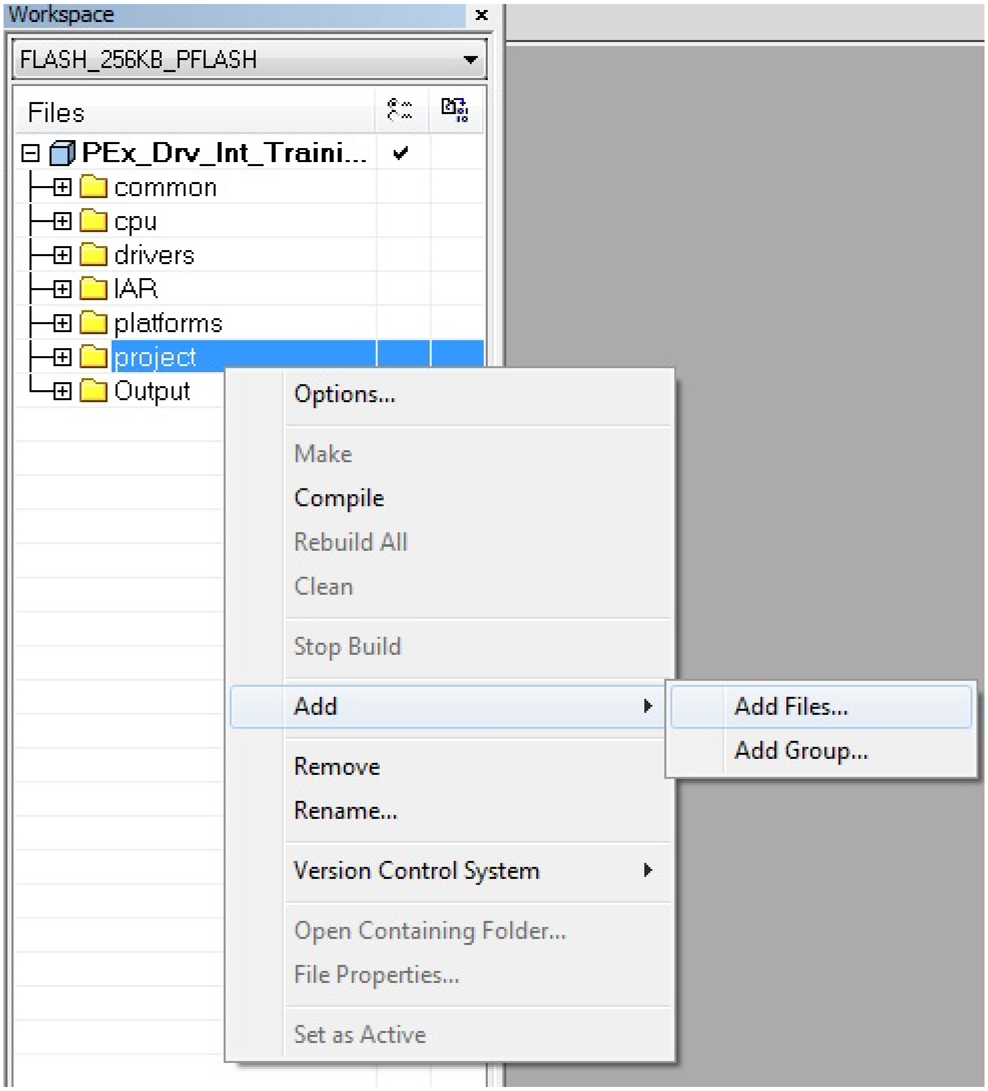
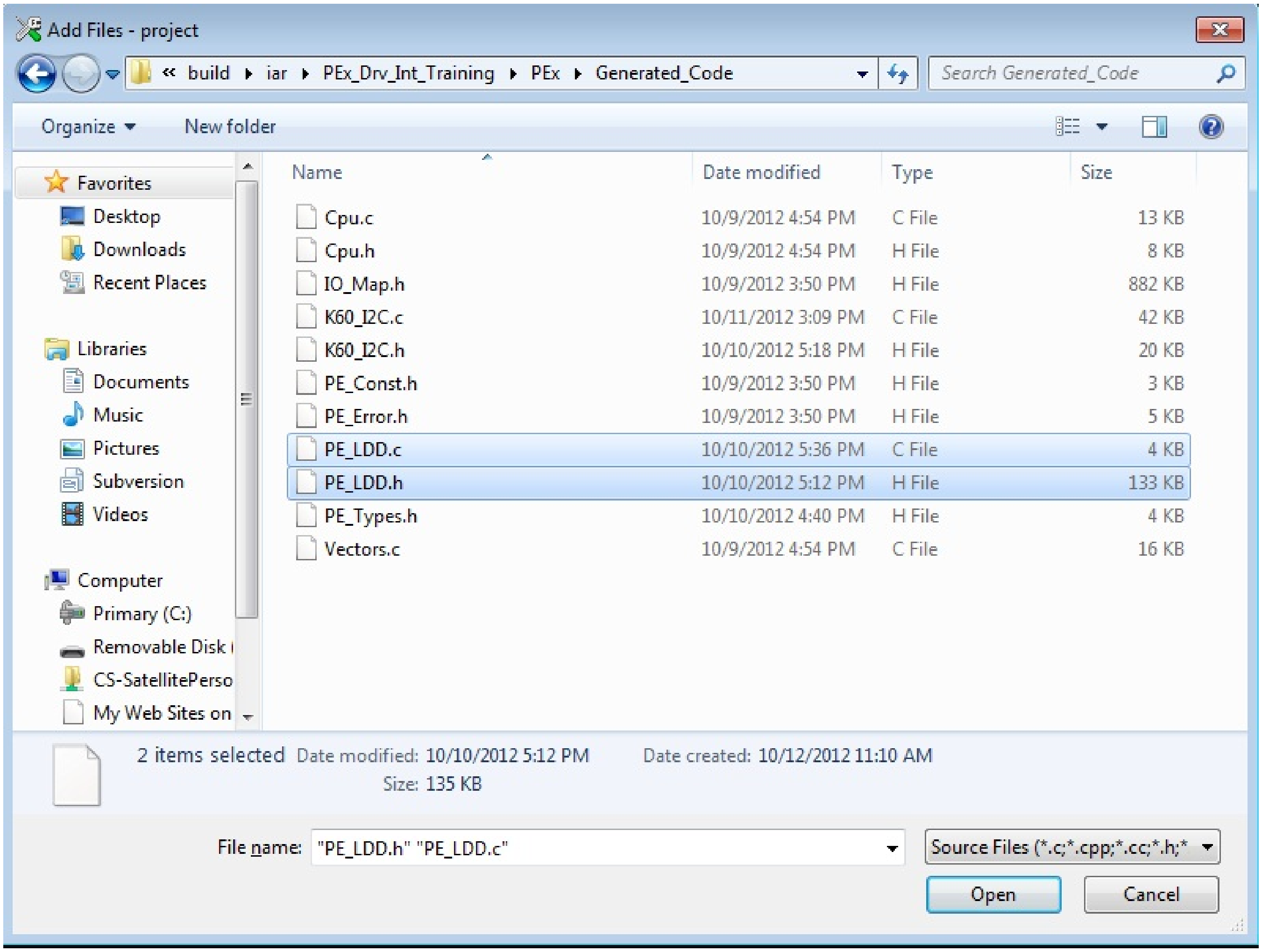


Figure 19. Adding files to the Project group in IAR

f. Point IAR to the generated code folder and select PE_LDD.c and PE_LDD.h.



Name	Date modified	Type	Size
Cpu.c	10/9/2012 4:54 PM	C File	13 KB
Cpu.h	10/9/2012 4:54 PM	H File	8 KB
IO_Map.h	10/9/2012 3:50 PM	H File	882 KB
K60_I2C.c	10/11/2012 3:09 PM	C File	42 KB
K60_I2C.h	10/10/2012 5:18 PM	H File	20 KB
PE_Const.h	10/9/2012 3:50 PM	H File	3 KB
PE_Error.h	10/9/2012 3:50 PM	H File	5 KB
PE_LDD.c	10/10/2012 5:36 PM	C File	4 KB
PE_LDD.h	10/10/2012 5:12 PM	H File	133 KB
PE_Types.h	10/10/2012 4:40 PM	H File	4 KB
Vectors.c	10/9/2012 4:54 PM	C File	16 KB

2 items selected Date modified: 10/10/2012 5:12 PM Date created: 10/12/2012 11:10 AM
Size: 135 KB

File name: "PE_LDD.h" "PE_LDD.c"
Source Files (*.c;*.cpp;*.cc;*.h;*

Figure 20. Add files dialog in IAR - PE_LDD files

   g. Repeat steps "e" and "f" to also add Events.c and Events.h, located at build\iar\PEx_Drv_Int_Training\PE \Sources, to the sources folder.

9. Add the paths to the Preprocessor include paths.

   a. Right-click the project and select Options.

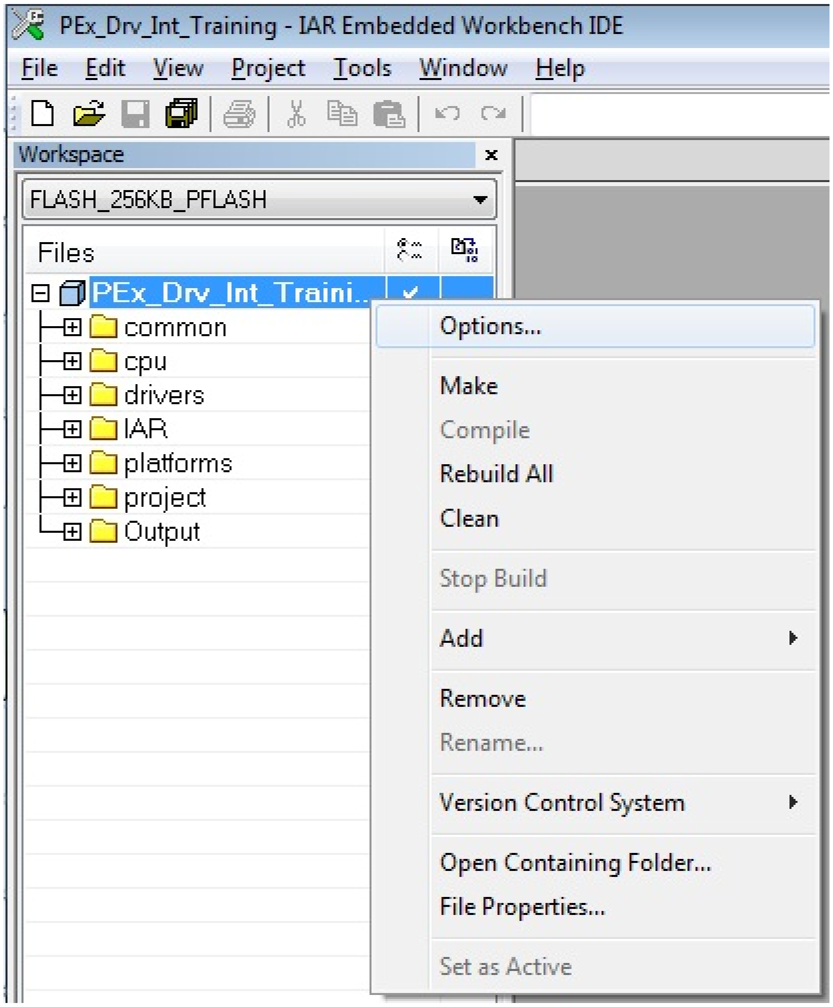
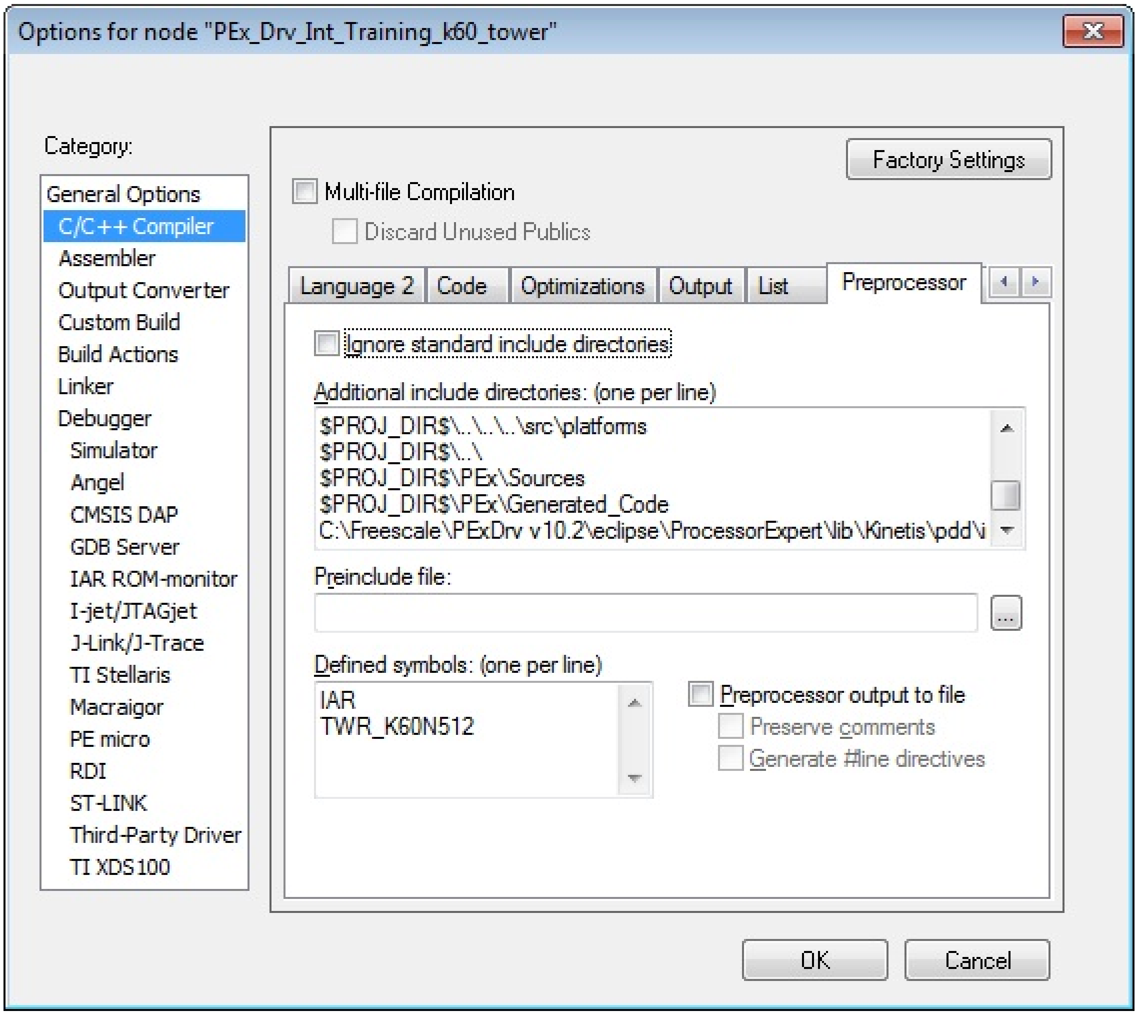


Figure 21. Opening project options in IAR

b. A dialog will open for node "PEx_Drv_Int_Training_k60_tower". In the dialog, select the Preprocessor tab from the C/C++ Compiler Category and add the following paths (also shown in the figure below):
  - $PROJ_DIR$\PE\Generated_Code
  - $PROJ_DIR$\PE\Sources
  - The PDD include folder of the PEx install directory. (If you chose the default location for the PEx install, this location will be C:\Freescale\PExDrv v10.2\eclipse\ProcessorExpert\lib\Kinetis\pdd\inc).



Options for node "PEx_Drv_Int_Training_k60_tower"

Category:

General Options
C/C++ Compiler
Assembler
Output Converter
Custom Build
Build Actions
Linker
Debugger
Simulator
Angel
CMSIS DAP
GDB Server
IAR ROM-monitor
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
RDI
ST-LINK
Third-Party Driver
TI XDS100

Factory Settings

□ Multi-file Compilation

□ Discard Unused Publics

Language 2 | Code | Optimizations | Output | List | Preprocessor

□ Ignore standard include directories

Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\..\src\platforms
$PROJ_DIR$\..\
$PROJ_DIR$\PEx\Sources
$PROJ_DIR$\PEx\Generated_Code
C:\Freescale\PExDrv v10.2\eclipse\ProcessorExpert\lib\Kinetis\pdd\i
```

Preinclude file:

Defined symbols: (one per line)

```
IAR
TWR_K60N512
```

□ Preprocessor output to file

□ Preserve comments

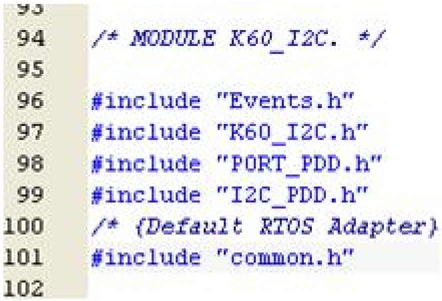
□ Generate #line directives

OK | Cancel

Figure 22. Preprocessor tab in the project options of an IAR project

c. Click OK.

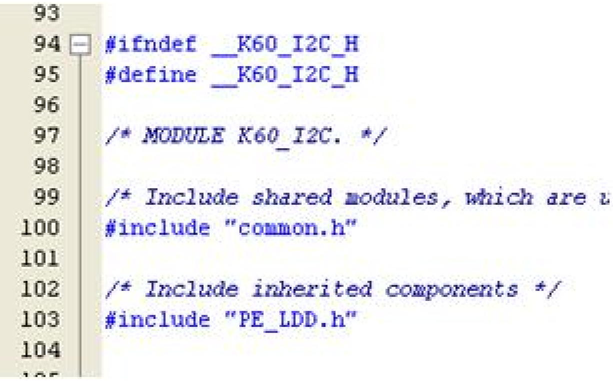
10. Modify the driver file, K60_I2C.c.
    - K60_I2C.c should not include IO_Map.h, but should include common.h. Replace #include "IO_Map.h" with "#include "common.h"" as shown in this figure.



```
/* MODULE K60_I2C. */

#include "Events.h"
#include "K60_I2C.h"
#include "PORT_PDD.h"
#include "I2C_PDD.h"
/* {Default RTOS Adapter} */
#include "common.h"

```

Figure 23. K60_I2C.c includes

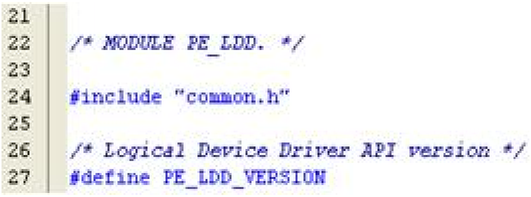
11. Modify the driver header file, K60_I2C.h.
    - K60_I2C.h should only include common.h and PE_LDD.h. Replace all of the file inclusions (#include "<file name here>") as shown in this figure.



```
#ifndef __K60_I2C_H
#define __K60_I2C_H

/* MODULE K60_I2C. */

/* Include shared modules, which are u
#include "common.h"

/* Include inherited components */
#include "PE_LDD.h"

```

Figure 24. K60_I2C.h includes

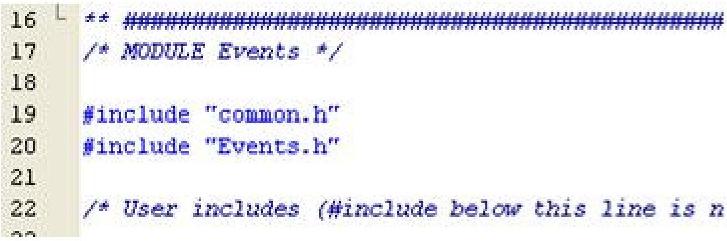
12. Modify PE_LDD.c.
    - Remove the "#include "CPU.h"" statement.
    - Remove all functions except for the declaration LDD_TDeviceData *PE_LDD_DeviceDataList [1].
13. Modify PE_LDD.h.
    - Replace all of the file inclusions (#include "<file name here>") with "#include "common.h"" as shown.



```

/* MODULE PE_LDD. */

#include "common.h"

/* Logical Device Driver API version */
#define PE_LDD_VERSION
```

Figure 25. PE_LDD.h includes

14. Modify Events.c.
    - Events.c should include common.h but not CPU.h. Replace #include "CPU.h" as shown in the following figure.

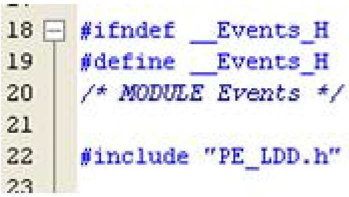


```
** ###################################################################
/* MODULE Events */

#include "common.h"
#include "Events.h"

/* User includes (#include below this line is n
```

Figure 26. Events.c includes

- Add the following lines immediately after the header file inclusions (where the user includes are allowed):
    - extern volatile bool DataTransmittedFlg;
    - extern volatile bool DataReceivedFlg;
- Add to the K60_I2C_OnMasterBlockSent function "DataTransmittedFlg = TRUE;" this semaphore is used by the provided example application to signal that data has been transmitted to the accelerometer.
- Add to the K60_I2C_OnMasterBlockReceived function "DataReceivedFlg = TRUE;" this semaphore is used by the provided example application to signal that data has been received from the accelerometer.

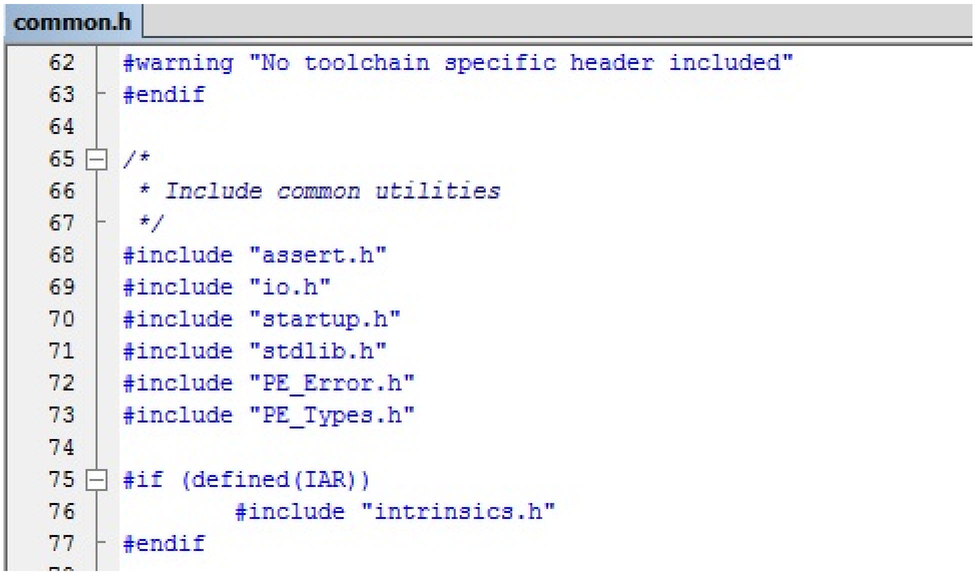
15. Modify Events.h.
    - Events.h should only include PE_LDD.h. Replace all of the file inclusions (#include "<file name here>") as shown in this figure.



```
#ifndef __Events_H
#define __Events_H
/* MODULE Events */

#include "PE_LDD.h"
```

Figure 27. Events.h includes

16. Modify common.h.
    - common.h should include PE_Error.h and PE_Types.h. PE_Error.h defines error codes used by LDD files that have been generated. PE_Types.h defines the constants necessary for the LDD drivers to operate. Add these includes at line 72 as shown.

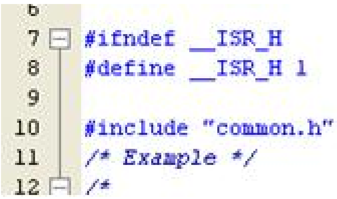


common.h

```
62  #warning "No toolchain specific header included"
63  #endif
64
65  /*
66   * Include common utilities
67   */
68  #include "assert.h"
69  #include "io.h"
70  #include "startup.h"
71  #include "stdlib.h"
72  #include "PE_Error.h"
73  #include "PE_Types.h"
74
75  #if (defined(IAR))
76          #include "intrinsics.h"
77  #endif
```

Figure 28. common.h includes

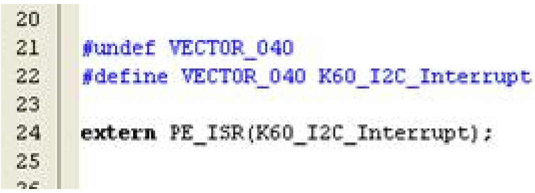
17. Modify isr.h to install the Processor Expert generated interrupt handler.
    - isr.h should include common.h



```
#ifndef __ISR_H
#define __ISR_H 1

#include "common.h"
/* Example */
/*
```

Figure 29. isr.h includes

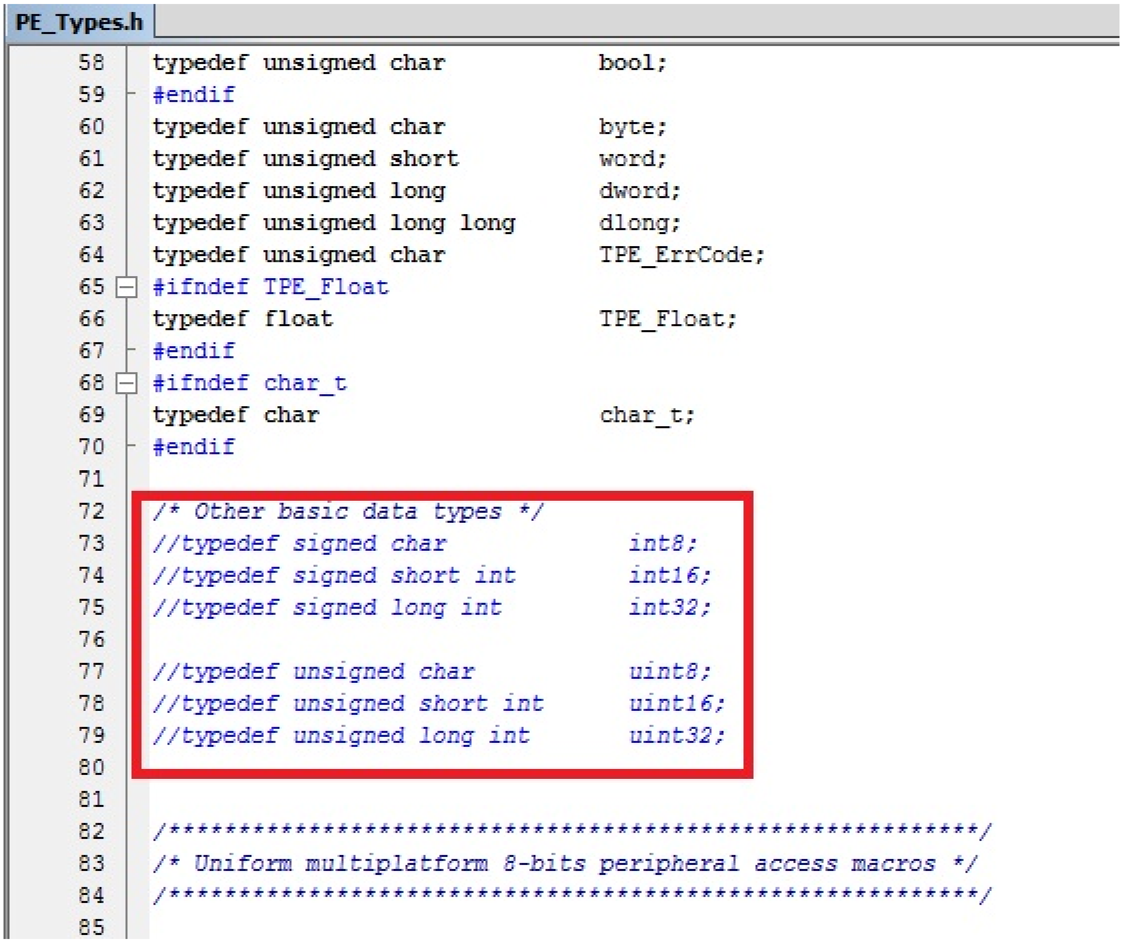
- To install the interrupt vector, redefine the appropriate vector as shown (remember to declare the ISR function).



```

#undef VECTOR_040
#define VECTOR_040 K60_I2C_Interrupt

extern PE_ISR(K60_I2C_Interrupt);

```

Figure 30. Example of installing an interrupt service routine

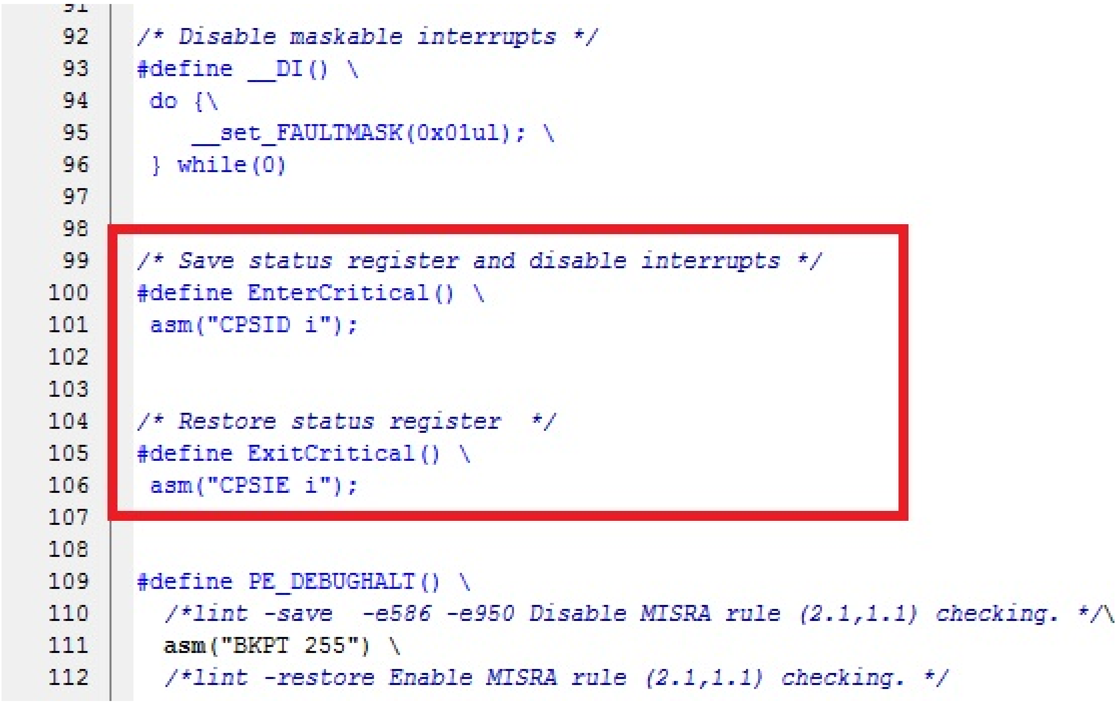
18. a. Remove the definition of "Other basic data types"



```
typedef unsigned char           bool;
#endif
typedef unsigned char           byte;
typedef unsigned short          word;
typedef unsigned long           dword;
typedef unsigned long long      dlong;
typedef unsigned char           TPE_ErrCode;
#ifndef TPE_Float
typedef float                   TPE_Float;
#endif
#ifndef char_t
typedef char                    char_t;
#endif

/* Other basic data types */
//typedef signed char             int8;
//typedef signed short int        int16;
//typedef signed long int         int32;

//typedef unsigned char           uint8;
//typedef unsigned short int      uint16;
//typedef unsigned long int       uint32;


/**********************************************************/
/* Uniform multiplatform 8-bits peripheral access macros */
/**********************************************************/

```

Figure 31. Removal of "Other basic data types" from PE_Types.h

b. Modify the "EnterCritical" and "ExitCritical" functions as shown in the following figure.



```
/* Disable maskable interrupts */
#define __DI() \
 do {\
    __set_FAULTMASK(0x01ul); \
 } while(0)


/* Save status register and disable interrupts */
#define EnterCritical() \
 asm("CPSID i");


/* Restore status register  */
#define ExitCritical() \
 asm("CPSIE i");


#define PE_DEBUGHALT() \
  /*lint -save  -e586 -e950 Disable MISRA rule (2.1,1.1) checking. */\
  asm("BKPT 255") \
  /*lint -restore Enable MISRA rule (2.1,1.1) checking. */
```

Figure 32. Modification of EnterCritical and ExitCritical functions in PE_Types.h

6 Conclusion

In this application note, a general procedure is outlined for integrating a single Processor Expert Driver into an existing non-Processor Expert project. A specific example of such an integration is also discussed using the Freescale sample code. The general procedure is as follows.

1. Create a Processor Expert project.
2. Configure the PEx project as your project will be setup (this is a good general practice but will be required if your driver is a time-dependent module, such as a timer or communications module).
3. Generate the code.
4. Add the generated code to the project (typically you only need to add Events.c, Events.h, PE_LDD.c, PE_LDD.h, <Component Name>.c, and <Component Name>.h) and the PDD files.
5. Modify the driver files that have been added (no files will need Cpu.h, PE_Types.h, IO_Map.h, or PE_Const.h. In addition, all of the code should be removed from PE_LDD.c except for the LDD_DeviceData defintion. Also, several standard type definitions, EnterCritical, ExitCritical, and PE_ISR definitions must be added to a common included file, such as the part specific header file or arm_cm4.h.).

It must be noted that each application may use drivers in a different manner and different drivers may require different definitions and other code located in other Processor Expert files. Thus, each driver integration may be slightly different, but the general procedure will remain the same.

7 References

- KINETIS512_SC: Kinetis family example projects, available at freescale.com
- KINETIS512_SC_V2: Kinetis 100MHz Rev 2 Example Projects, available at freescale.com
- Kinetis resources are available at www.freescale.com/Kinetis
- Processor Expert resources are available at www.freescale.com/ProcessExpert or at www.freescale.com/infocenter.

8 Revision history

Revision number	Date	Substantial changes
0	08/2013	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, Kinetis, and Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.