Hands-on Workshop: PEx Driver Suite and MQX™ Lite RTOS

AMF-ENT-T1017

Greg Hemstreet
Manager, Processor Expert Team

August 2013

Agenda

- **Embedded Code Development & Processor Expert**
  - **Introduction slides – don't worry, this is not just a lecture**
  - **LAB 1: Create New Project Using Processor Expert & CodeWarrior**
    - Create a new project with the New MCU Project wizard
    - Add and configure components.
    - Generate code.
    - Add application code.
    - Build and run project
  - **LAB 2: Use the Driver Suite & IAR to create the same application**
- **High Level and Logical Device Driver Components Revisited**
  - **LAB 3: Exploring the difference between Internal Peripheral Drivers and Logical Device Drivers…**

MQX-lite Agenda

- **MQX Lite Overview**
  - **Again with the slideware ☺**

- **LAB 4: Create New Project for MKL25Z Using Processor Expert and MQX-Lite**
  - Create a new project with the New MQX-Lite Project wizard
  - Configure the RTOS timer
  - Add and configure tasks
  - Add and configure components
  - Generate code
  - Add application code
  - Build and run the project

- **LAB 5: Using the Hardware Perspective & other cool stuff**
  - Hardware Engineers love the Hardware Perspective – eye candy to the hardware guys
  - Export a board configuration file
  - Import an empty project for a Freedom board
  - Configure the entire platform from the configuration file

Processor Expert Software

Either…

Microcontroller Driver Suite v10.0
QorIQ Configuration Suite v2.2
Component Development Environment v1.1
… and more

…or

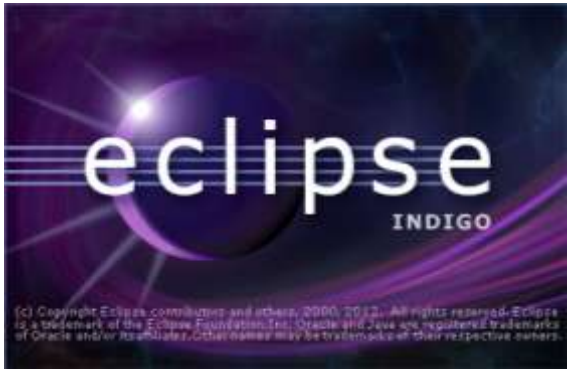
CodeWarrior for Microcontrollers
CodeWarrior for Power Architectures
Other Eclipse based IDEs

Different installs for different folks…

Either…

- Full Installation – Easiest, Fastest Starter
  - Installer creates new Eclipse installation that includes Processor Expert pre-configured.
  - Also adds other capabilities like doxygen to Eclipse install.

…Or



- Eclipse Updater Archive – For existing tools
  - Eclipse “Help>Install New Software” used to add Processor Expert feature to an existing Eclipse toolset (like CodeRed, GNU tools, etc.)
  - Also used to add QorIQ Configuration Suite to an existing CodeWarrior for PA.

Embedded Code Development

In the beginning there were…

You have to read them,

K60 Sub-Family Reference Manual

Supports: MK60N256VLL100, MK60X256VLL100, MK60N512VLL100

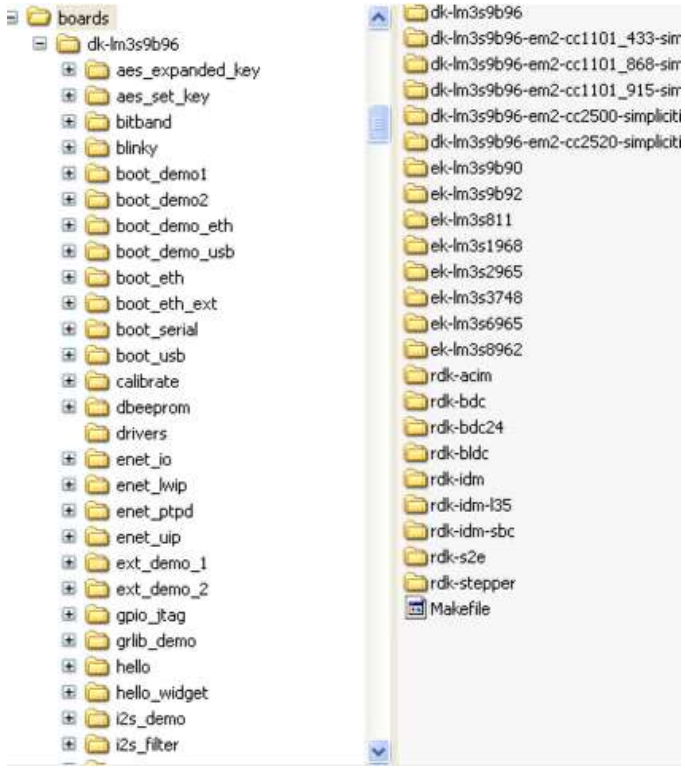
Document Number: K60P100M100SF2RM
Rev. 4, 1 Mar 2011

comprehend them,

and then write your code.

Silicon Data Sheets

As silicon complexity grew, manufacturers began providing…



Static Libraries

You still have to read the silicon data sheets.

But, now you have to become familiar with the code in the library,

choose the code to use,

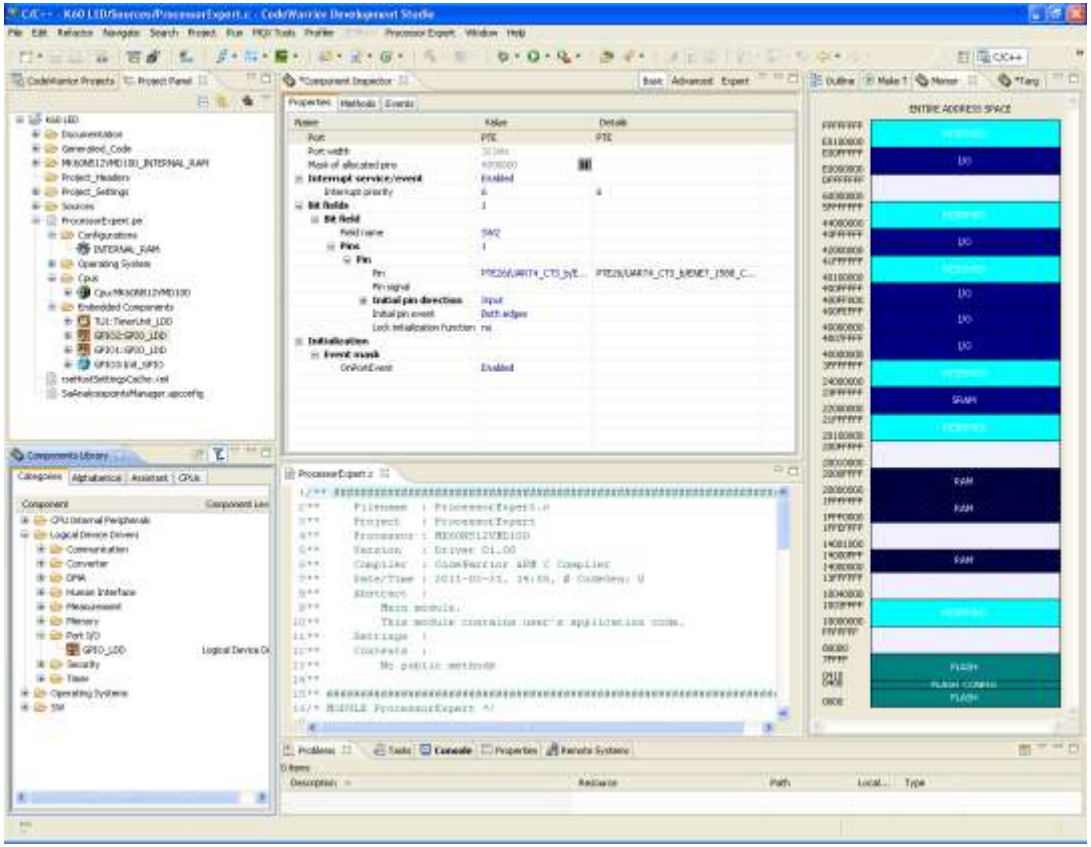
determine the correct calling parameters for the code,

and then write your application code.

freescale™

Embedded Code Development

Freescale provides a better solution…



It is a rapid application development tool for automated code generation.

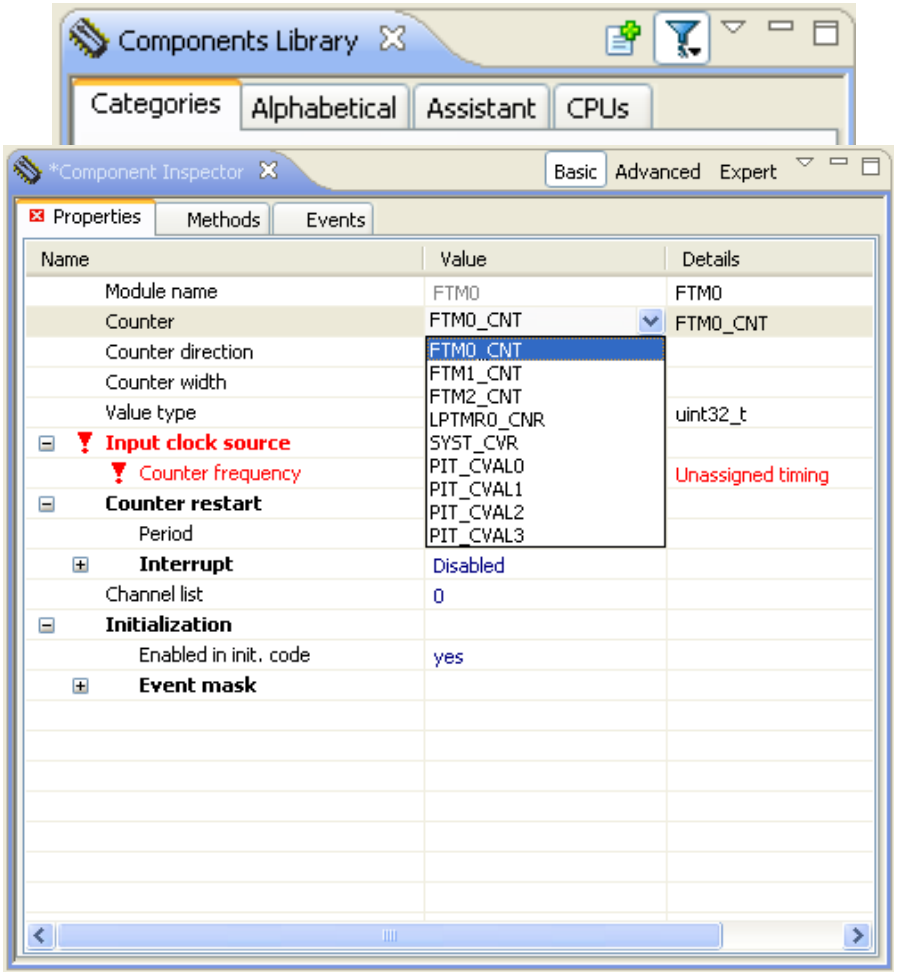
It is an integrated silicon knowledgebase.

It is a software deployment tool.

Processor Expert

Embedded Code Development

Freescale provides a better solution – Processor Expert



Knowledgebase

The GUI interface allows you to explore the silicon

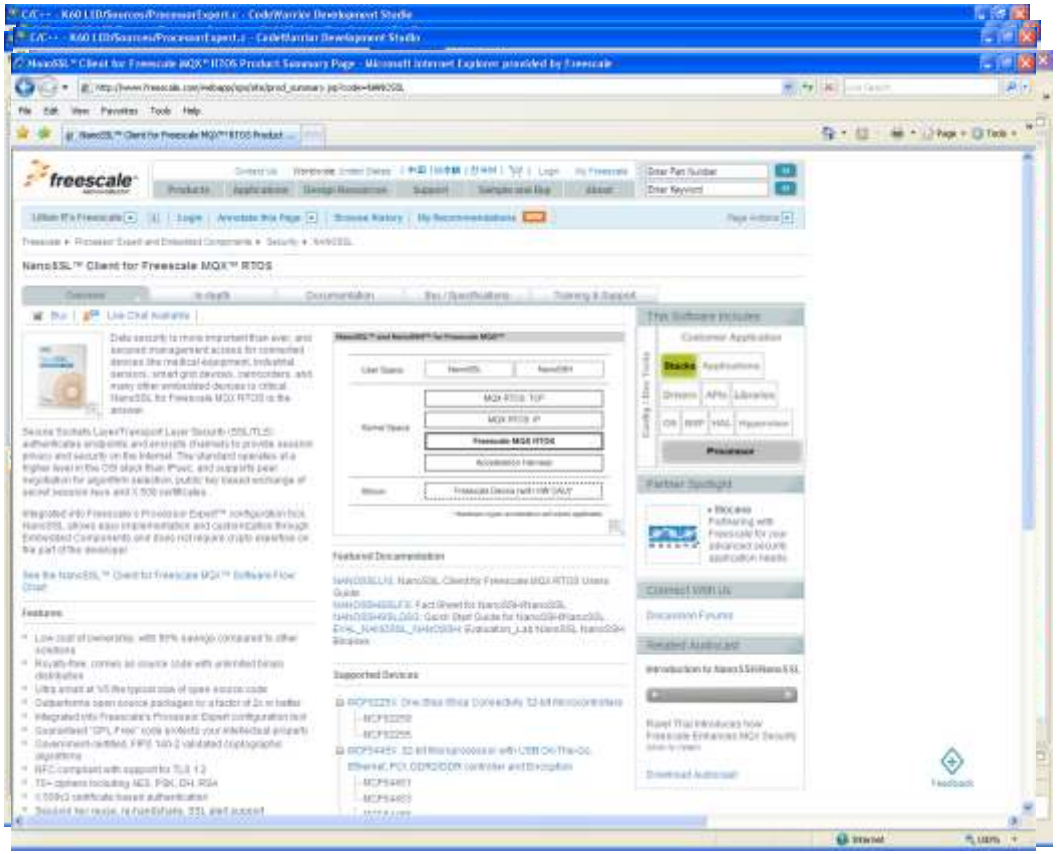
and select functionality based on your application needs.

Only silicon resources that provide the desired functionality are listed.

Resource conflicts / incorrect or incomplete settings are immediately flagged.

Freescale provides a better solution

Processor Expert



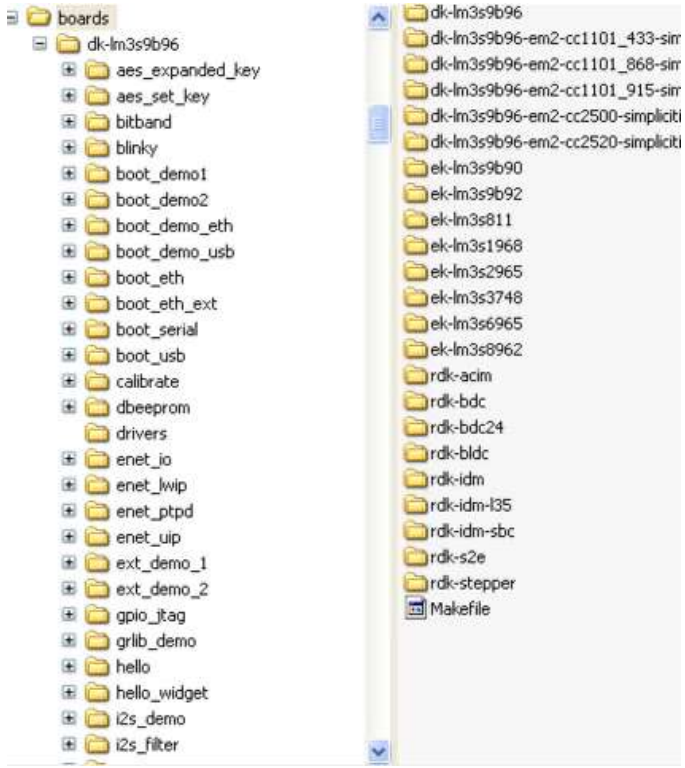
Software Deployment

You can create and distribute pre-set software components (templates) to your software teams.

You can use Component Wizard to encapsulate legacy code into software components for easy maintenance and distribution.

You can purchase third party IP to add to your application.

As your application complexity grows, you may want to add an RTOS



Using Static Libraries

You have to be familiar with the code in the library,

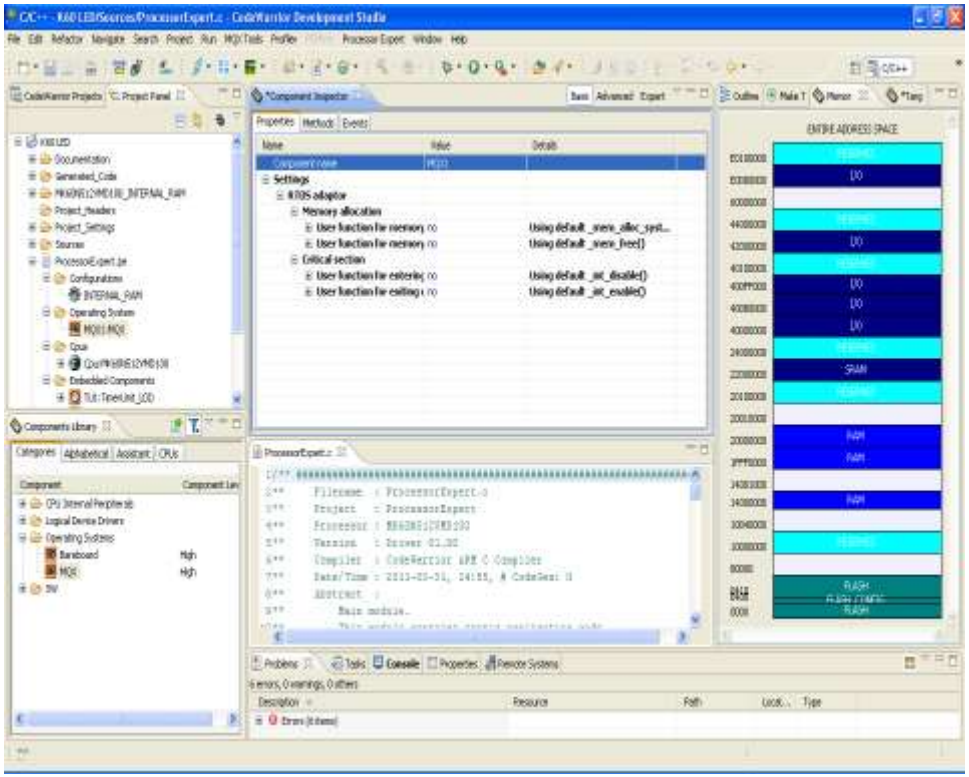
choose the code to use,

determine the correct calling parameters for the code,

determine how to interface the code with the RTOS

and then write your application code.

As your application complexity grows, you may want to add an RTOS



Using Processor Expert

Add the RTOS adapter component to your project.

It tells Processor Expert how the RTOS handles memory allocation and critical code.

Processor Expert then generates code that works with the RTOS – allowing you to extend the peripheral support of the RTOS.

Processor Expert Software

What the heck is Processor Expert?

Processor Expert Software is a development system to create, configure, optimize, migrate, and deliver software components that generate source code for Freescale silicon.

- Features
  - Extensive and comprehensive knowledgebase for all supported silicon encapsulating all pins, registers, etc.
  - Silicon resource conflicts flagged at design time, allowing early correction
  - Simple creation of optimized peripheral drivers without reading silicon documentation
  - Easy integration of an RTOS with peripheral drivers using RTOS adaptor component
  - Enables straightforward migration to new hardware
  - Can configure, package, and deliver components to your team
  - Available as part of the CodeWarrior tool suite or as an Eclipse-based plug-in feature for installation into an independent Eclipse environment.
  - Support for CodeWarrior, IAR, Keil and gcc build environments (Red Suite)

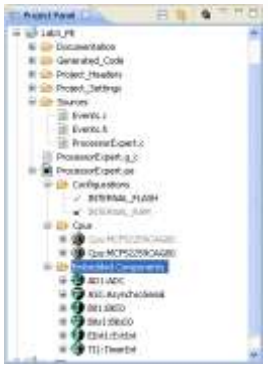
Embedded Components – mini code generators

- Embedded Components encapsulate the functionality of basic elements of embedded systems.
  - CPU core
  - CPU on-chip peripherals
  - Standalone peripherals
  - Virtual devices
  - Software – an RTOS, a library (e.g. FSL's touch sensing library)
- Processor components include support for the following architectures
  - ColdFire/ColdFire+
  - 56800/E (DSC)
  - Kinetis
  - RS08/S08
  - S12Z

Processor Expert Project Design

Create Project
- Create a new project with Processor Expert

Configure Components
- Use Inspector to set all component settings

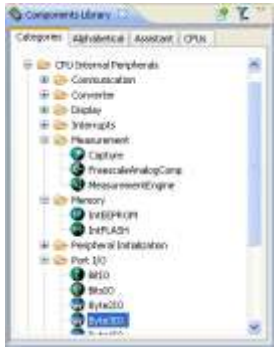


Generate Code
- Let Processor Expert generate all components drivers code

Build and Debug
- Build the application common way
- Debug the application with CodeWarriror

Processor Expert Project Design Timeline

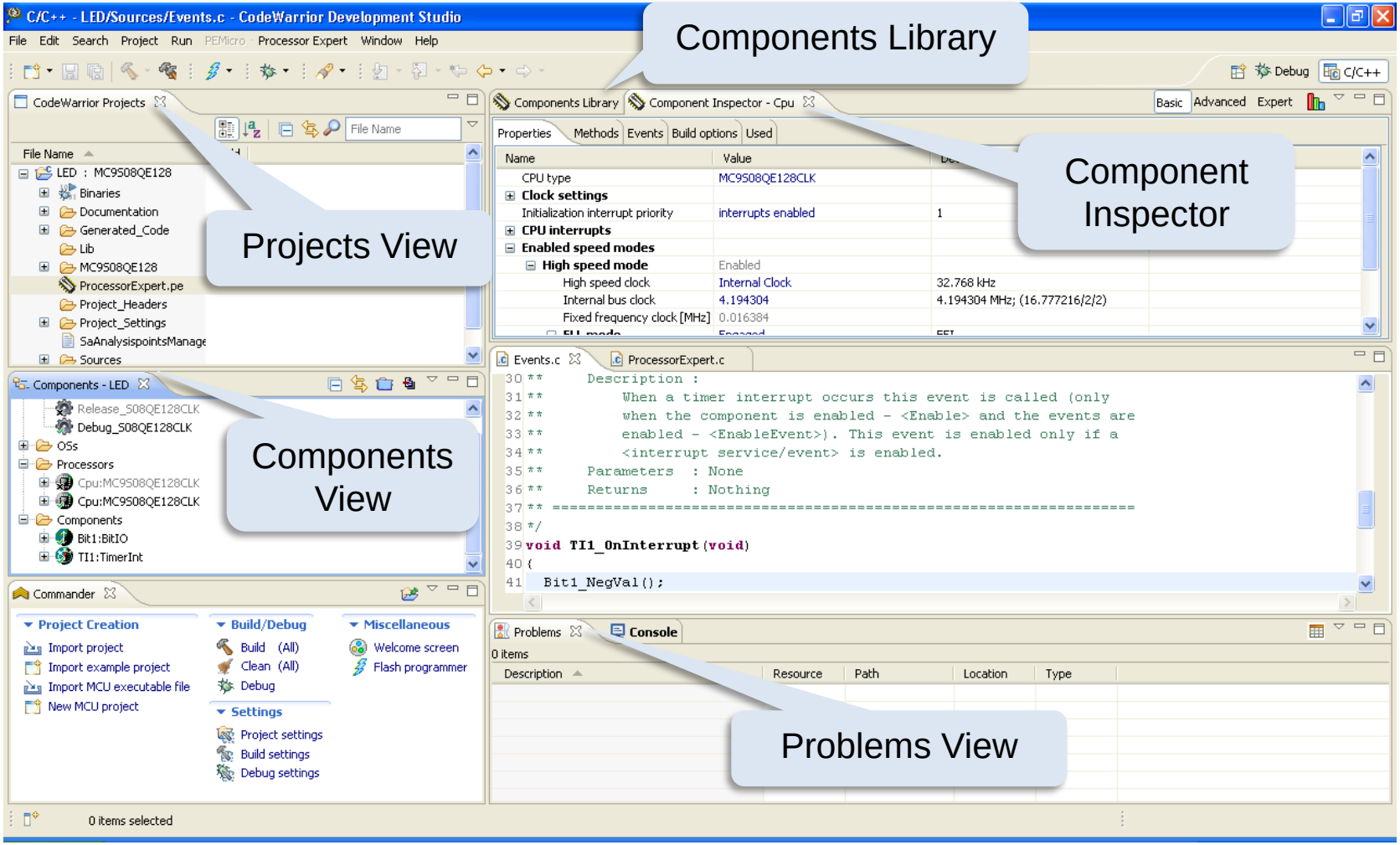
Add Components
- Select components required in the project from Components Library



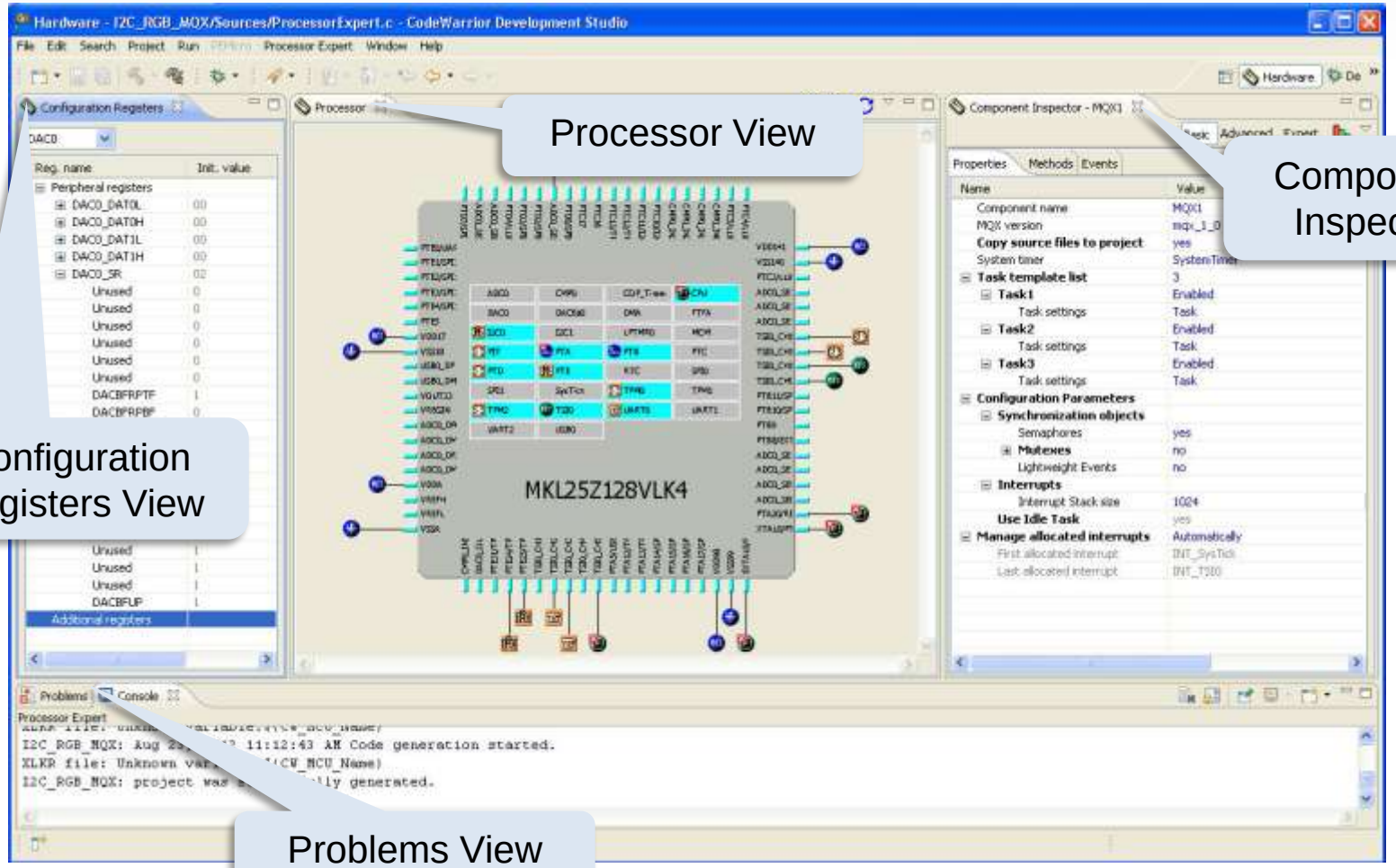
Verify Settings
- Make sure there are no design-time errors in the project

Write Code
- Write application code using code generated for components

Processor Expert Views – C/C++ Perspective

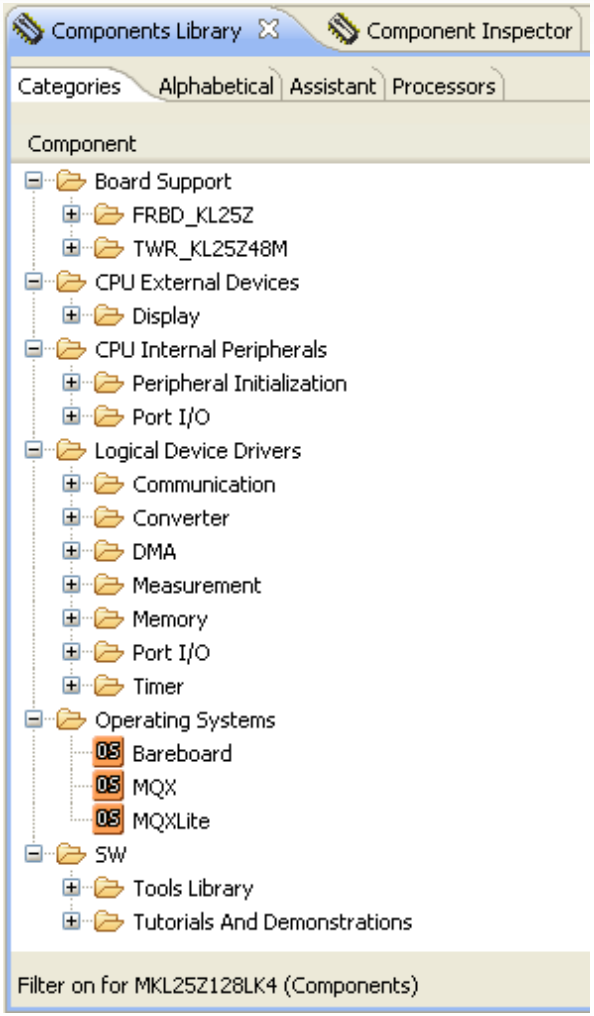


Processor Expert Views – Hardware Perspective

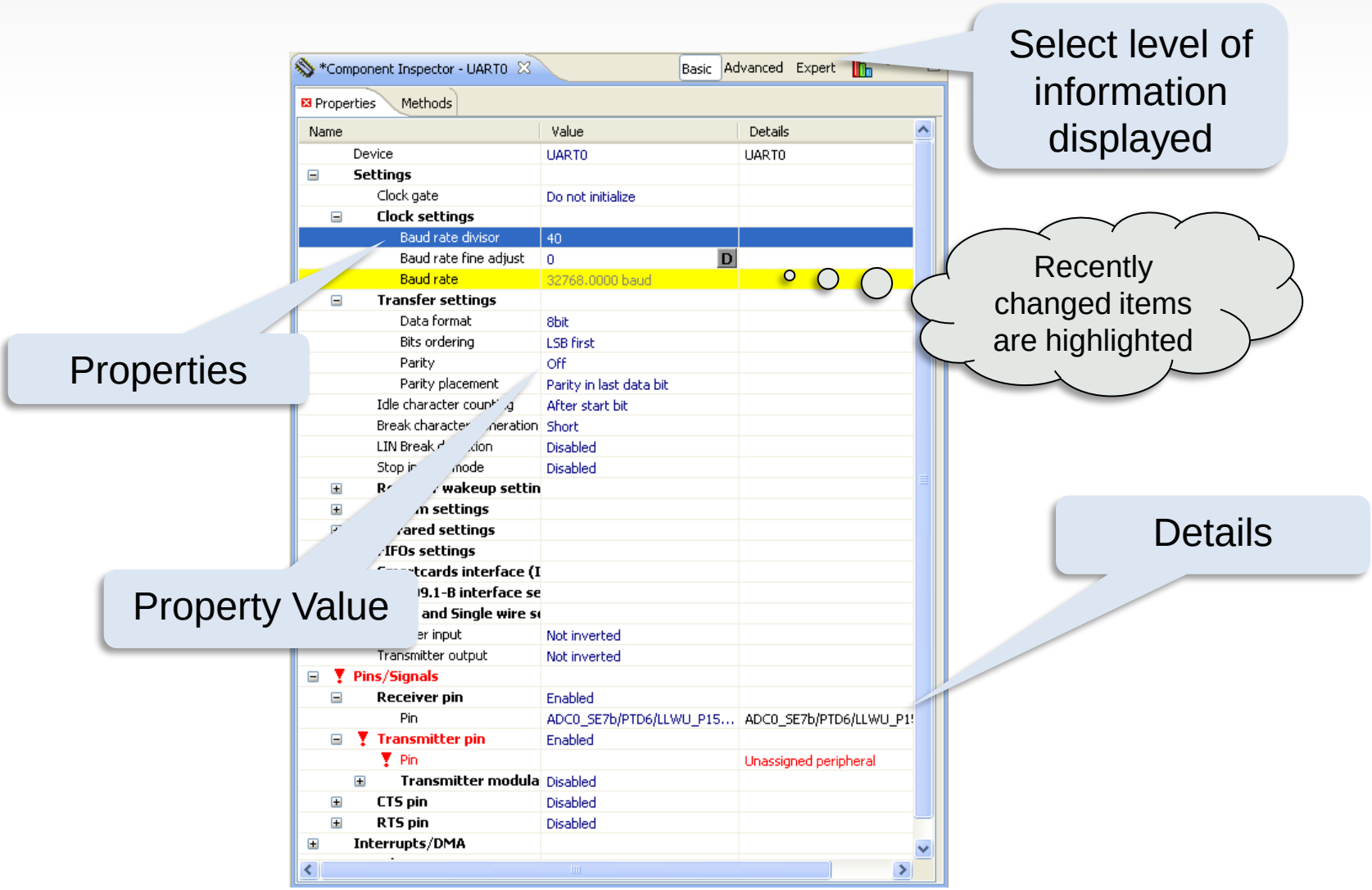


Components Library

- Available components are displayed in Components Library
- Select appropriate TAB to change how components are organized and displayed
  - Categories
  - Alphabetical
  - Assistant
  - Processors
- Select filter to toggle displayed components
  - Show only components available for selected processor.
  - Show all components available .

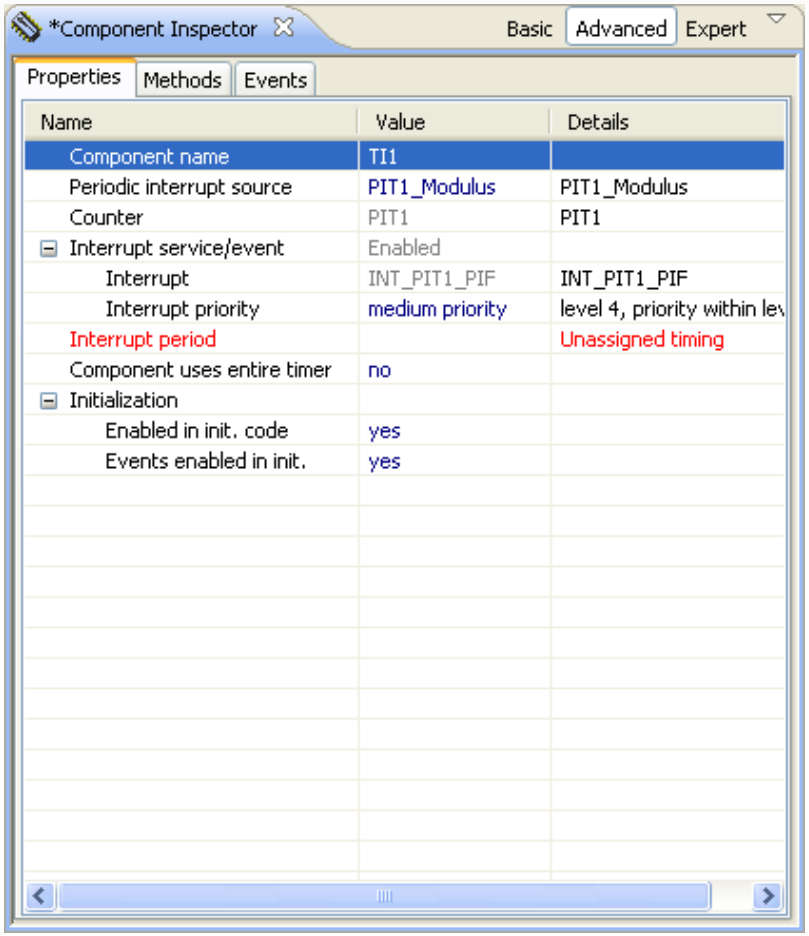


Component Inspector



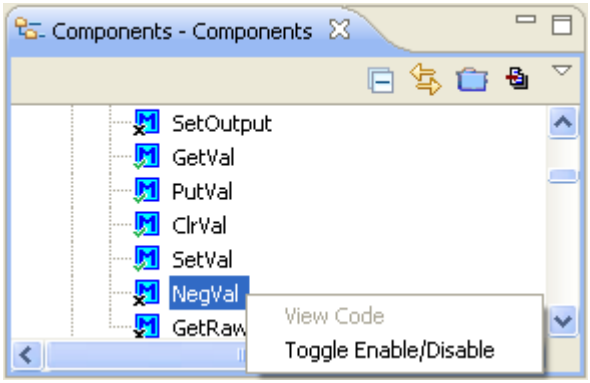
Component Inspector – Properties

- Properties define design-time settings
- Code is generated based on the property settings
- Recently changed properties are highlighted
- Properties that are undefined will be displayed in red text
- Properties with resource conflicts will be displayed in red text and tagged with an exclamation mark.
- Errors will be displayed in the *Problems View* and must be resolved before code can be generated.

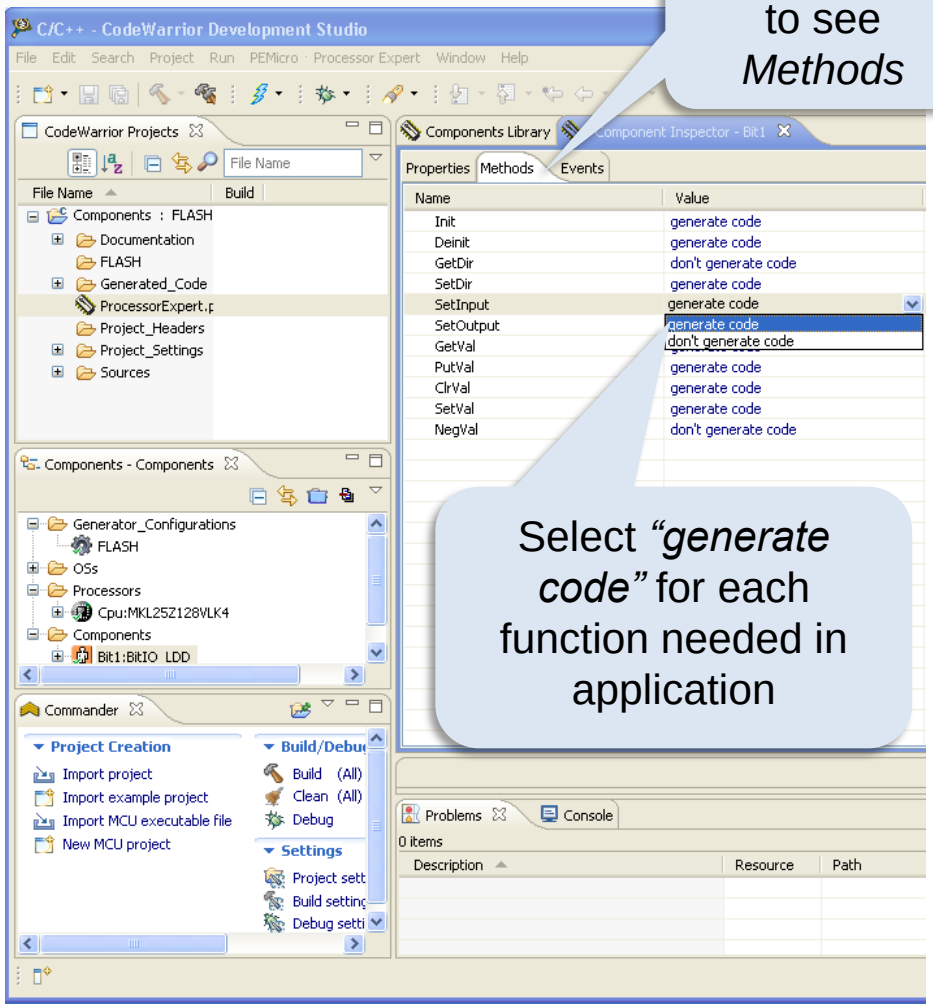


Component Inspector – Methods

- Select the *Methods TAB*
- Select “generate code” for each function required in the application
- Code will only be generated for the functions selected

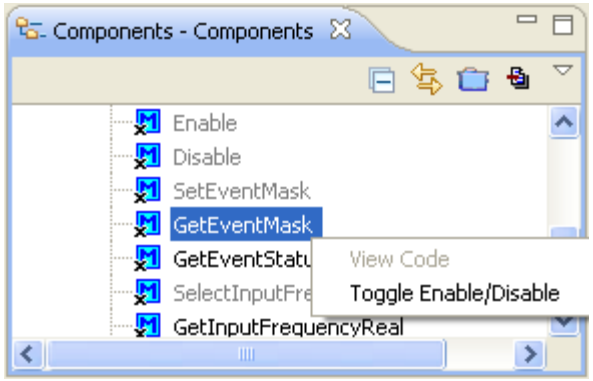


- To enable code generation for a method in the Components View, right-click on the method.
- Select *“Toggle Enable/Disable”*

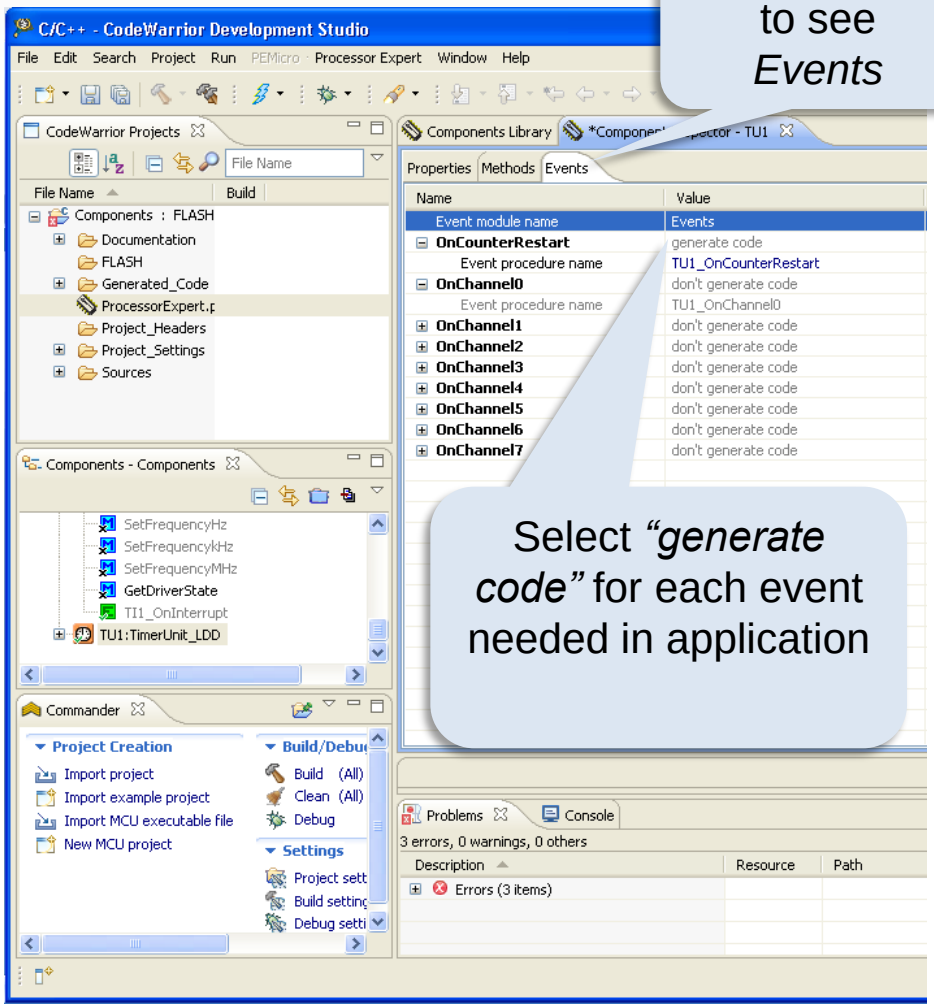


Component Inspector – Events

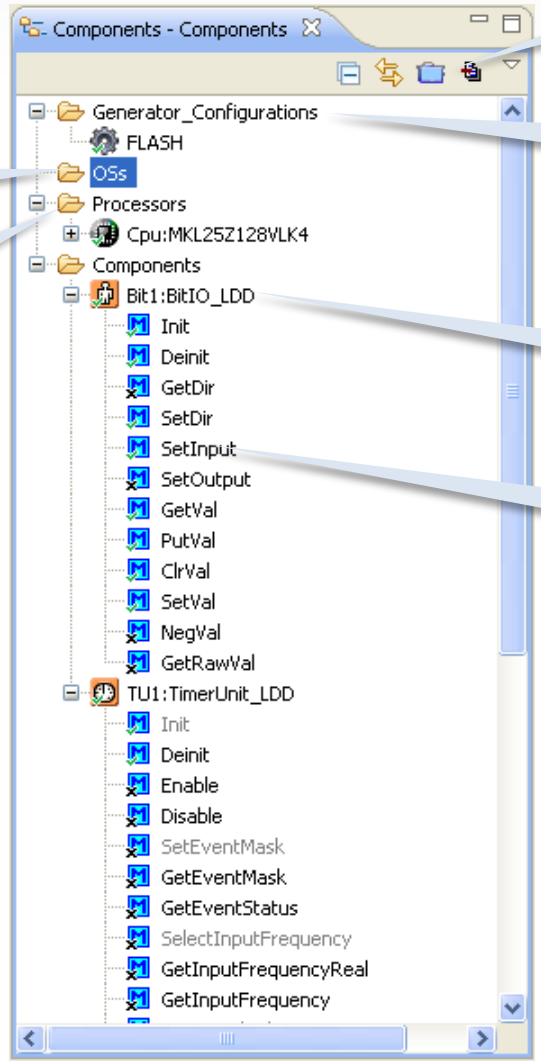
- Select the *Events TAB*
- Select “generate code” for each event required in the application
- Code will only be generated for the events selected



- To enable code generation for an event in the Components View, right-click on the event.
- Select *“Toggle Enable/Disable”*



Components View



Generate Code

Configurations

Operating Systems

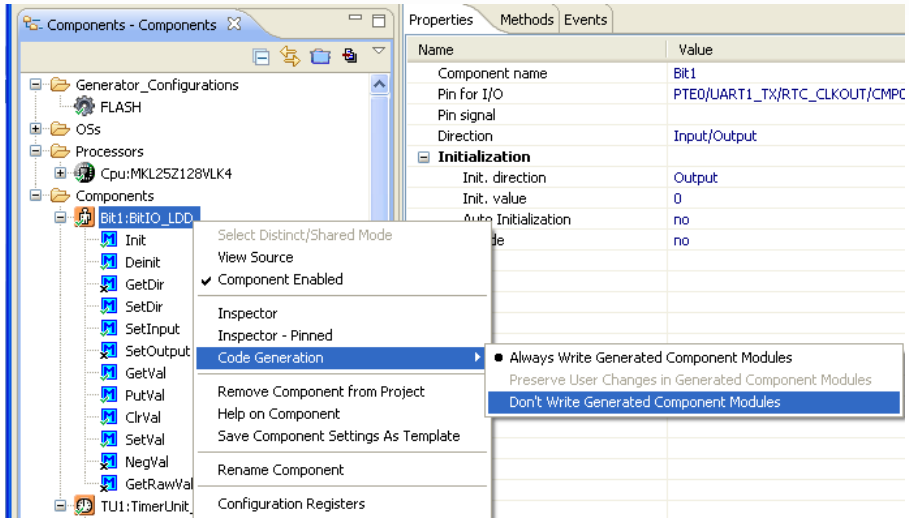
Processors

Components

Methods

Components View – Code Generation

- To generate code
  - Select *Generate Code* icon in the *Components View*
- To turn off code generation for a component
  - Right-click on component
  - Select *Code Generation*
  - Select *Don't write Generated Component Modules*



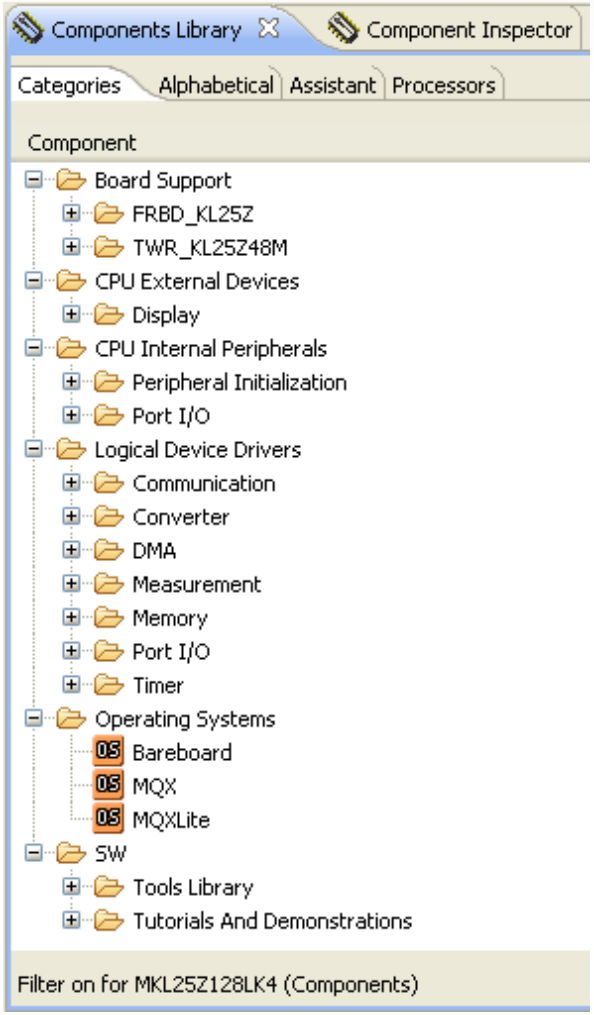
freescale™

Processor Expert Software

Embedded Components

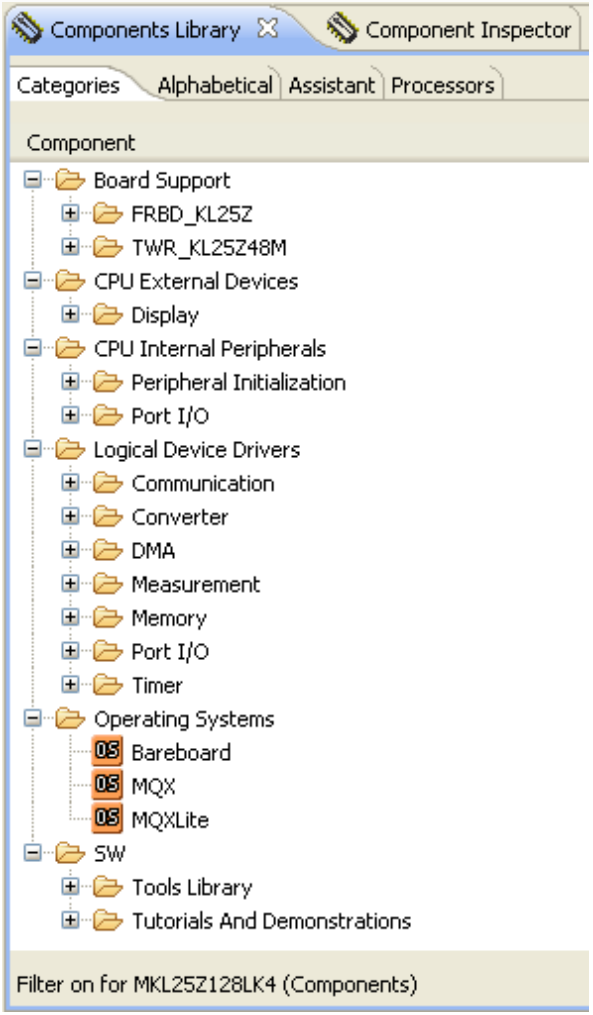
Component Categories

- **Board Support**
  - Configuration file (template) including
    - CPU component
    - One or more pre-configured peripheral components
- **CPU External Devices**
  - Embedded Component support for Console I/O, LED displays, etc.
- **CPU Internal Peripherals (High level)**
  - Embedded Component support for internal peripherals
  - Standard API ensures generated code is portable
  - Generates initialization code and low level device drivers.
- **Peripheral Initialization**
  - Hardware specific support, which is not portable
  - Generates initialization code only

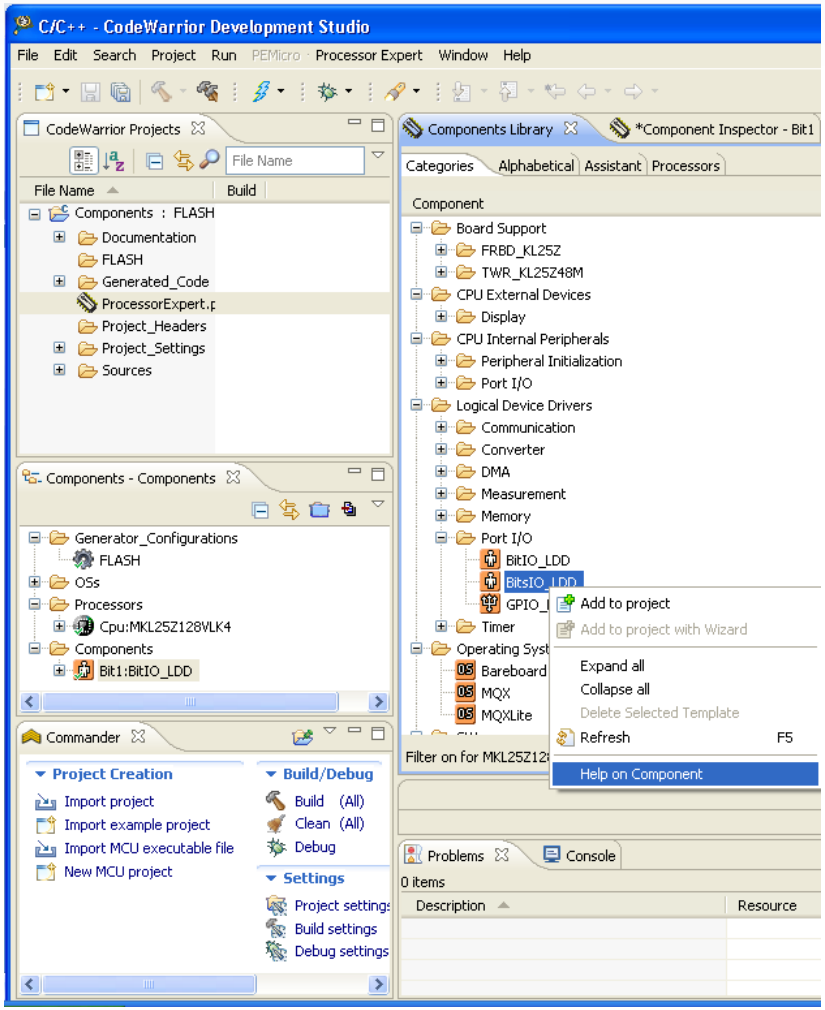


Component Categories

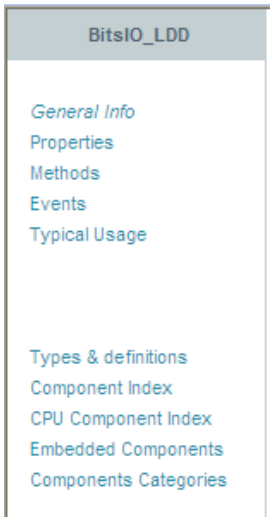
- **Logical Device Drivers**
  - Embedded Component support for internal peripherals
  - Uses a standard API, which allows generated device drivers to be adapted for used with bare-metal applications as well as RTOS applications.
- **Operating Systems**
  - RTOS adapter components
  - Adapter defines the RTOS API
    - How memory is allocated
    - How a critical section is created
    - How interrupt vectors are allocated
  - Generated device drivers are adapted to work with the RTOS API
- **Software**
  - Encapsulate software algorithms
  - Add functionality to an existing component



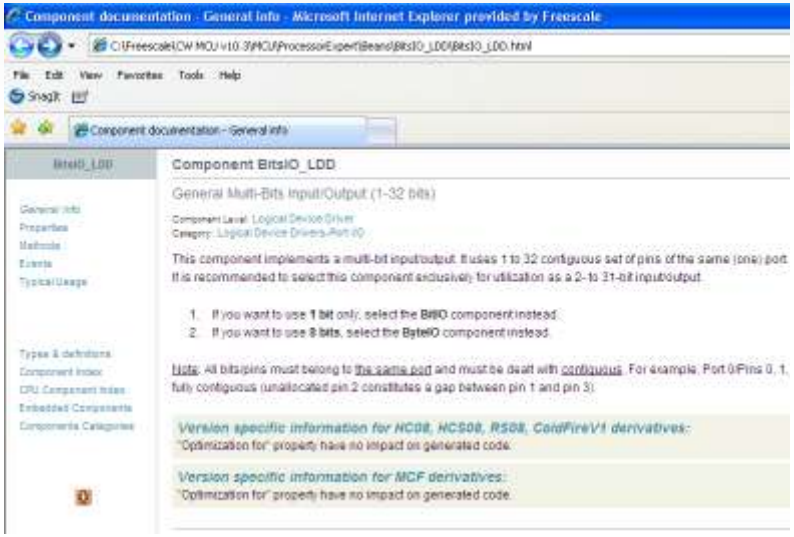
Help on Component

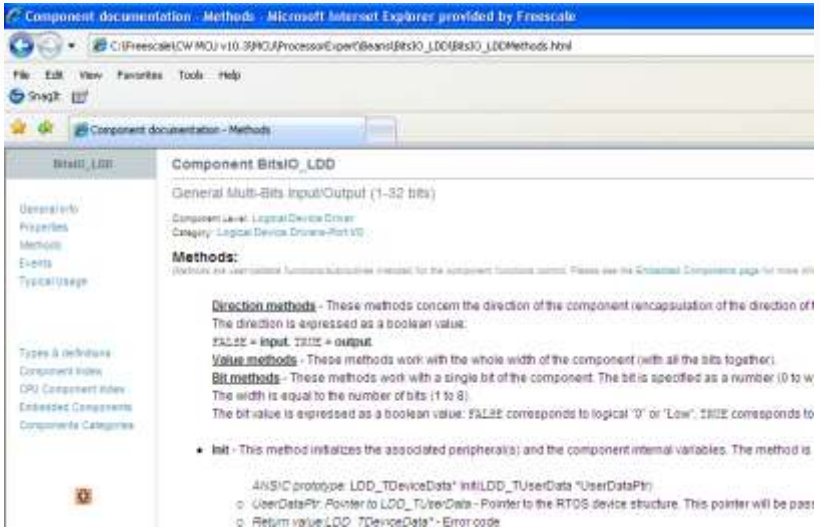


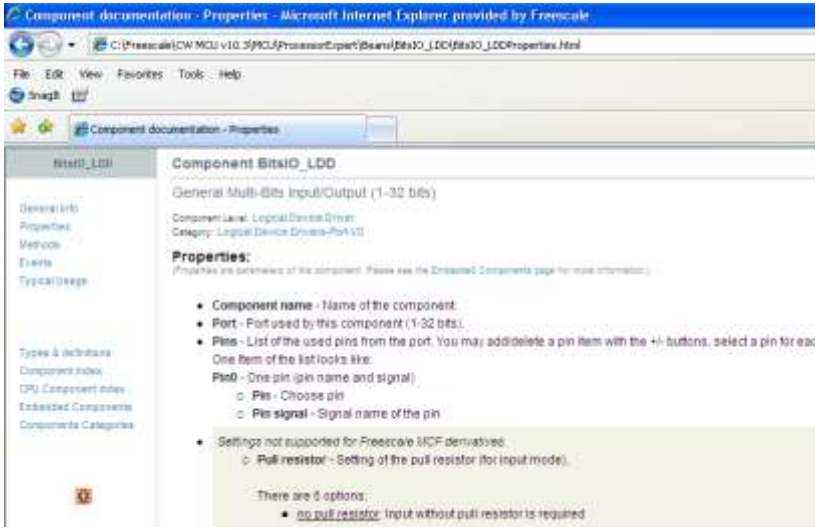
- Right-click on the Component in the *Components Library*
- Select *Help on Component* in the pop-up menu
- Component documentation will be displayed with the following topics
  - General Info
  - Properties
  - Methods
  - Events
  - Typical Usage

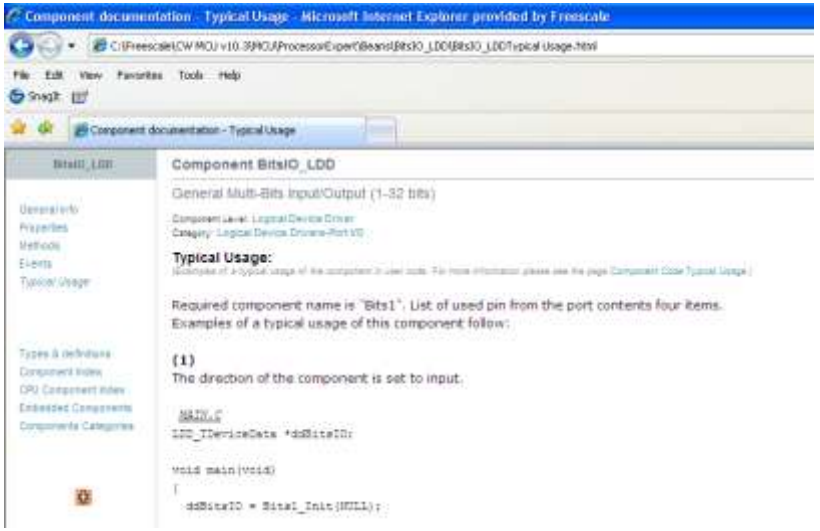


Help on Component









Lab 1

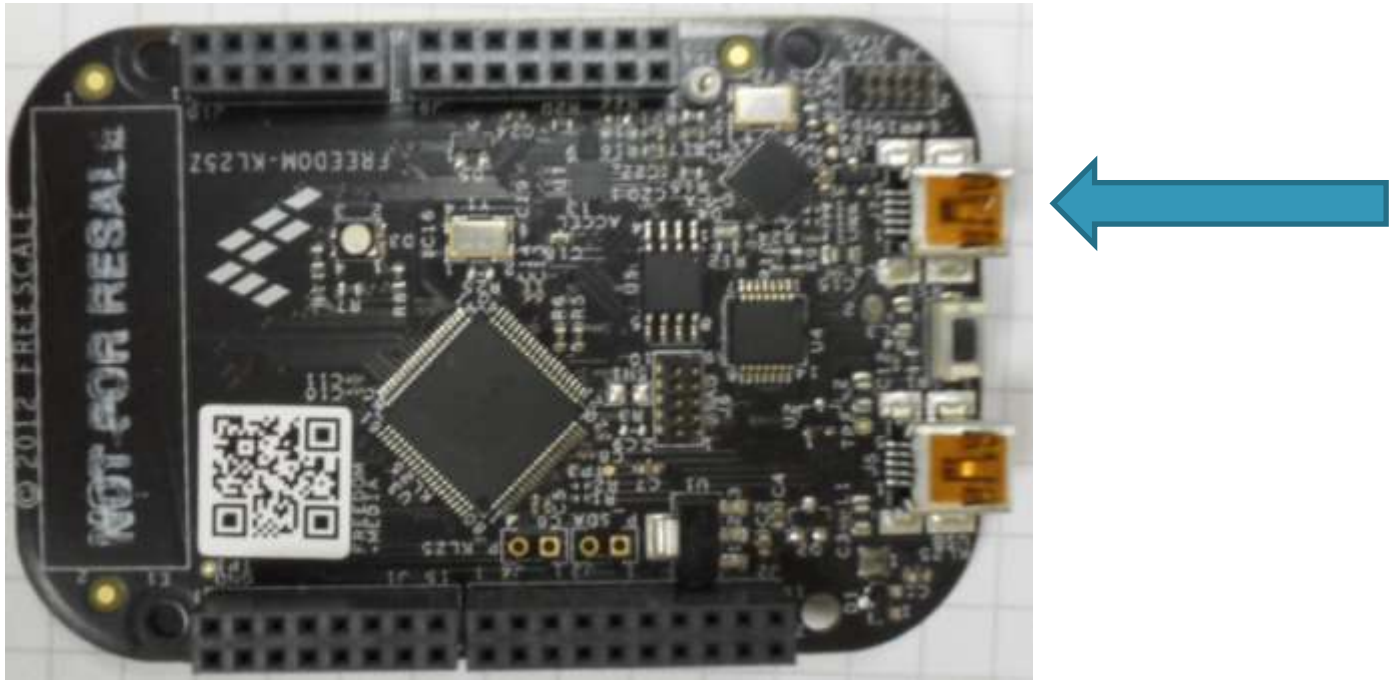
Making an LED Blink Using CodeWarrior

Simple Demo – Blink an LED

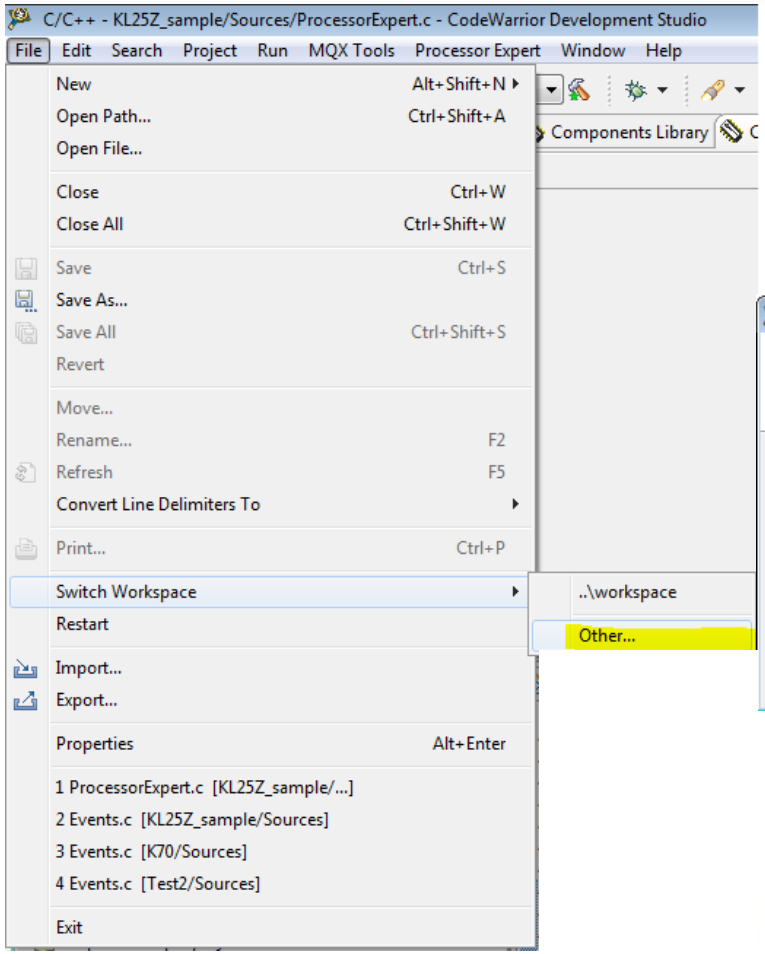
- Connect the Freedom Board demo
  - Use USB cable to connect the Freedom board
- Build a new project for the KL25Z128
  - Using CodeWarrior for Microcontrollers on your Laptop
  - Add an interval timer driver
  - Add a GPIO driver for one output bit configured to the GPIO bit for blue LED on Freedom board (PTD1 pin)
  - And we'll do it without writing a line of code.

Connect the Freedom board

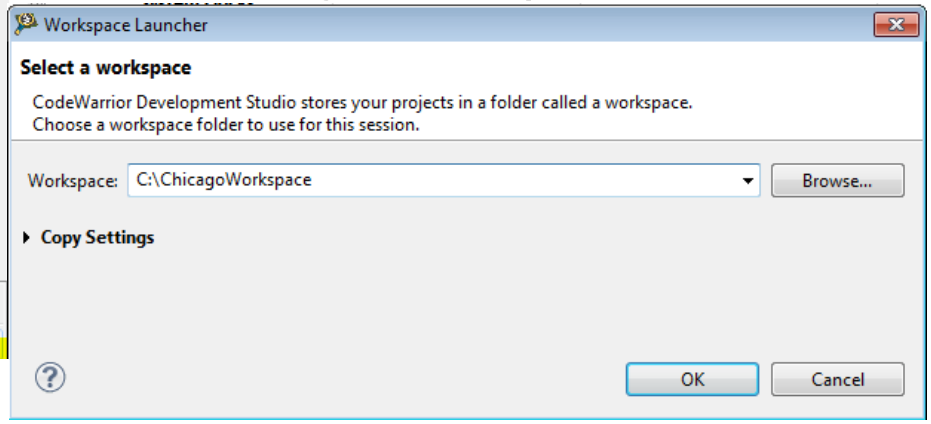
- Connect to the OpenSDA USB port on the Freedom board…



Open CodeWarrior



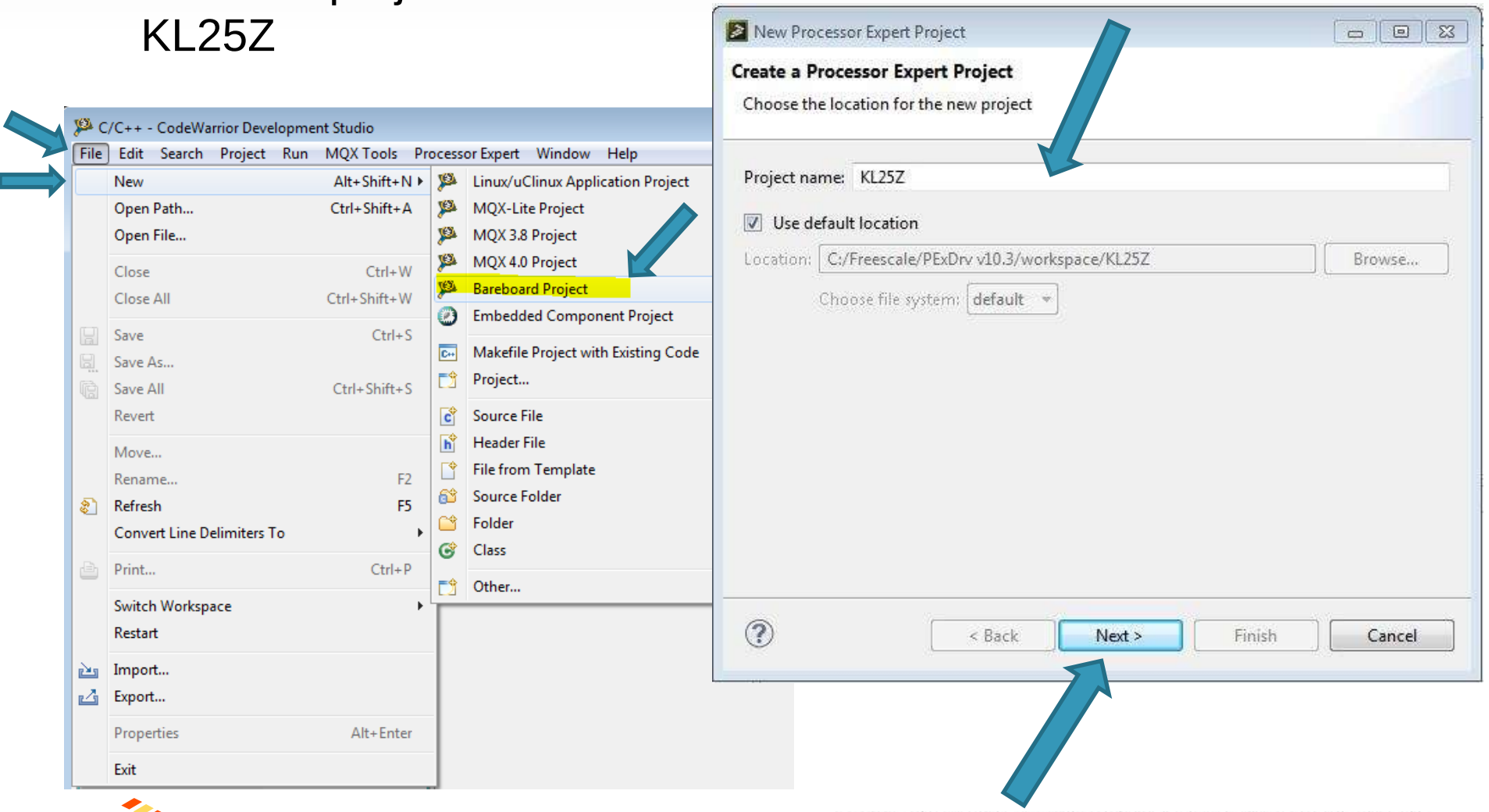
- Open the CodeWarrior on your laptops
- Create a new Workspace c:\ChicagoWorkspace



- This gives us a clean workspace to do our labs…
  - Note: workspaces are specific to versions of Eclipse and CodeWarrior.

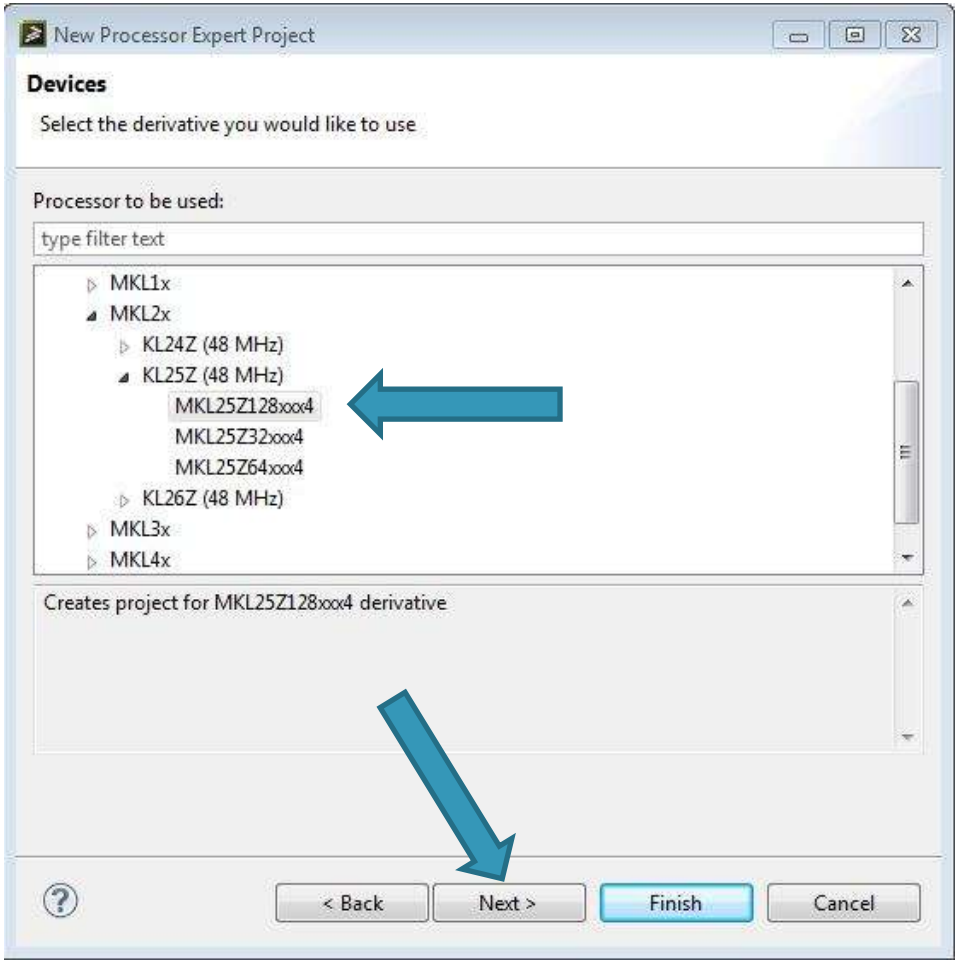
Create a new Project

- Create a project named KL25Z



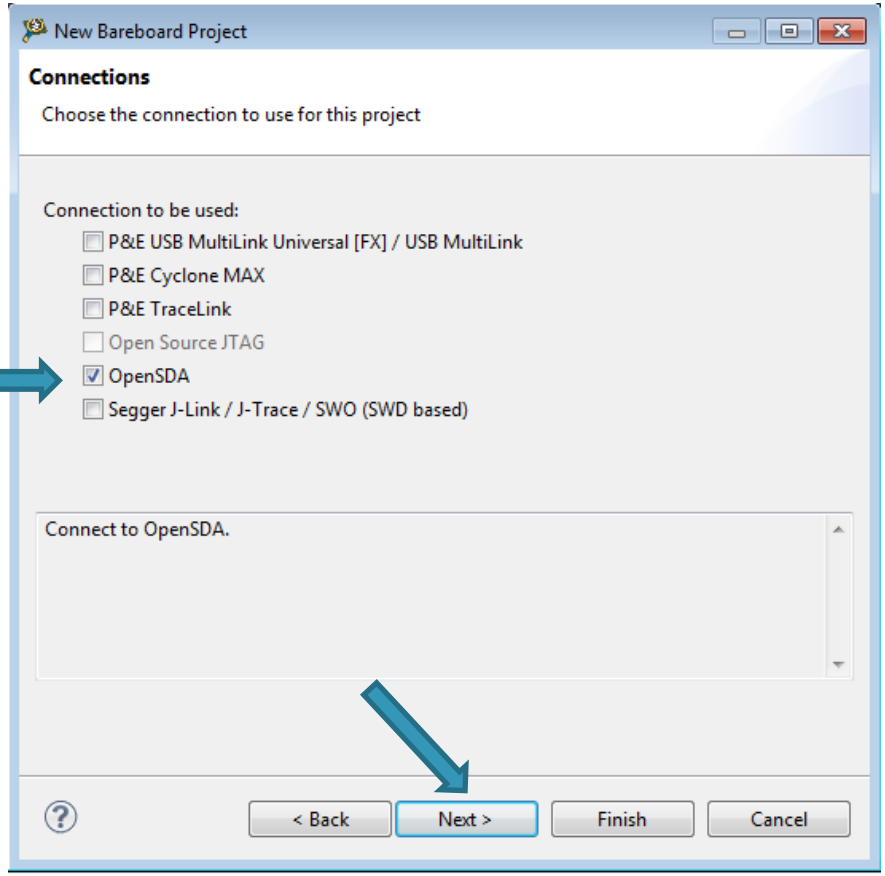
Choose the processor to be used

- Choose the MLK25Z128 part

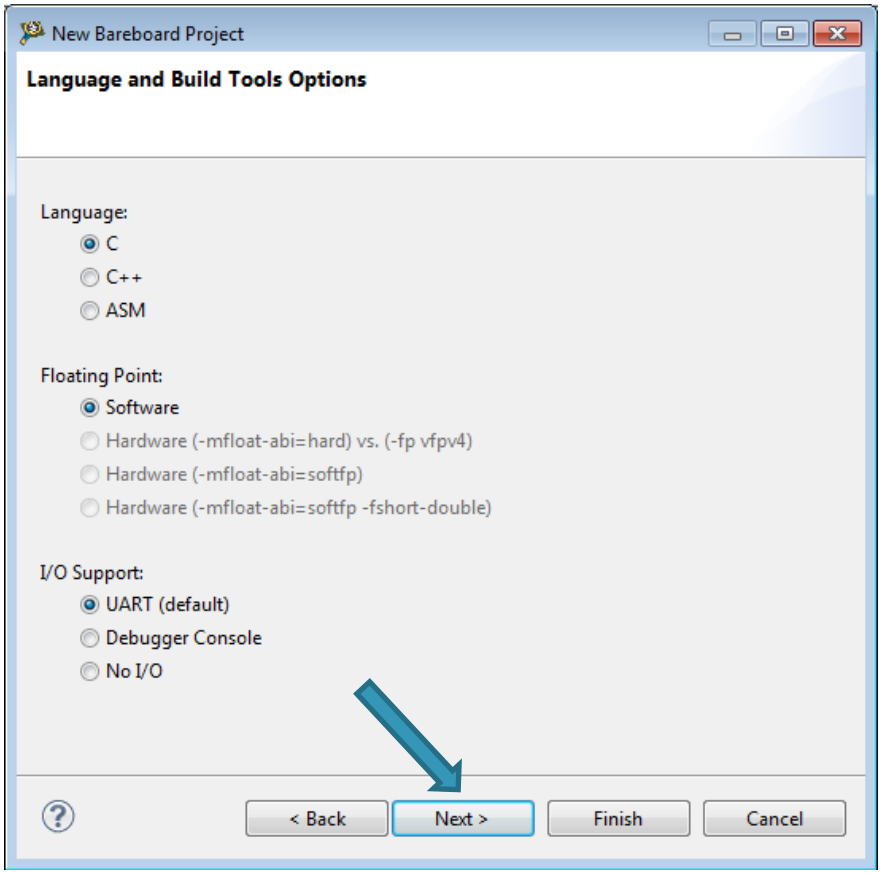


Define the CodeWarrior Specifics

- For the Driver Suite, we don't setup debug connections

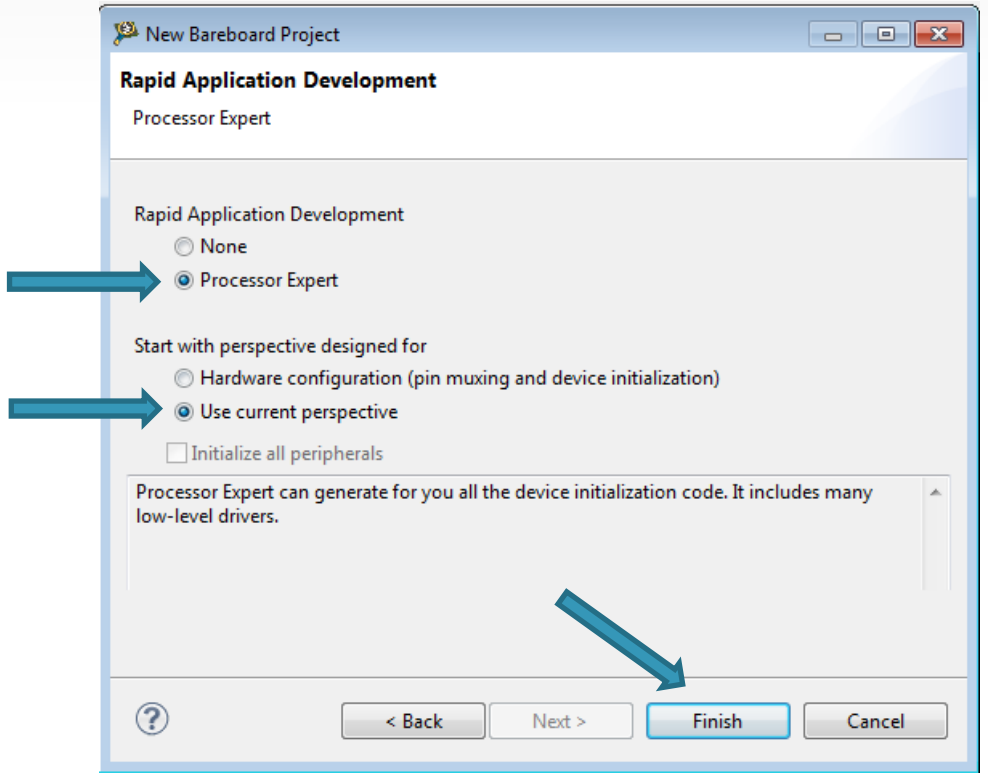


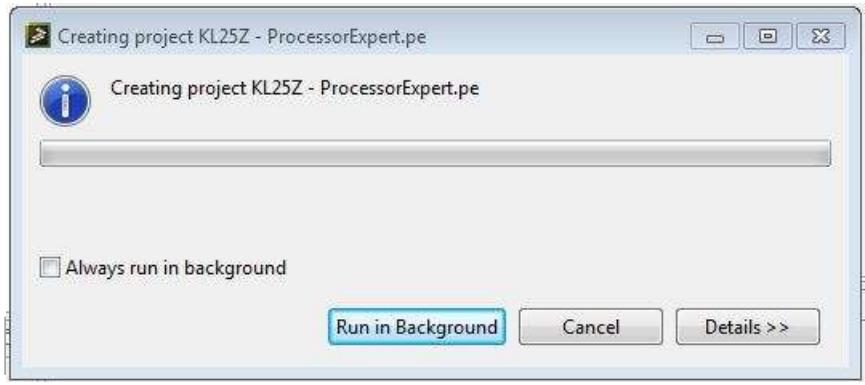
- Or CodeWarrior build specifics



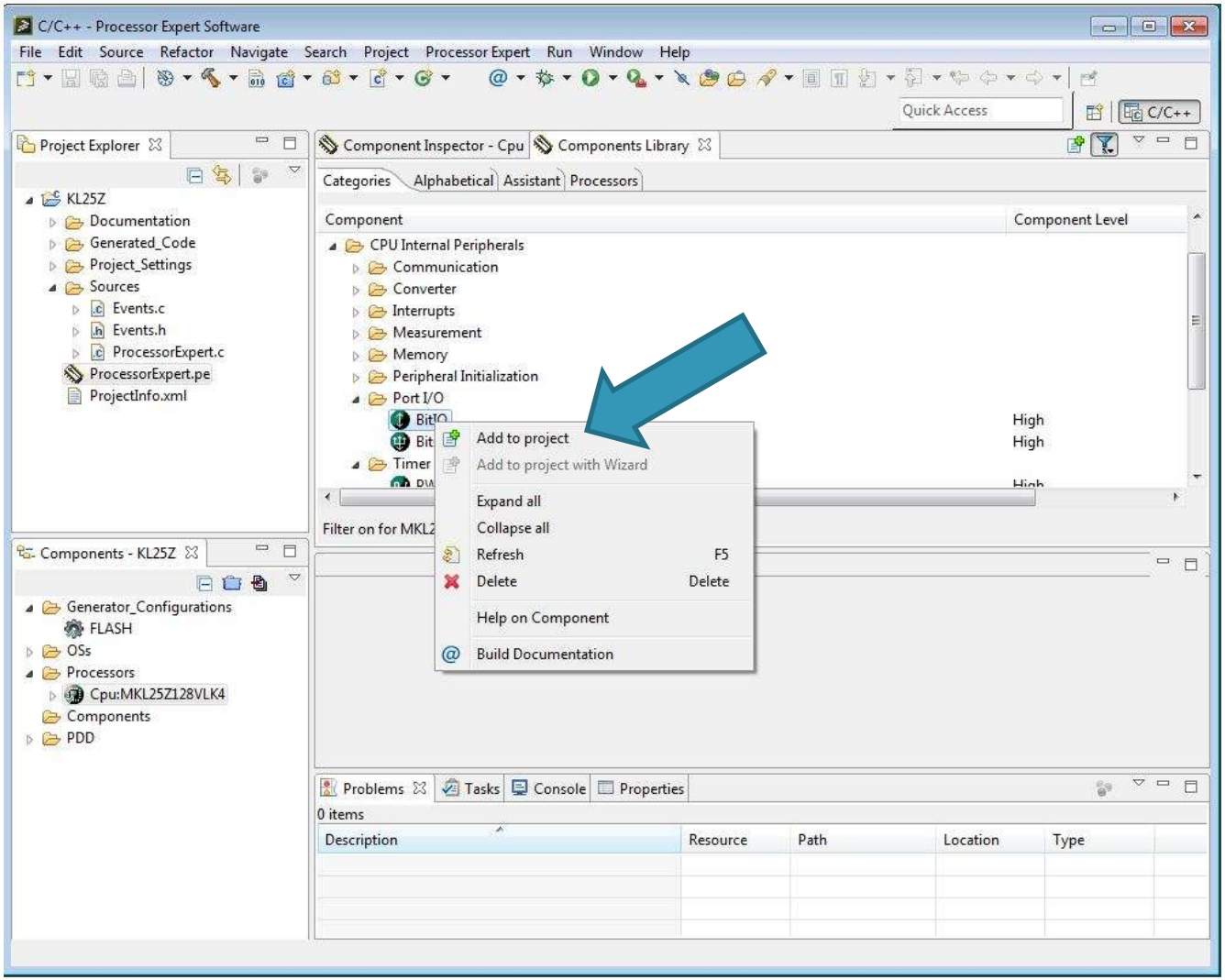
CodeWarrior Perspectives

- For CodeWarrior projects, you must choose to use Processor Expert
- And select the proper perspective
- Note: for Driver Suite, you only choose the Perspective since the wizard is for creating a New Processor Expert Project.

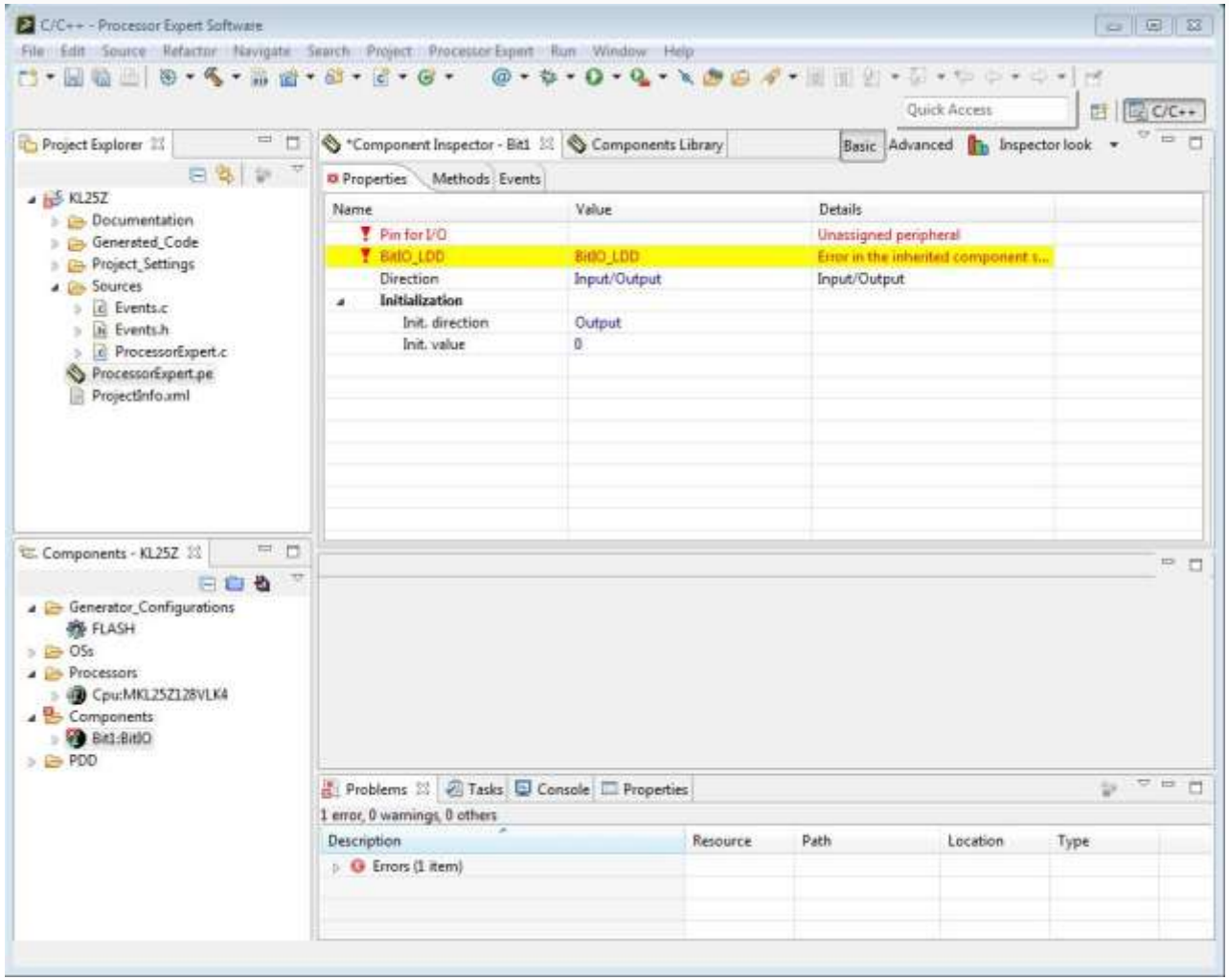




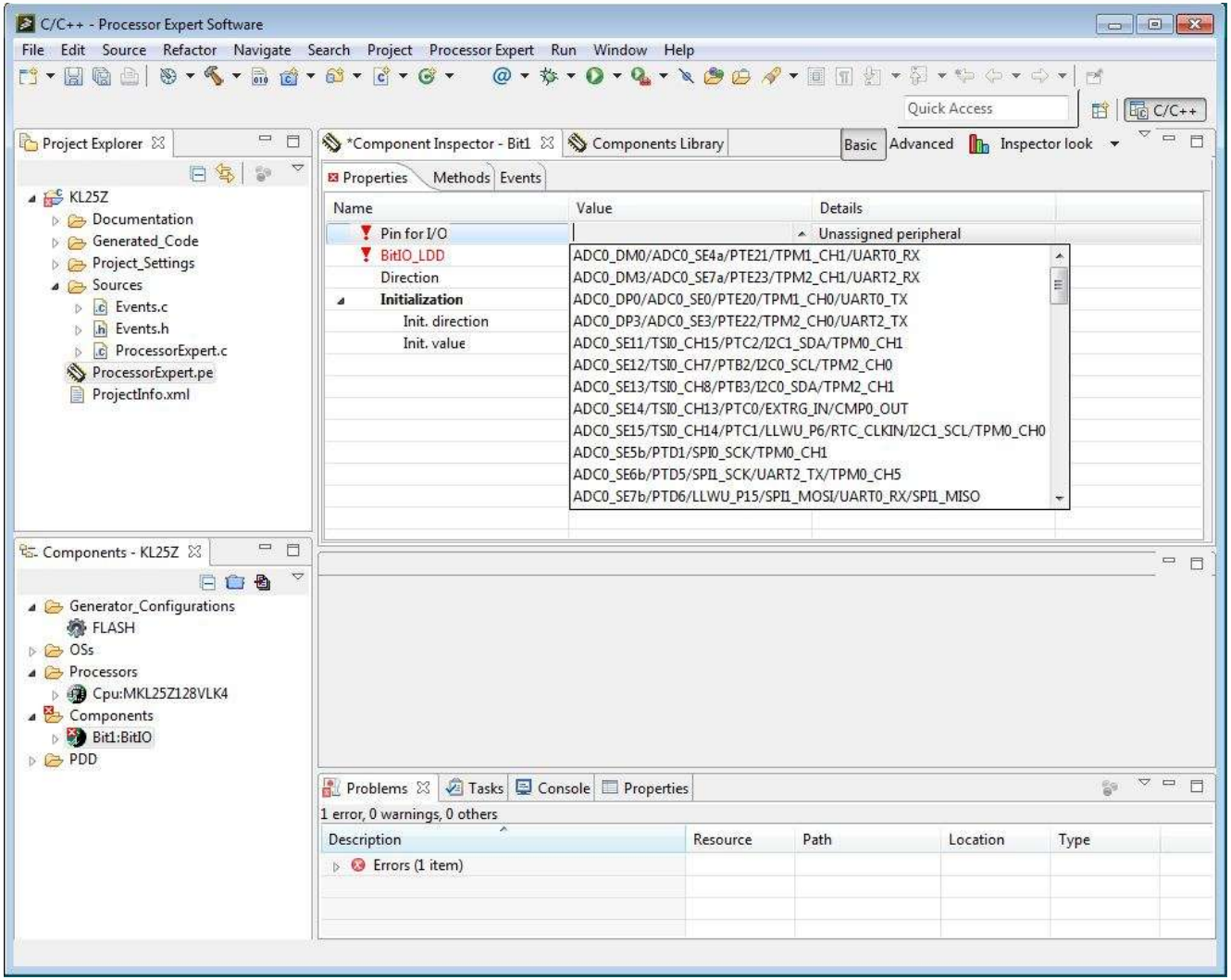
Add the peripheral driver for a single bit GPIO



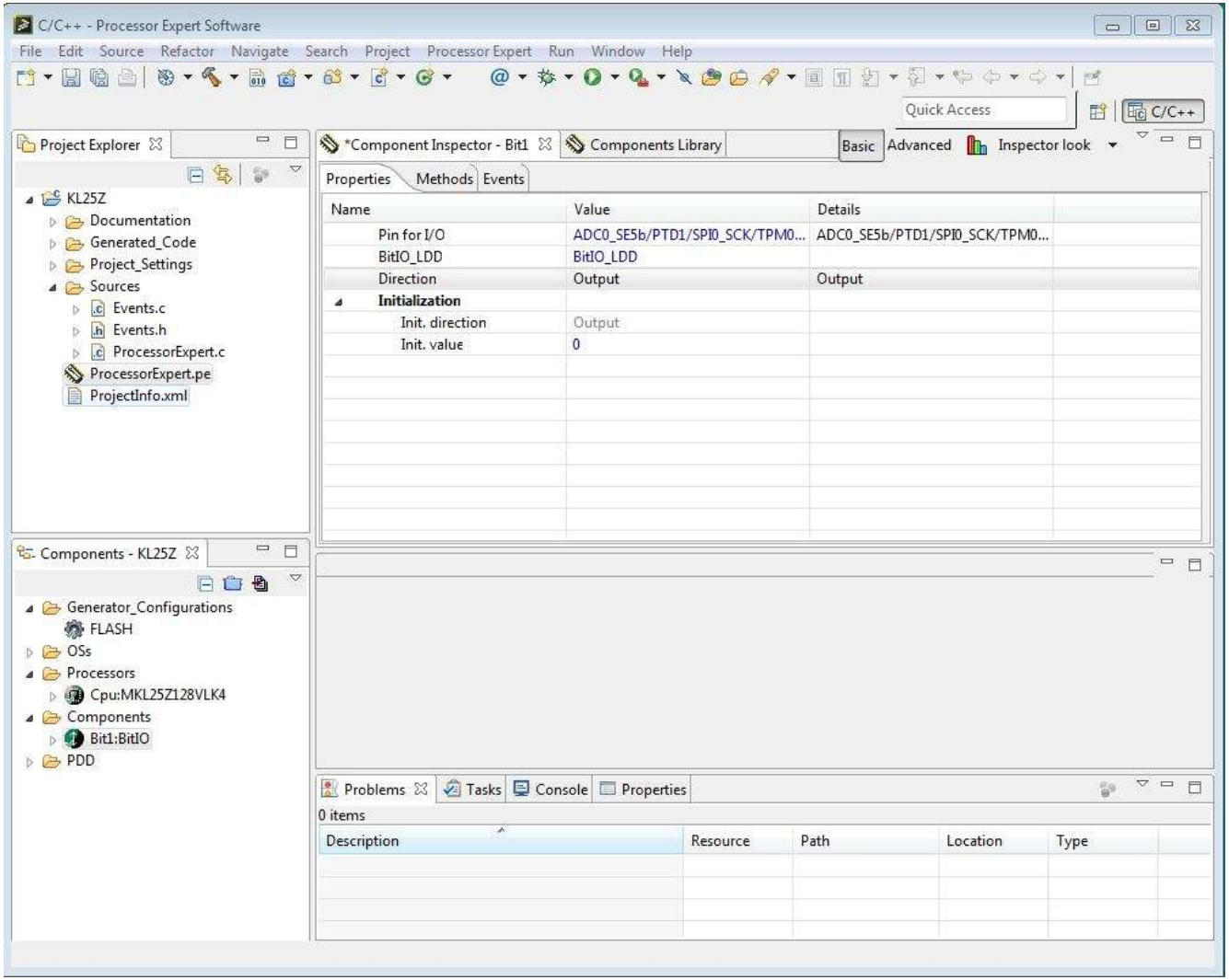
Component Inspector – set the pin to use



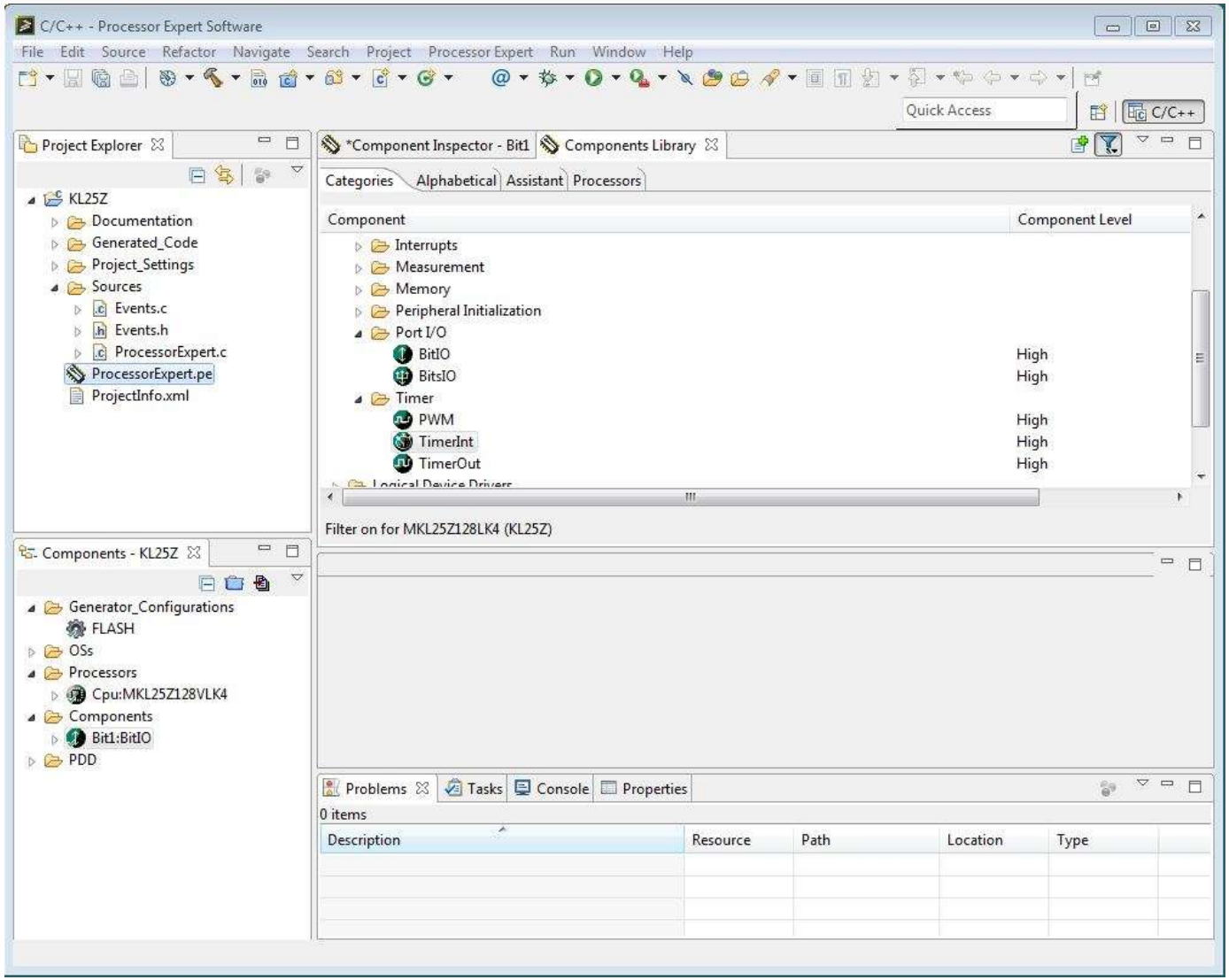
Choose pin to be ADC0_SE5b/PTD1/…



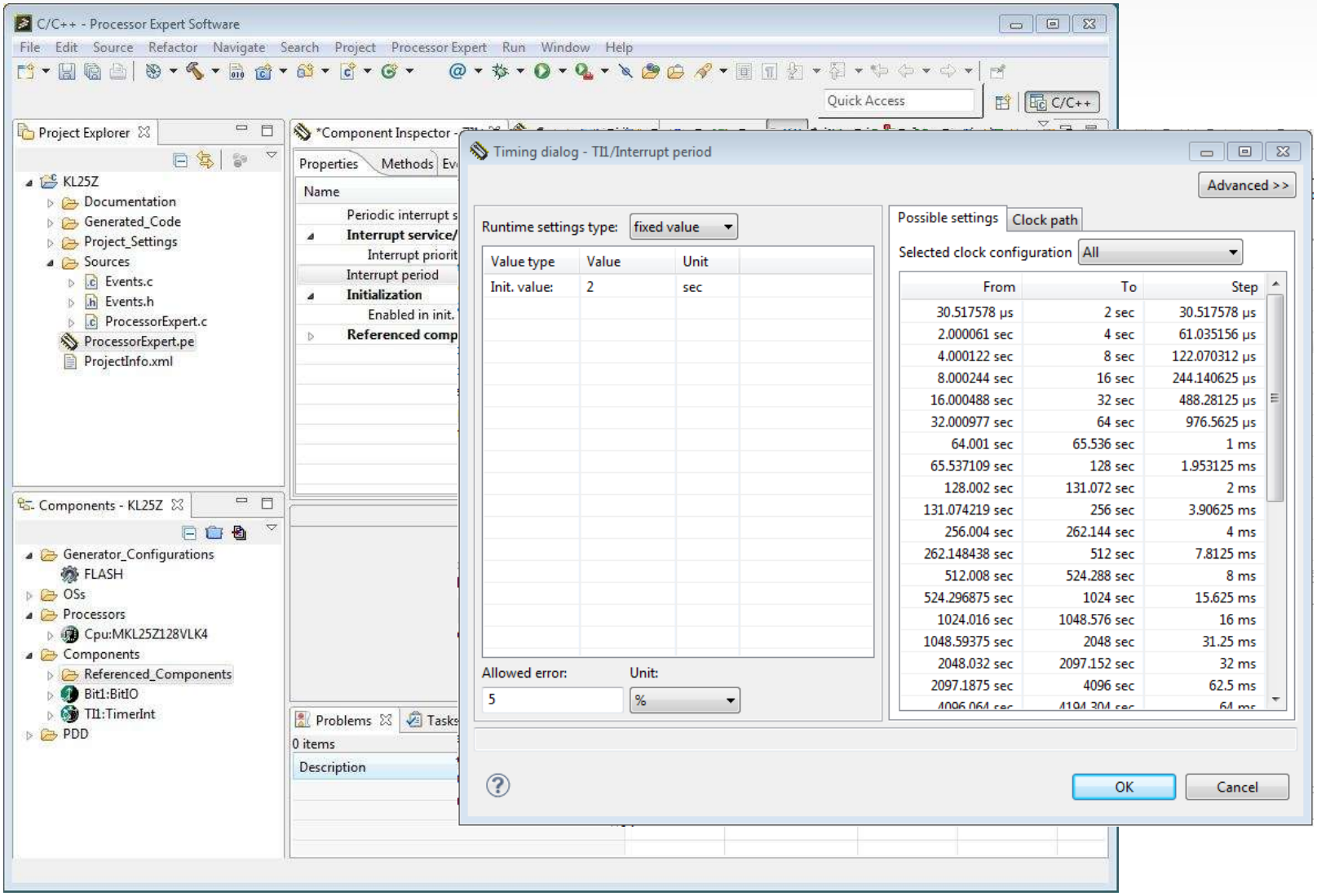
Define the GPIO direction to Output



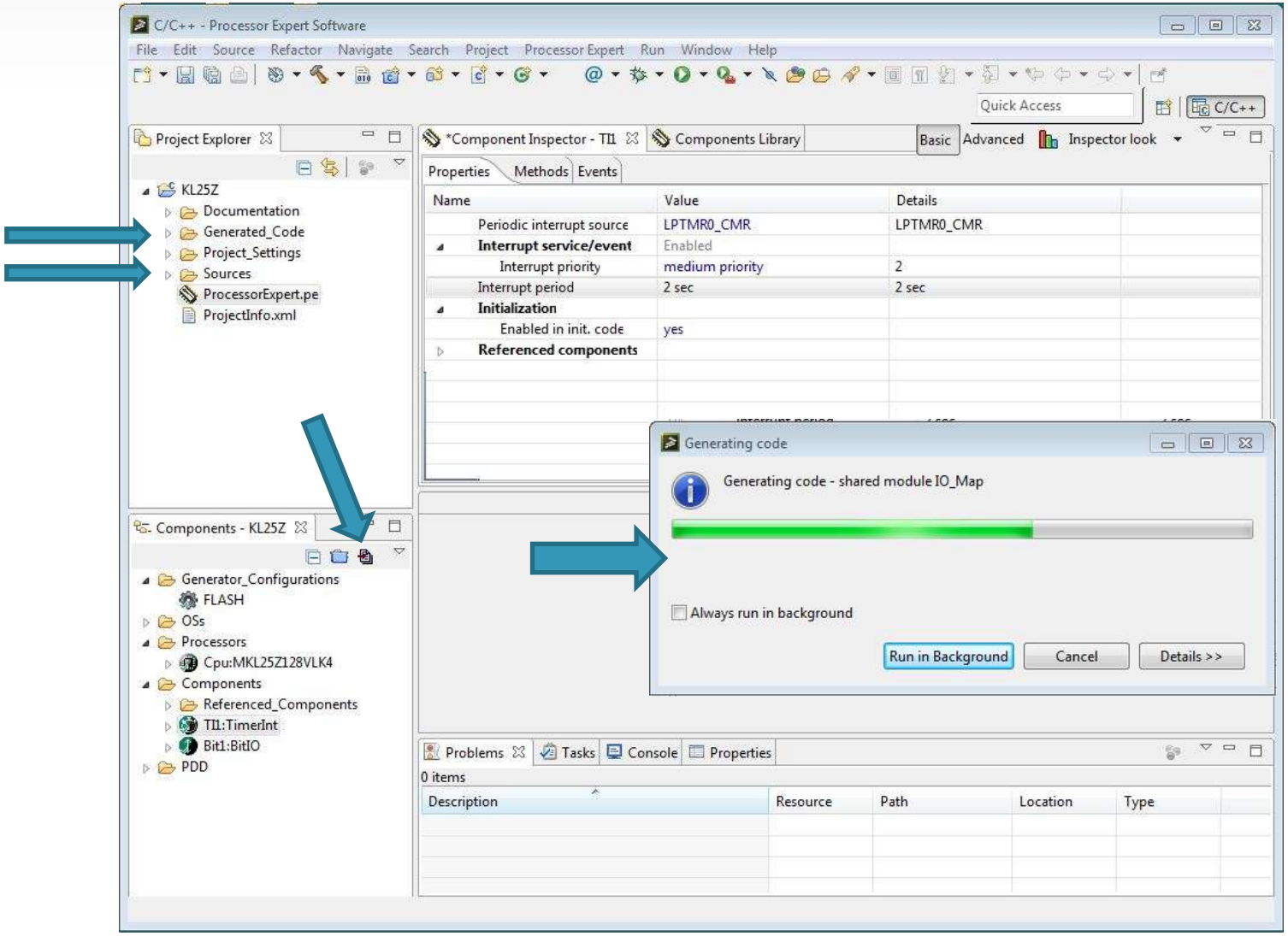
Now add the TimerInt (Interval Timer)



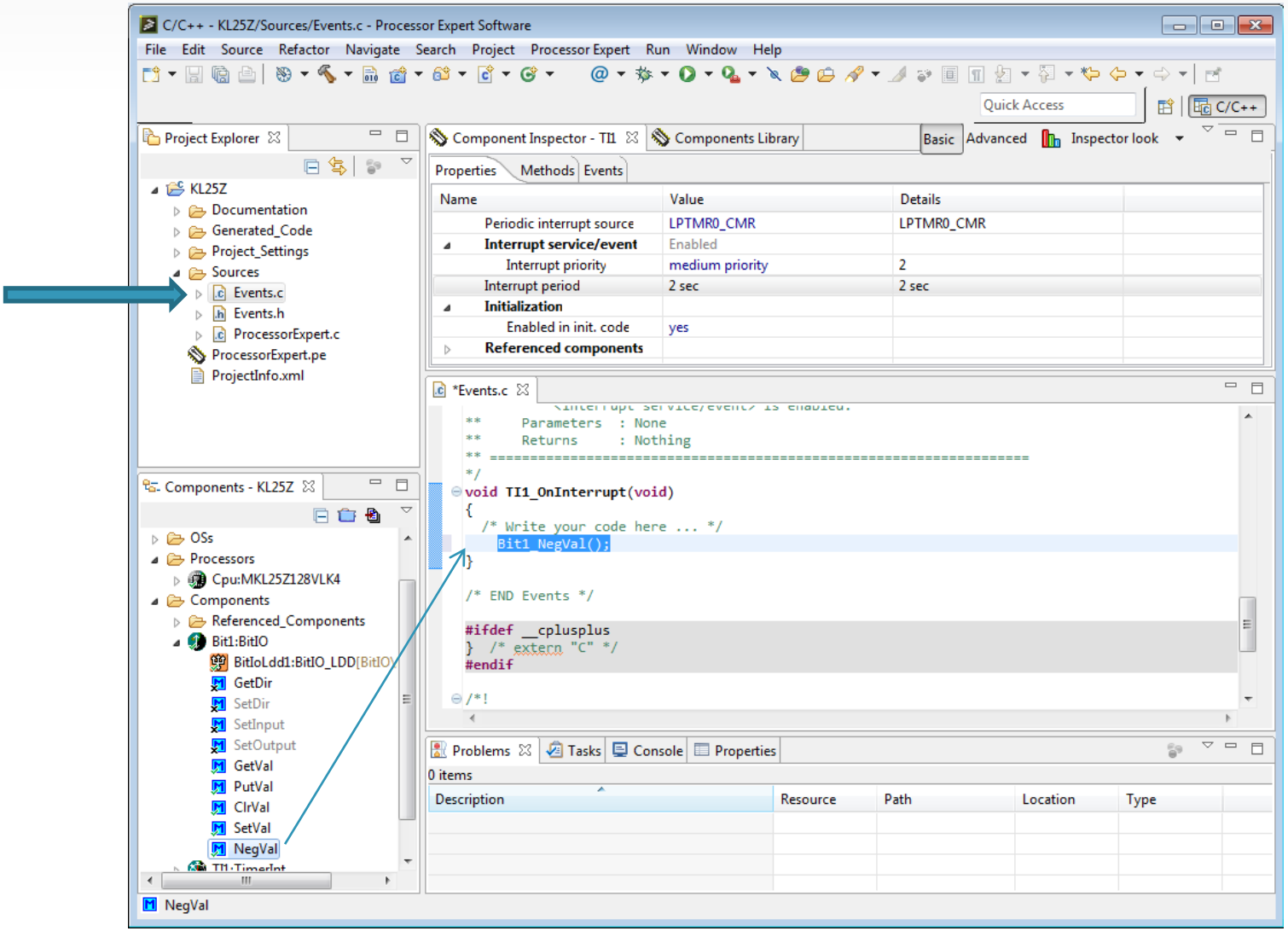
Set the timing to 2 seconds



Generate all source code



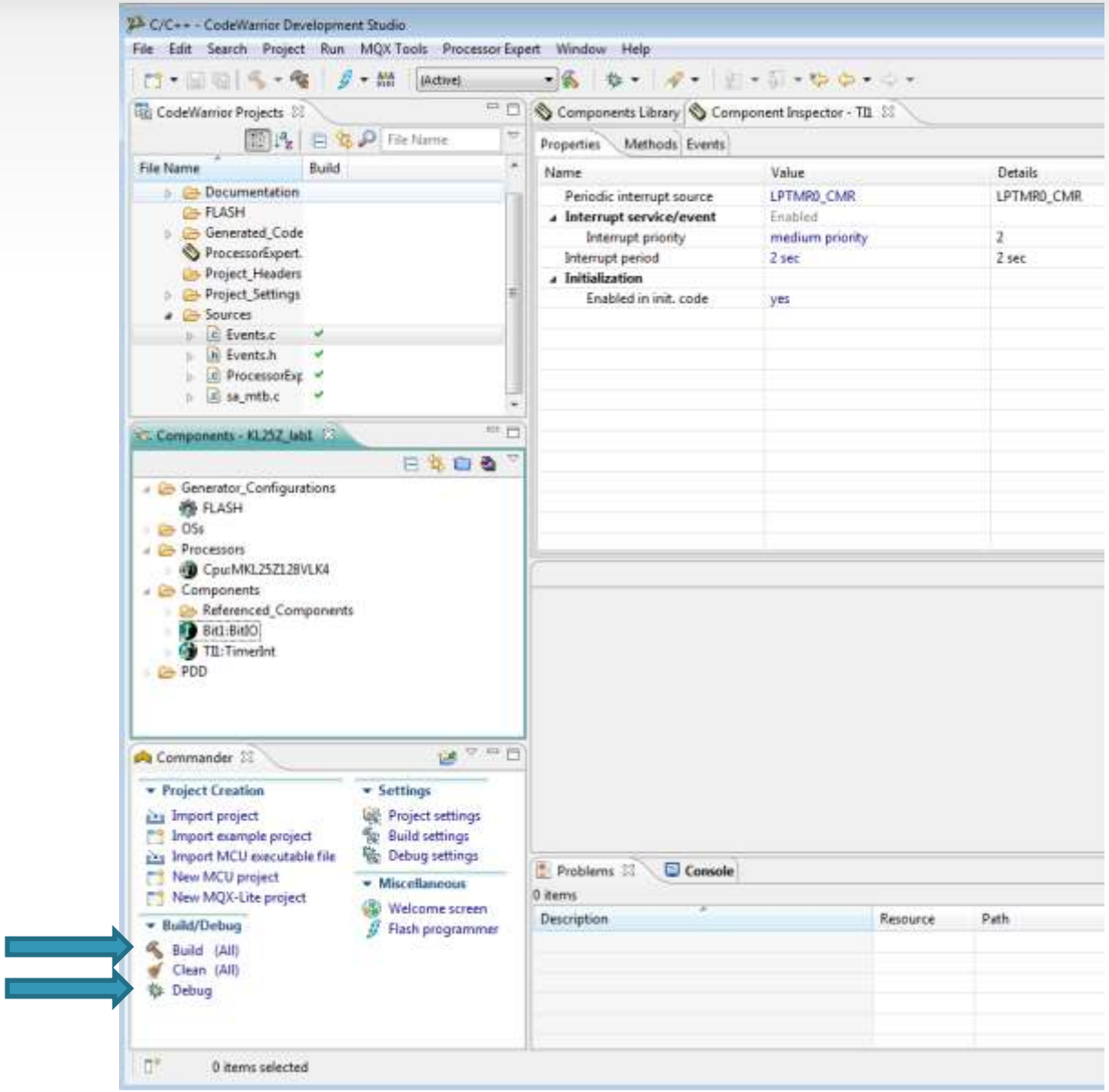
Add the NegVal() method by drag/drop



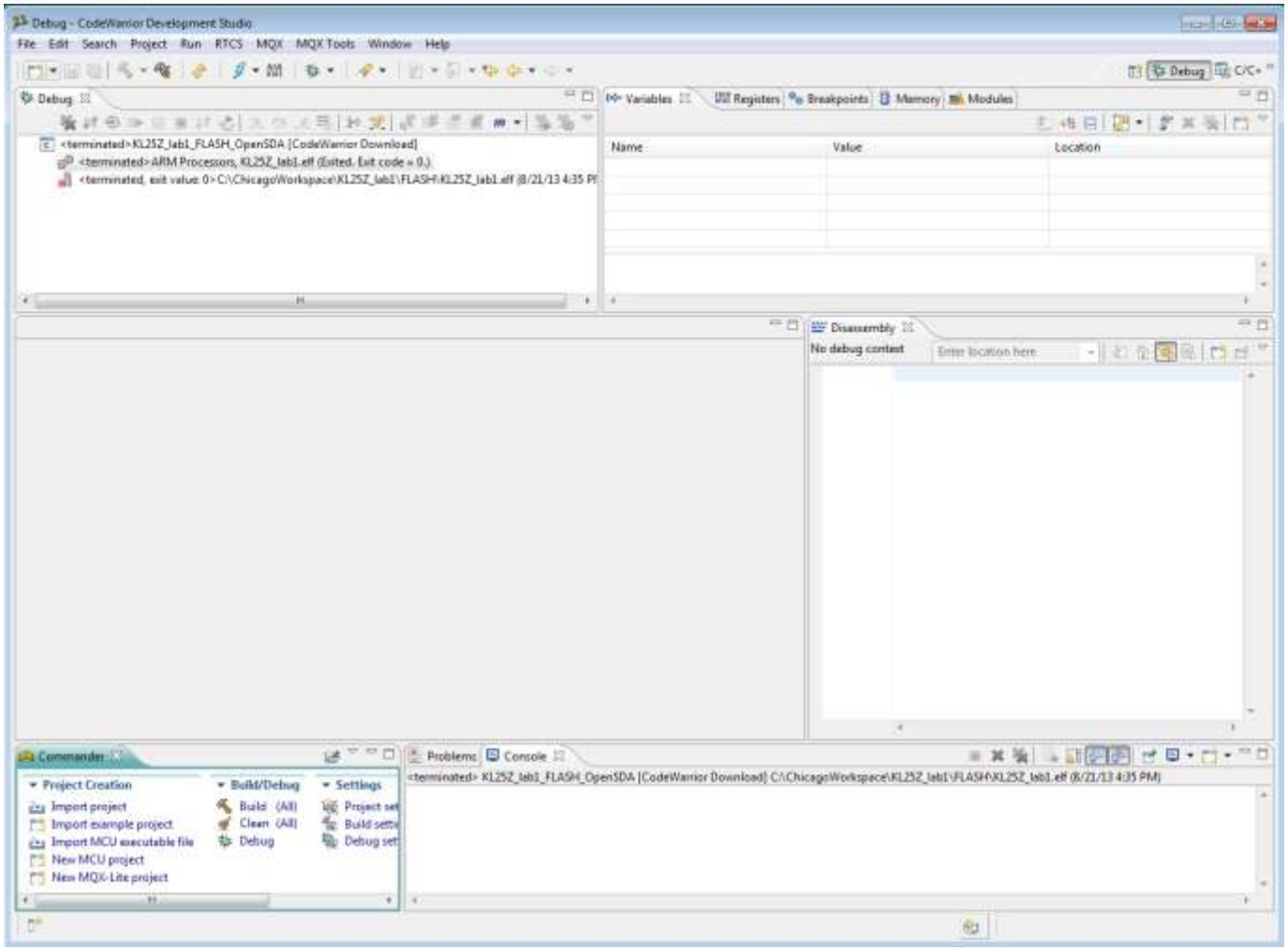
Build the project

- Build the project
- Debug the project

- Note: Build will also force a Generate Code if any of the components were modified.



Debug the project



Lab 2

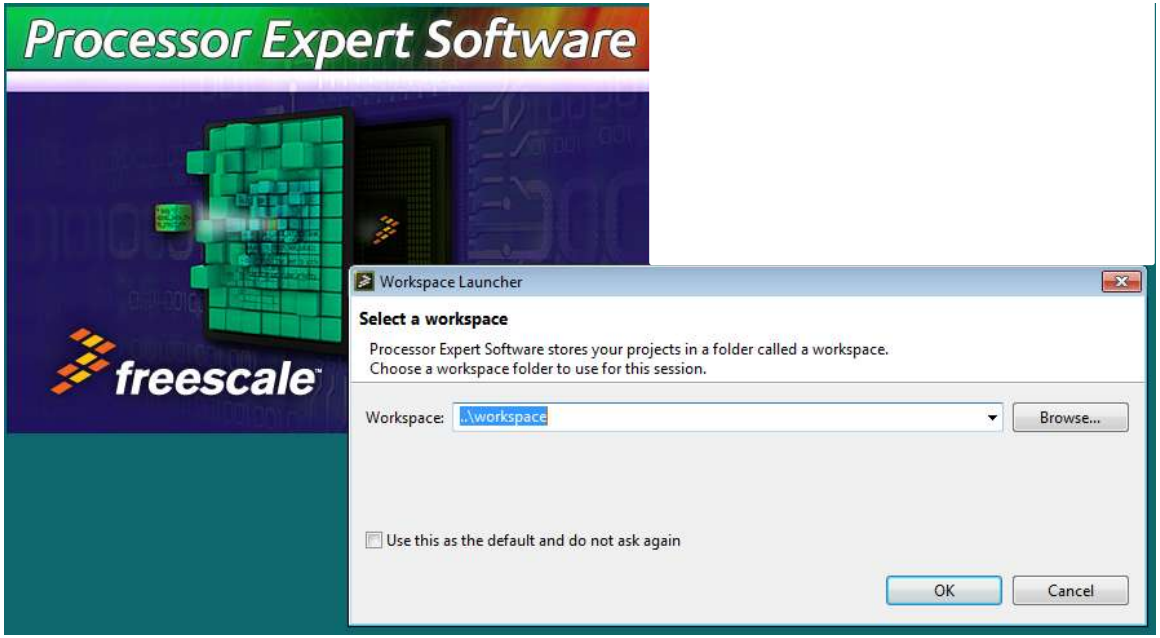
Making an LED Blink Using IAR Workbench

Simple Demo – Blink an LED

- Build a new project for the KL25Z128
  - Using Driver Suite for Microcontrollers on your Laptop
  - Add an interval timer driver
  - Add a GPIO driver for one output bit configured to the GPIO bit for blue LED on Freedom board (PTD1 pin)
  - And we'll do it without writing a line of code.

Open Driver Suite 10.2

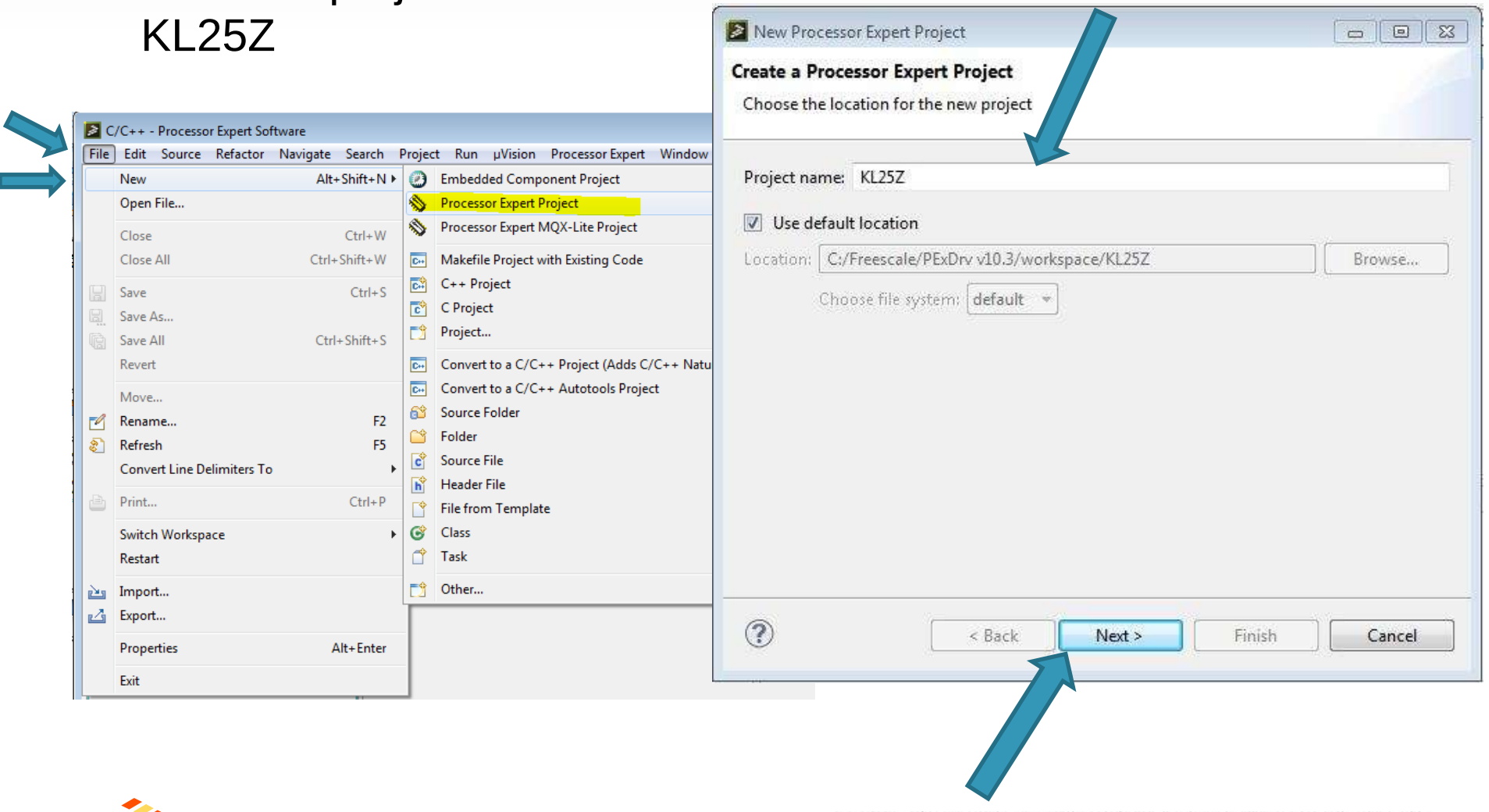
- Open the Processor Expert Driver Suite v10.2 on your laptops



- This gives us a clean workspace to do our labs…
  - Note: workspaces are specific to versions of Eclipse and CodeWarrior.

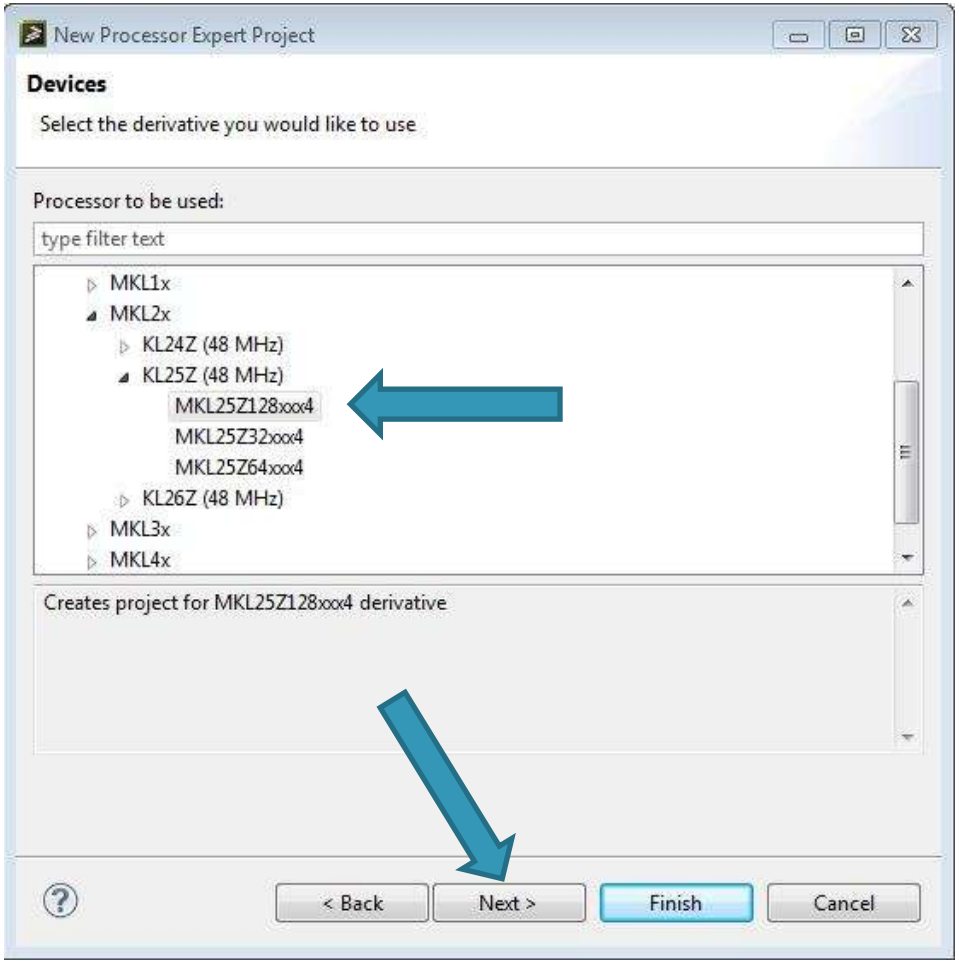
Create a new Project

- Create a project named KL25Z



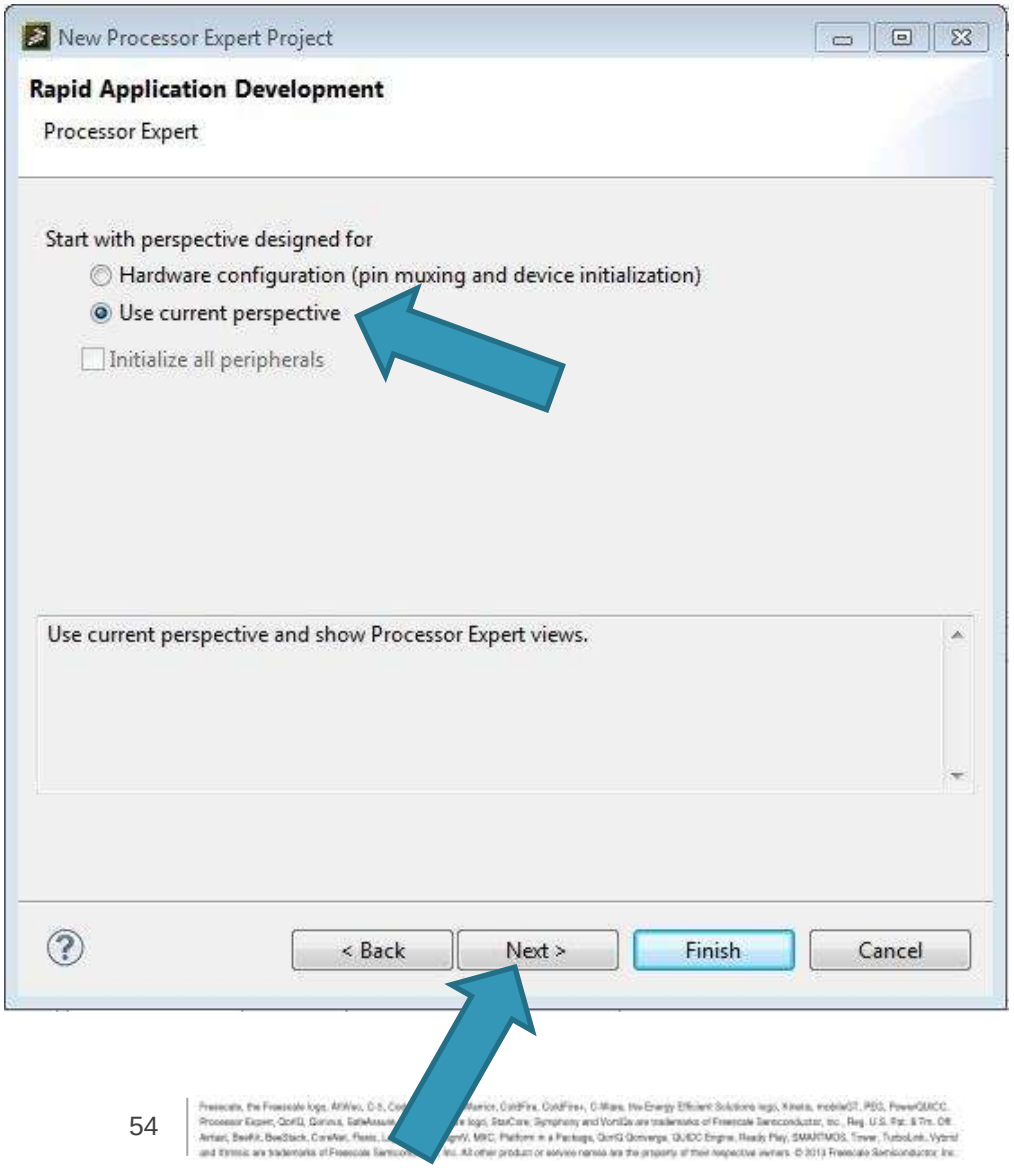
Choose the processor to be used

- Choose the MLK25Z128 part

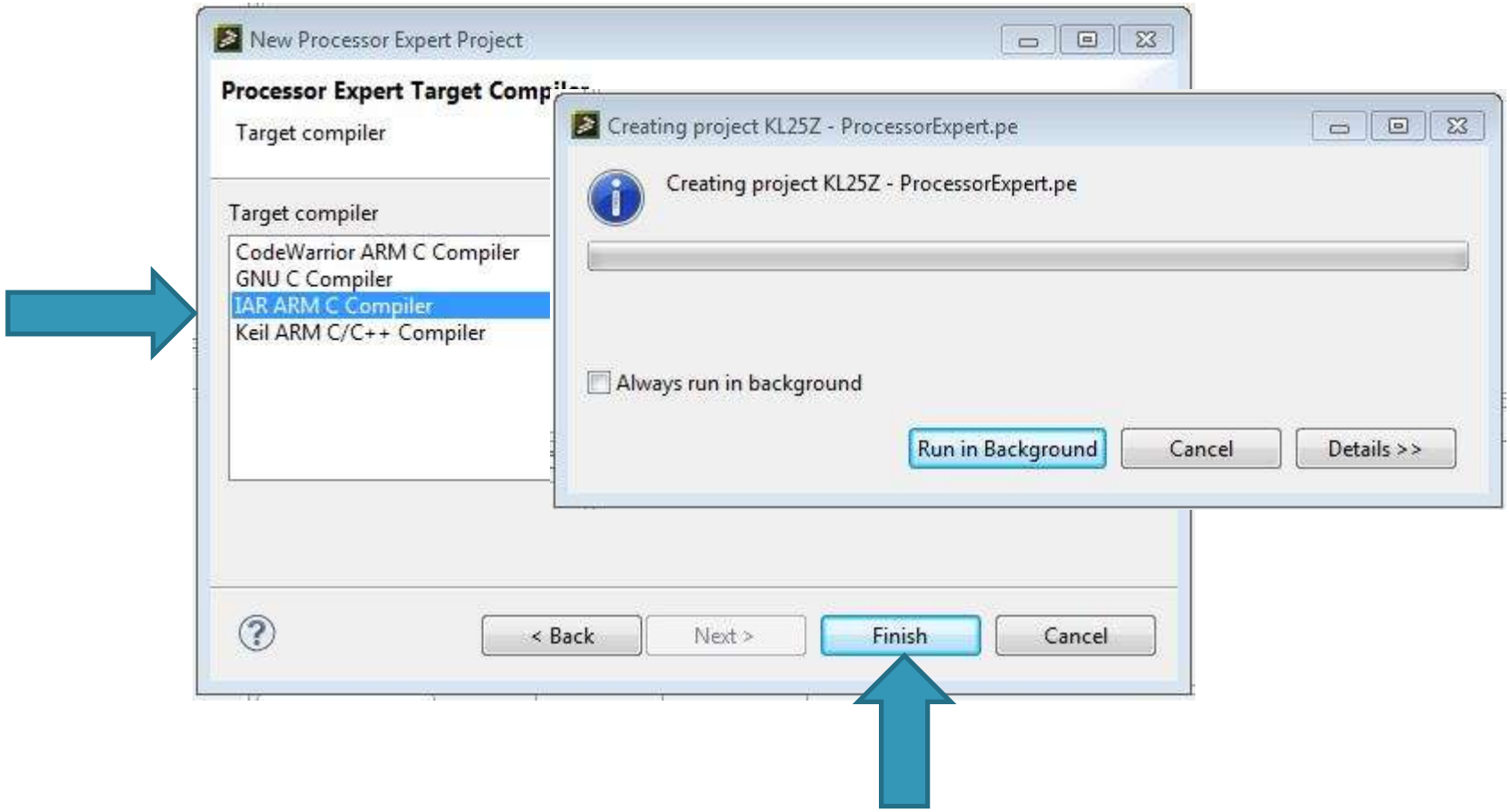


Choose the Perspective

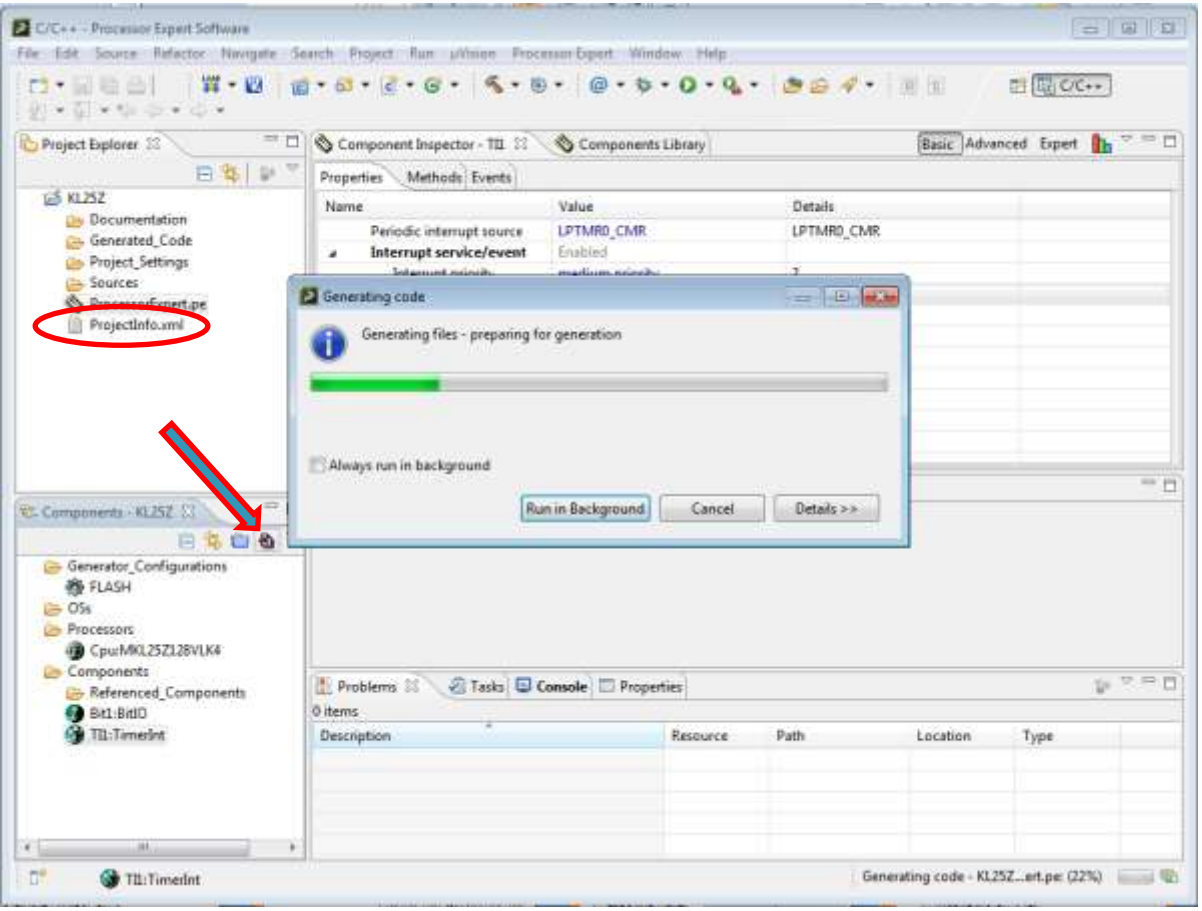
- Hardware Configuration
  - Pin muxing and initialization
- Software Configuration
  - Software project panel
  - Component library



Choose the target compiler



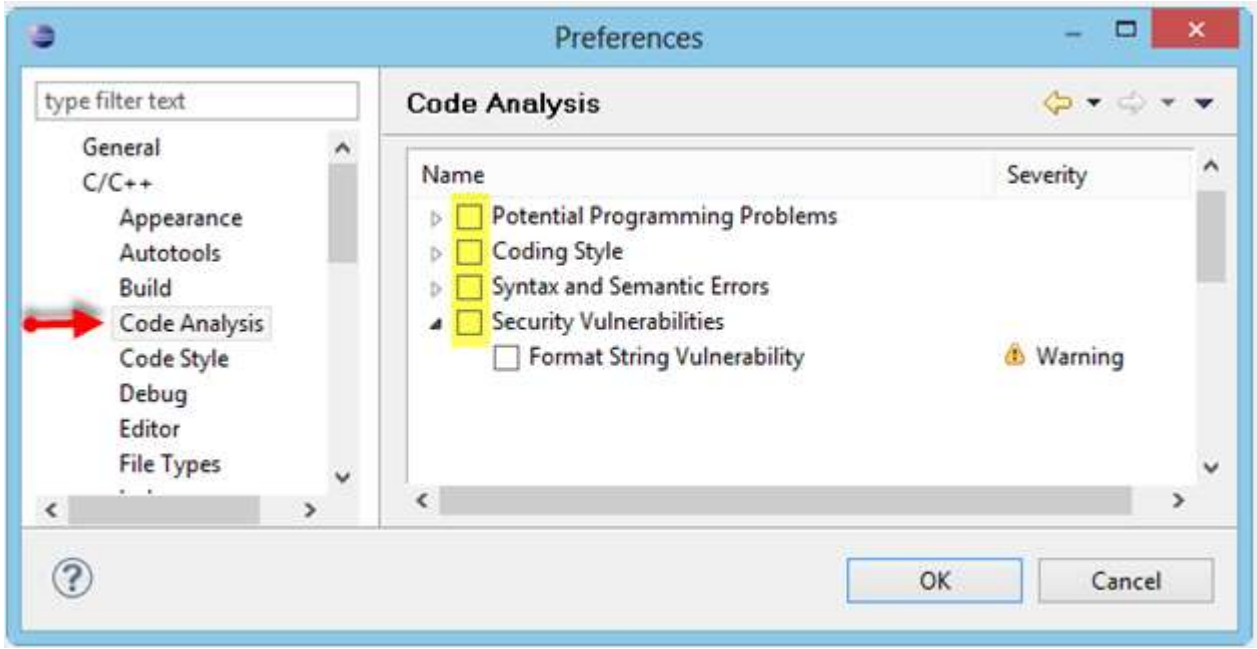
Once you finish adding components…



- After asking Processor Expert to Generate Code, the ProjectInfo.xml file is created.
- This file allows us to connect this project for code generation to an IAR Workbench

ASIDE: Disabling code analysis

- One flaw of Eclipse CDT is that it includes code analysis that is linked to build tools. This is a very cool/useful feature in CodeWarrior but because the Driver Suite does not contain build tools, you should turn it off…

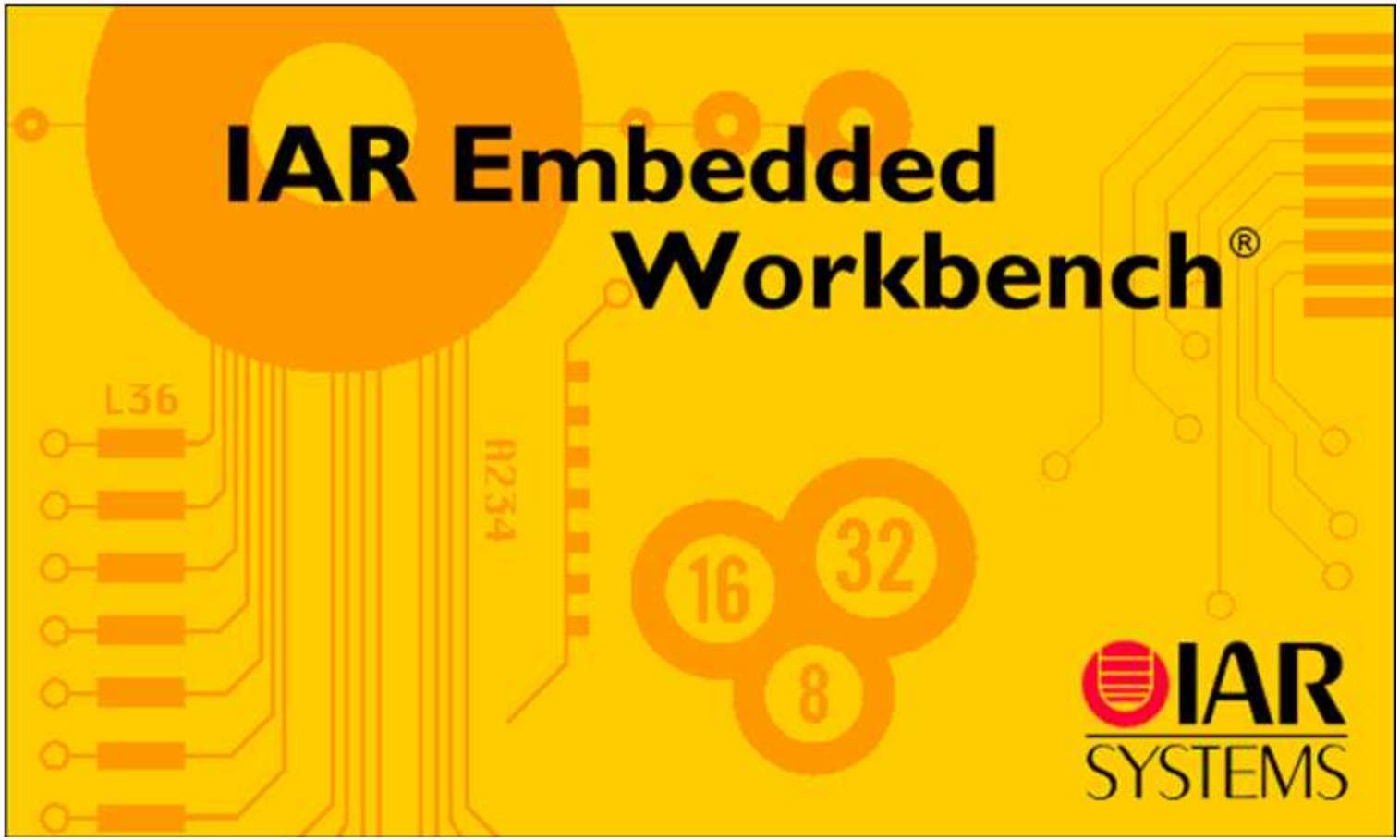


What's next? Now we must flip over to IAR

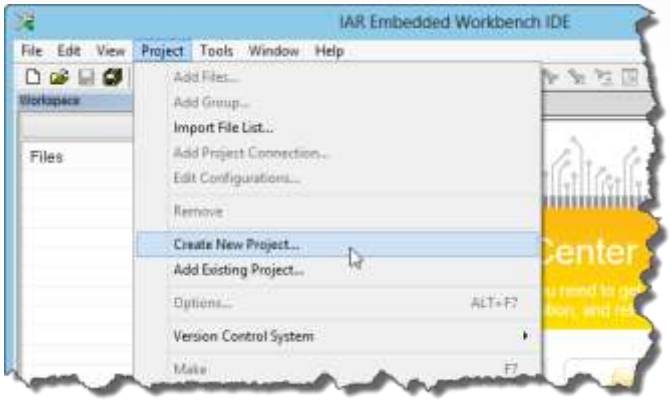
- The difference between CodeWarrior's Processor Expert and the Driver Suite is that CodeWarrior is a complete IDE with build tools (compilers, linkers, etc.) and a full debug environment.

- By contrast, the Driver Suite for Microcontrollers is just a code generation project. It knows nothing about how to compile or debug the source code generated.

Launch IAR Workbench v6.60 or later



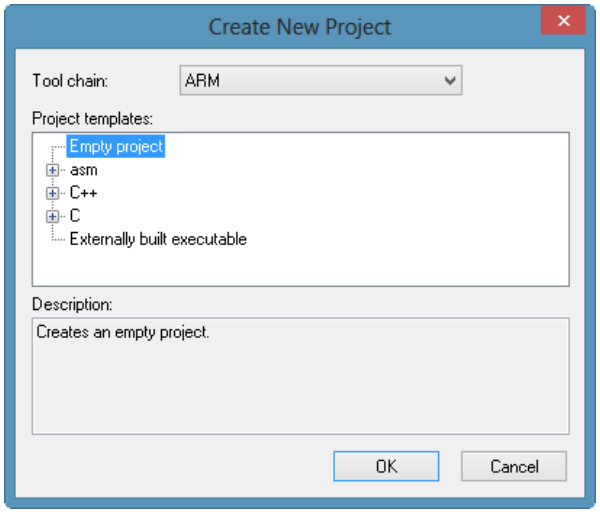
Create a new IAR Project



- Create a new project in IAR Workbench

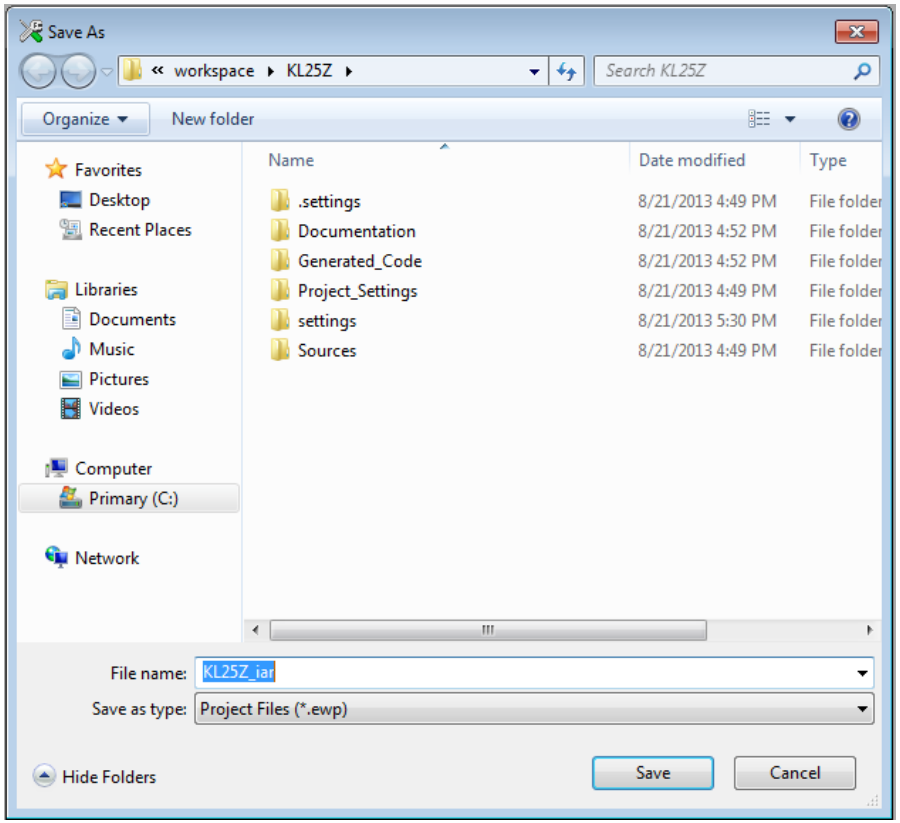
- Choose an Empty project

(anyone know why?)



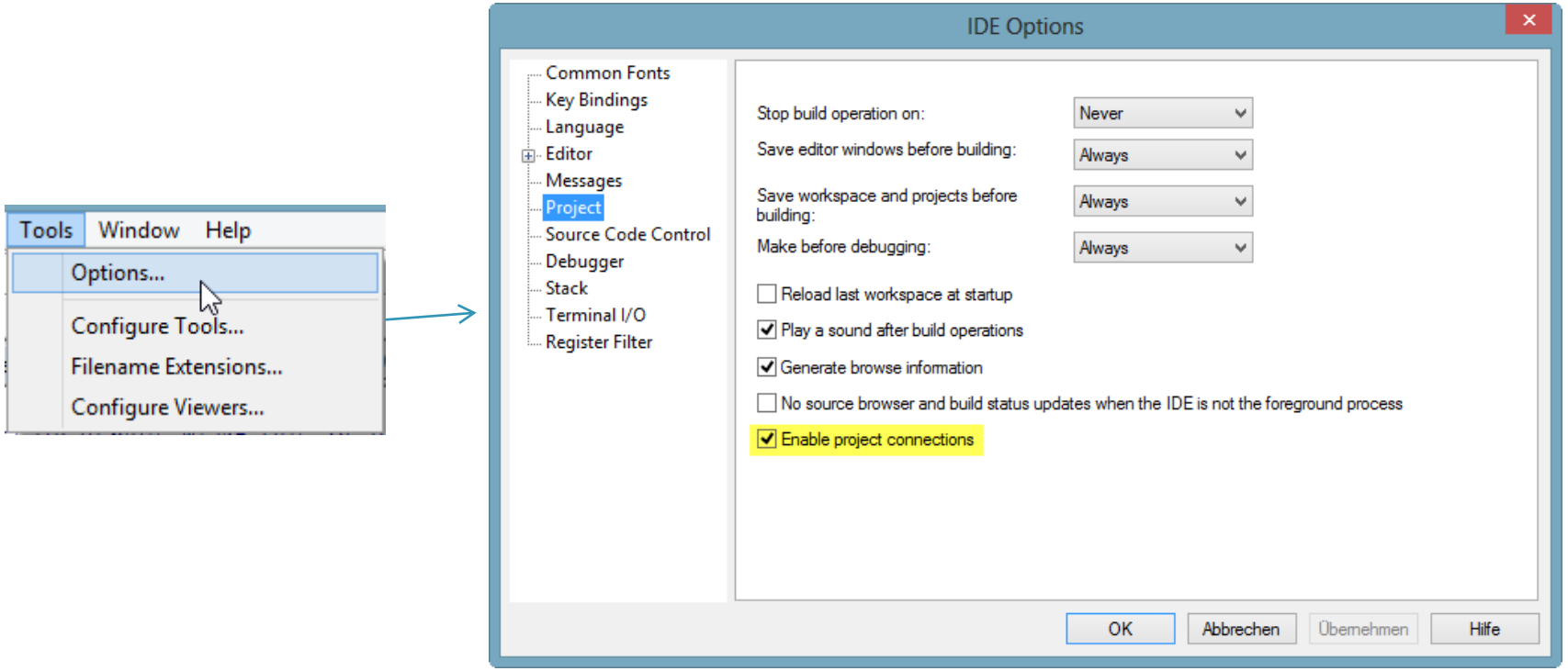
Save the project file (.ewp)

- You can save the project file anywhere
- We recommend saving it into the workspace project for Processor Expert.
- This way, all files are co-located and this makes path setup easier.
- Its also a much easier mechanism of supporting version control.

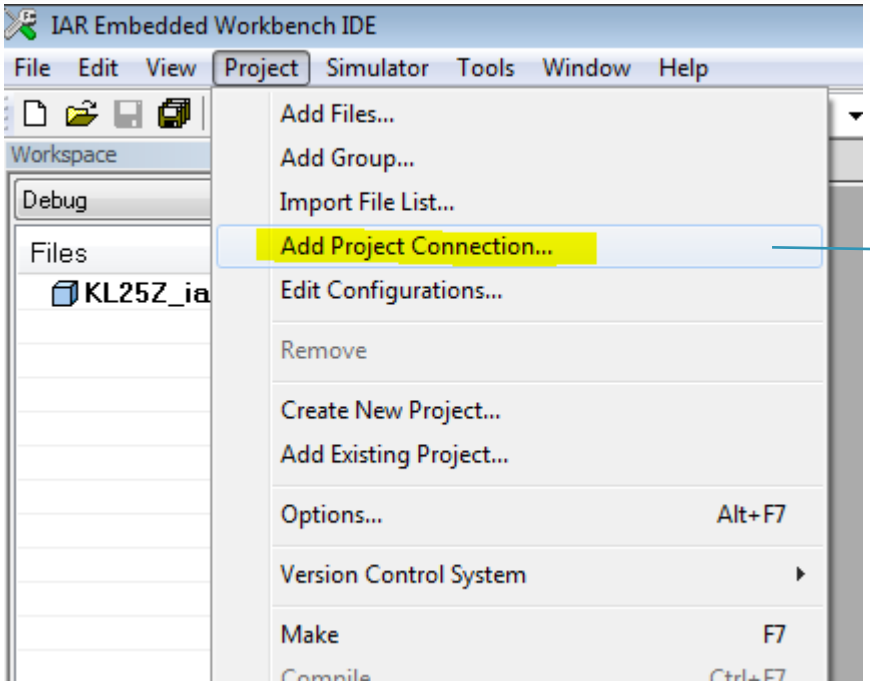


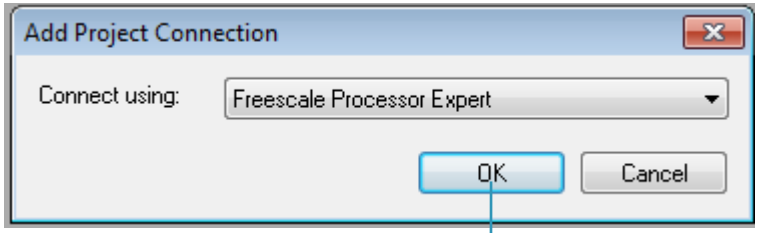
Enabling Project Connections

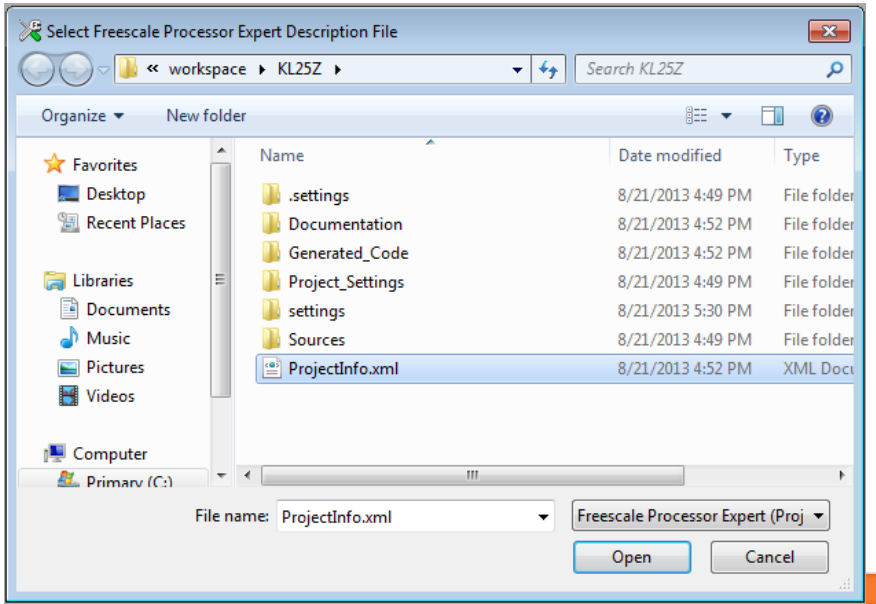
- IAR Workbench v6.5 or later comes with a great mechanism of connecting projects which makes linking Processor Expert into IAR much easier.
- You enable this capability by selecting Tools>Options…



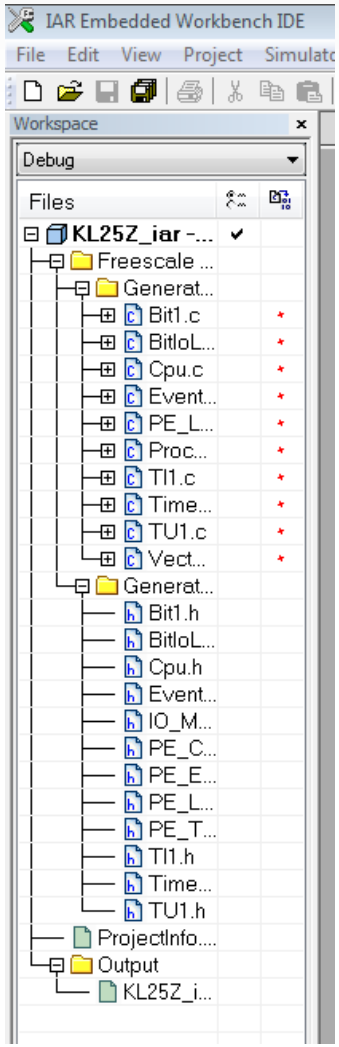
Now connect the Processor Expert project





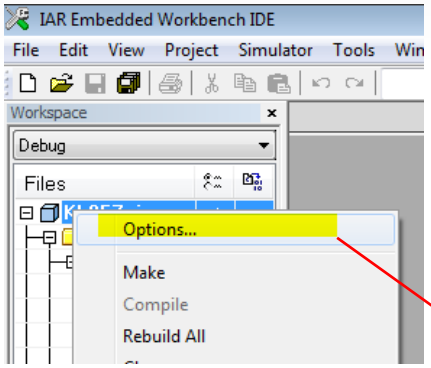


Connected Projects are cool!



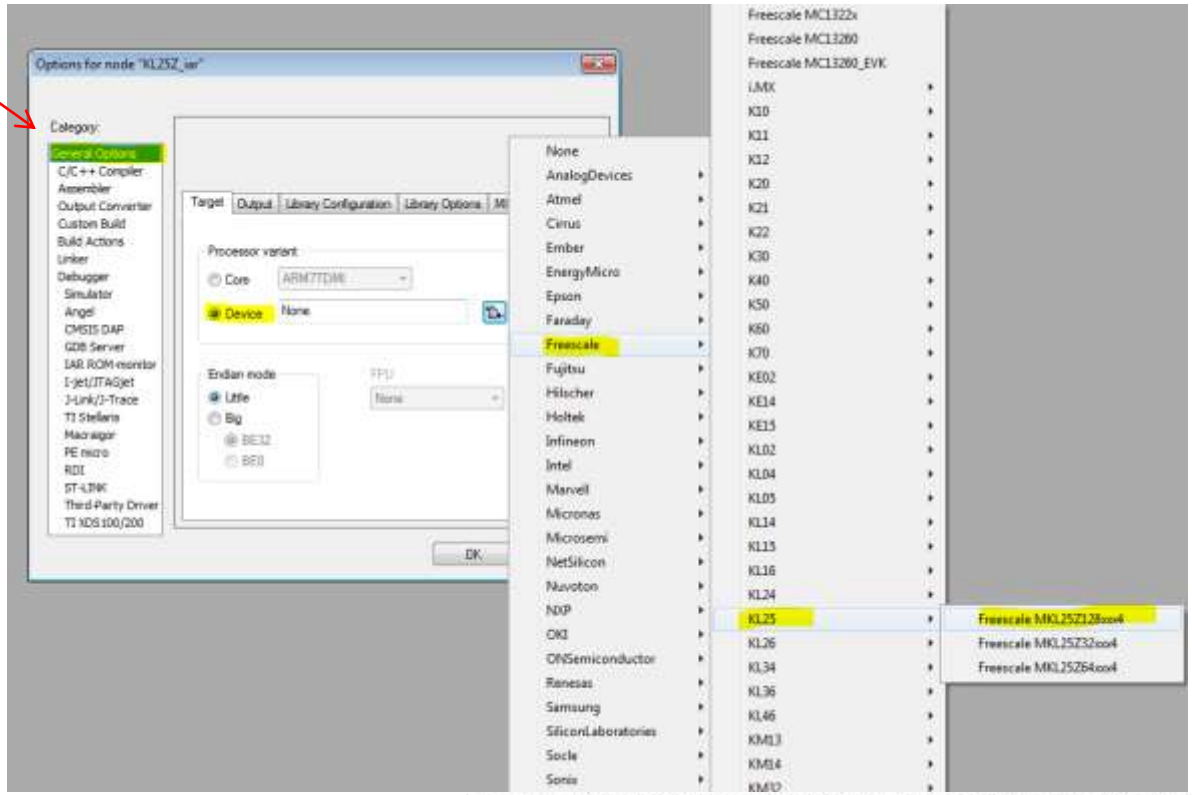
- IAR automatically adds the source and header files generated by Processor Expert.
- IAR will keep these files up-to-date if you make further changes in Processor Expert.
- IAR also sets up all the include paths, etc.
- But you need to still do two steps…

Project Options…



- Must setup

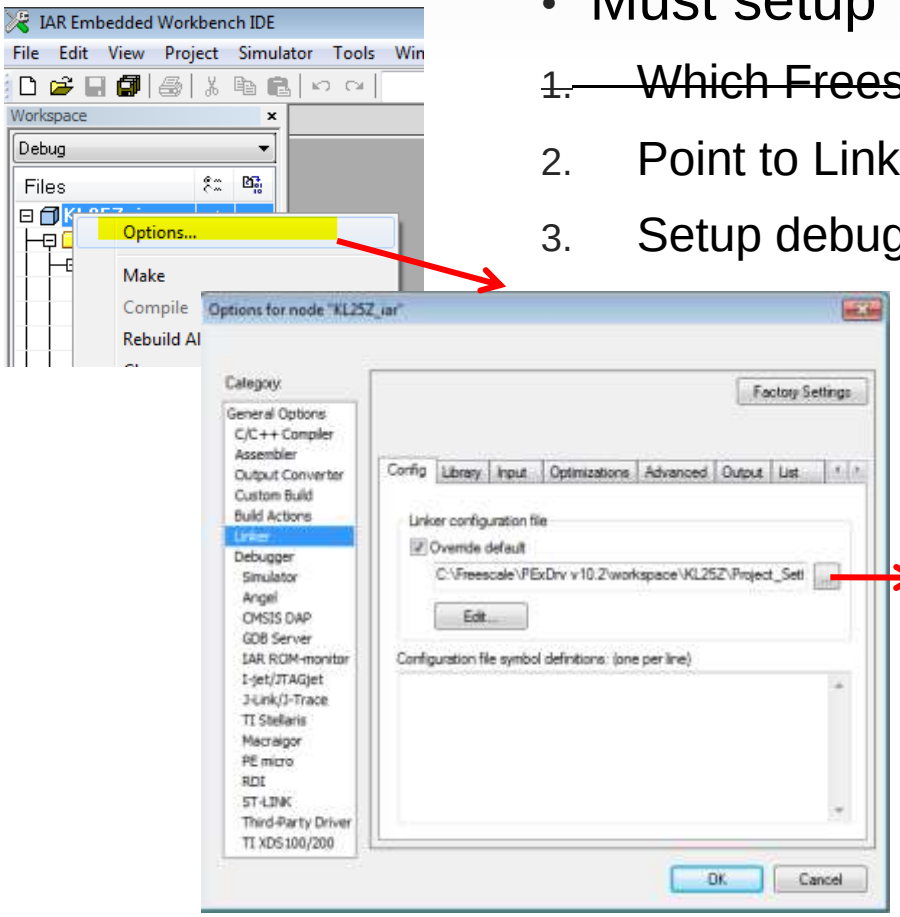
1. Which Freescale Processor is used
2. Point to Linker Configuration file
3. Setup debugger to use PE/OpenSDA

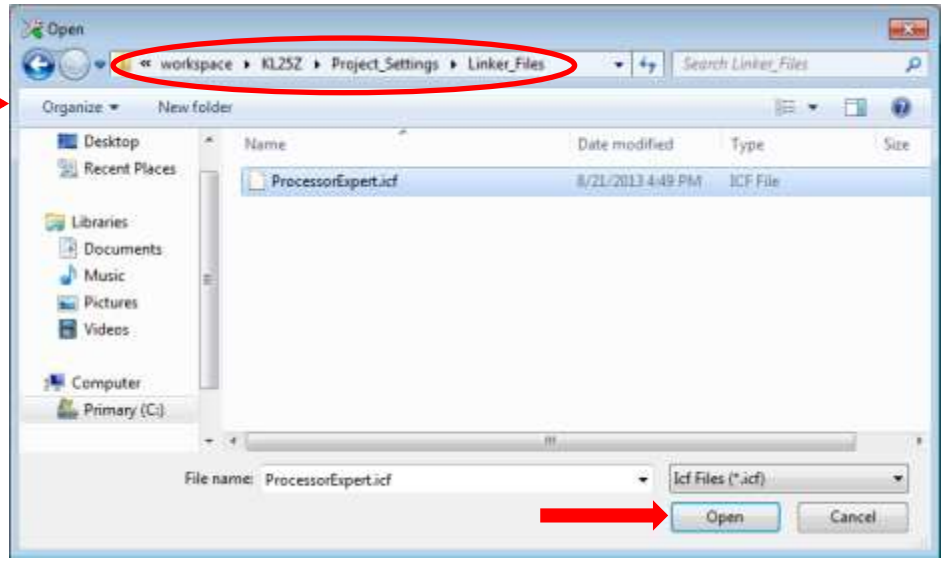


Project Options…

- Must setup

1. ~~Which Freescale Processor is used~~
2. Point to Linker Configuration file
3. Setup debugger to use PE/OpenSDA

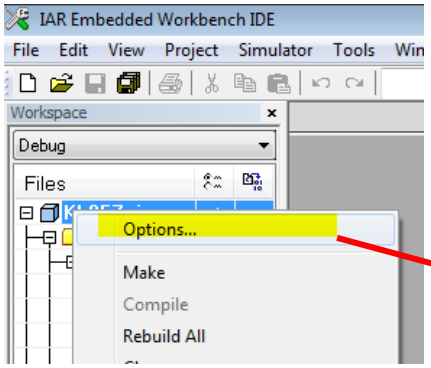


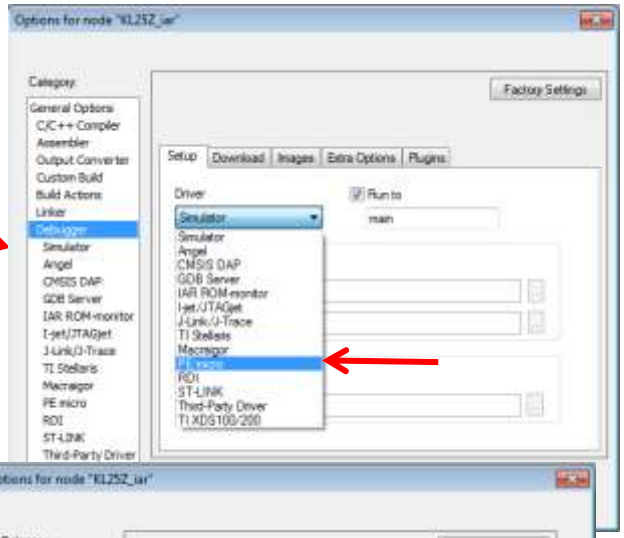


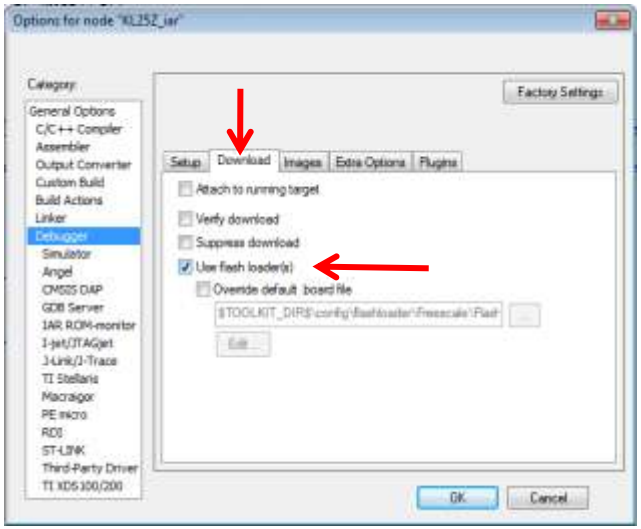
Project Options…

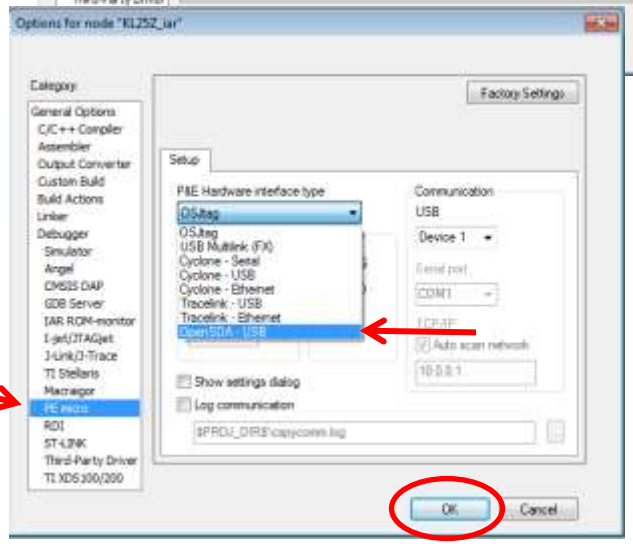
- Must setup

1. ~~Which Freescale Processor is used~~
2. ~~Point to Linker Configuration file~~
3. Setup debugger to use PE/OpenSDA



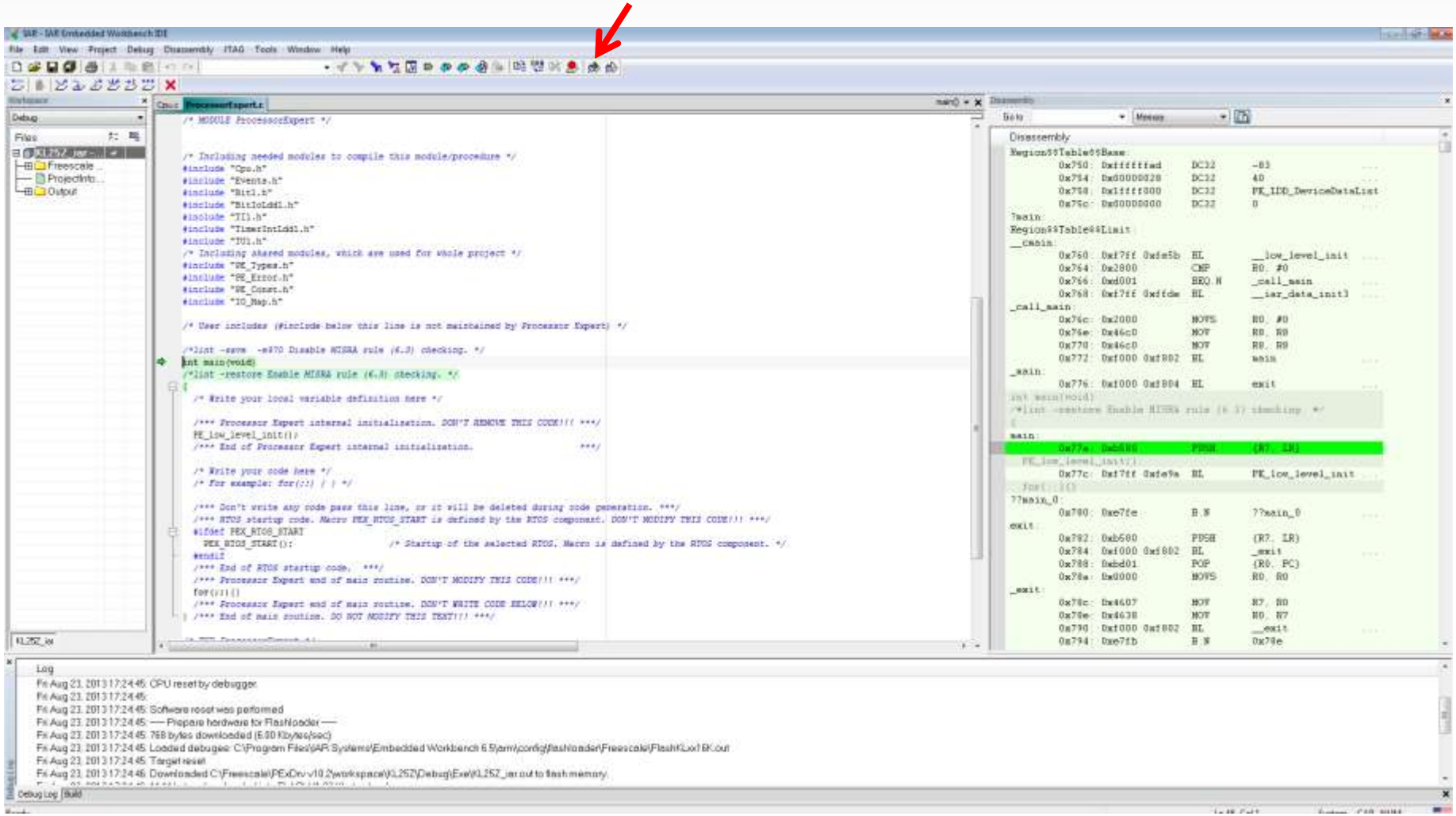






That's it!

Debug the project



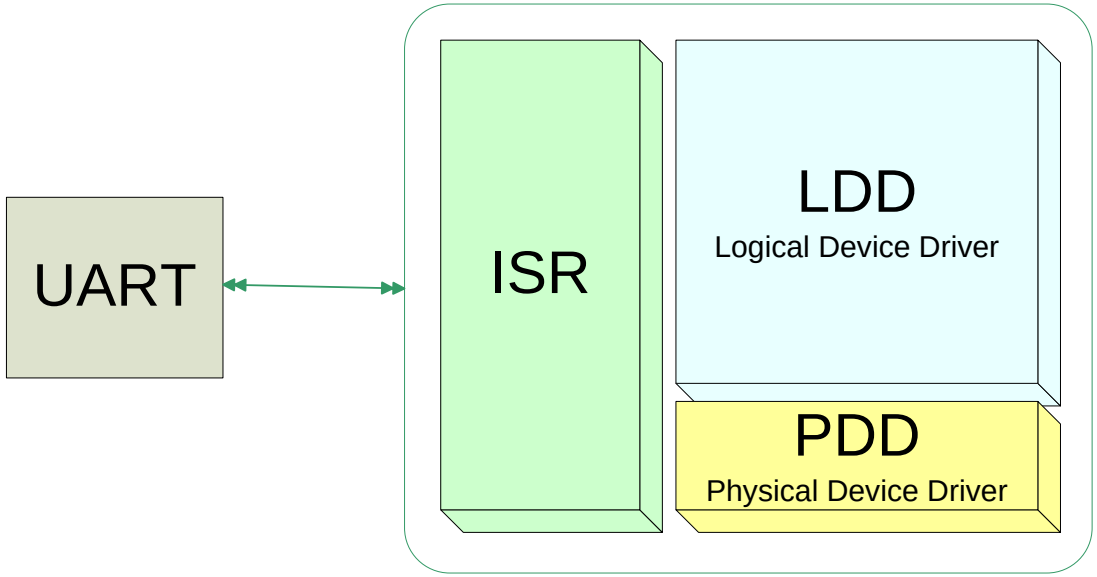
freescale™

High Level and Logical Device Driver Components

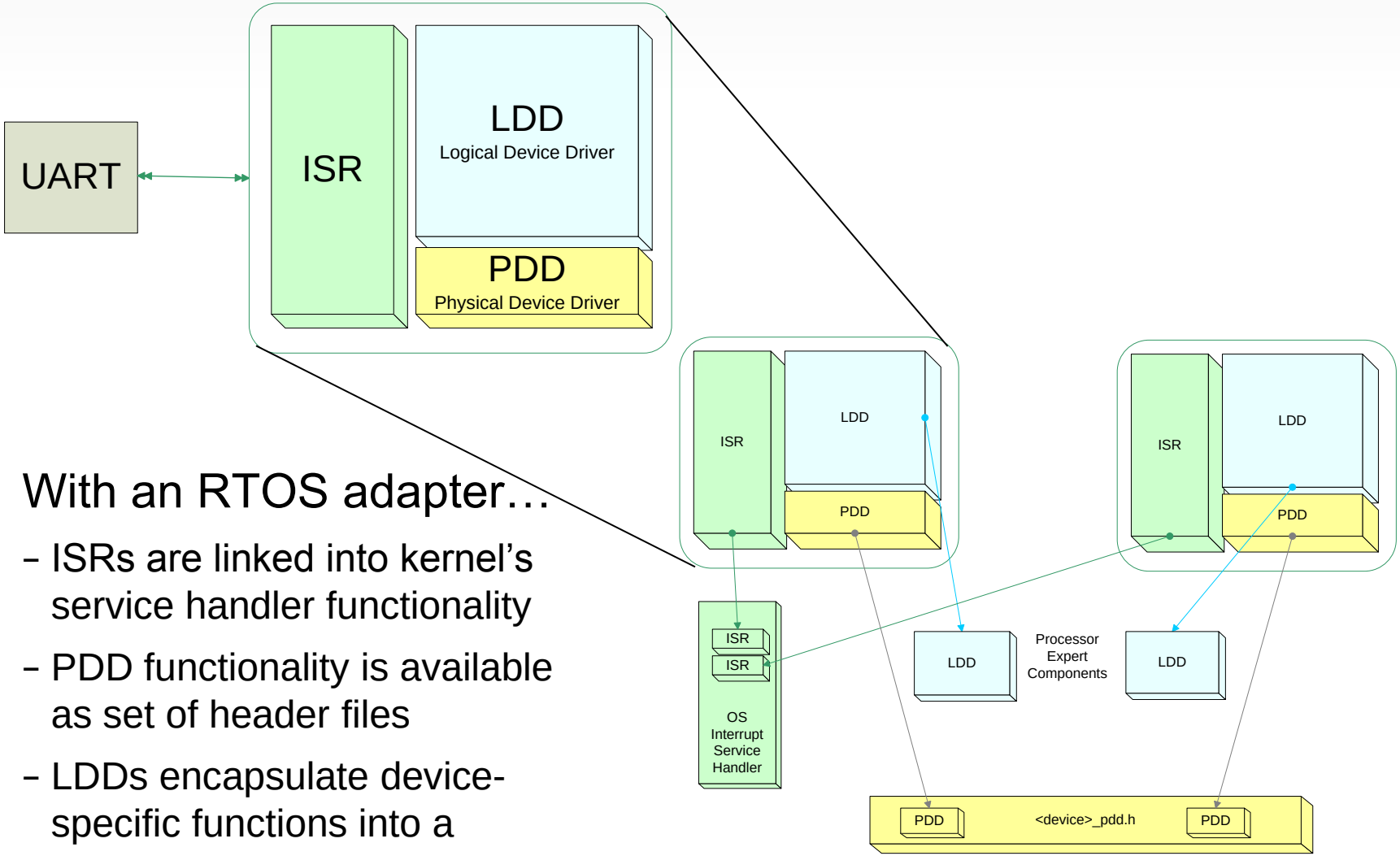
Revisited

Peripheral Driver Anatomy

- Peripheral drivers…
  - Always include Physical Device Driver (PDD) layer. PDD uses base address and control registers to interface with the device.
  - May include Logical Device Driver (LDD) component. LDD handles buffering and state machine type logic.
  - May include interrupt or deferred service (ISR) routine. ISR provides callback functionality.



Peripheral Driver Anatomy



- With an RTOS adapter…
  - ISRs are linked into kernel's service handler functionality
  - PDD functionality is available as set of header files
  - LDDs encapsulate device-specific functions into a system API

LDD Components

- Every component has Init() method
  - Only one parameter - UserDataPtr.
  - Initializes appropriate peripheral and driver.
- Every component has Deinit() method
  - De-initializes appropriate peripheral and driver.
- First parameter for every method is pointer to device structure returned from Init() method.
- Code generation is based on information in RTOS adapter
- Events can be enabled/disabled at runtime.
- Components can be disabled in Low Power Modes.
- CPU component
  - Does not automatically initialize components by default.
  - Auto-initialization can be enabled/disabled.

High Level Components

- Designed to provide standard API to functionality
- Component categories include:
  - Port I/O (i.e. BitIO, BitsIO, ByteIO)
  - Timers (i.e. TimerInt, PWM, Capture)
  - Communication (i.e. AsynchroSerial, SynchroMaster, SynchroSlave, I2C)
  - ADC
  - Internal memories
- Eases porting to another microcontroller supported by Processor Expert software
- Sets functionality based on application needs and does not require knowledge of hardware registers
- MCU specific features are supported as CPU specific settings or methods and are not portable

Migrate HLB to LDD

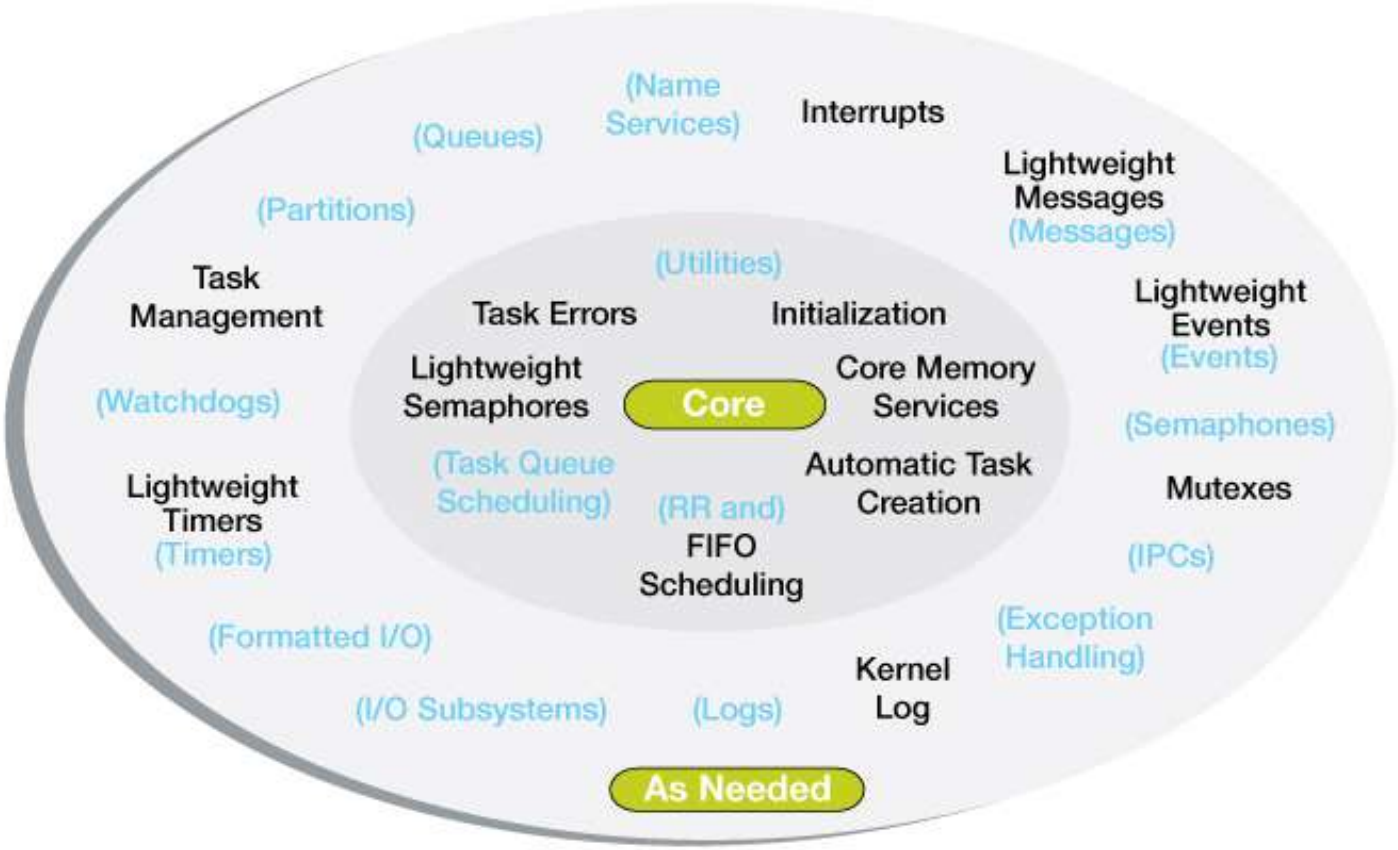
- Developed high level software components, which inherit from PDD or LDD, to migrate S08 projects to Kinetis L Series devices
  - Provides same application program interface (API)
  - Software wrapper at the level required to provide the necessary functionality
- High level components available for Kinetis
  - AsynchroSerial
  - SynchroMaster
  - SynchroSlave
  - ADC
  - ExtInt
  - FreescaleAnalogComp
  - IntFlash
  - BitIO
  - BitsIO
  - PWM
  - TimerInt

MQX-Lite Overview

MQX Lite – Overview

- **Very light MQX kernel for resource-limited MCUs**
  - Targeted at the Kinetis L family initially
  - Packaged as a Processor Expert component
- **I/O capability provided by Processor Expert**
  - USB via FSL bare-metal stack, also a Processor Expert component
  - No POSIX-like drivers or file access
- **Programming model allows upward code migration**
  - It is a true subset of the full MQX RTOS
  - Code built with MQX Lite will move to full MQX RTOS easily

MQX Subsystems Removed in MQX Lite RTOS



MQX Lite – Main Features

- **Scheduler**
  - Priority pre-emptive schedule
  - Support for lightweight semaphore, and mutex (with polling)

- **Task Management does not support dynamic task creation**
  - All task resources allocated at compile time
  - Dynamic memory management not allowed

- **Lightweight events and messaging only**

- **Lightweight timer included (one shot, and periodic notification)**

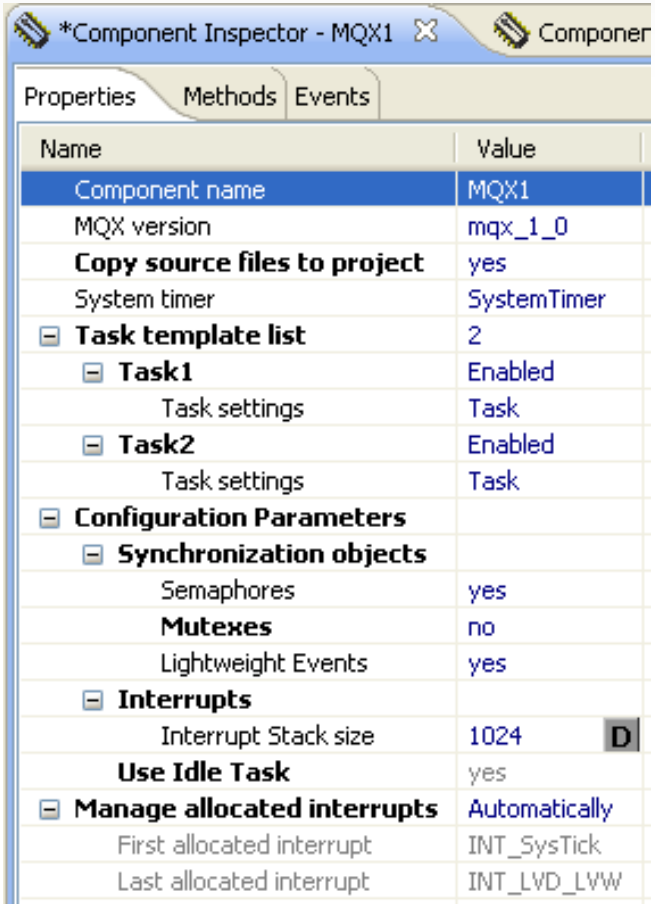
How Small Is MQX Lite?

- **Minimal App – Hello Task, Idle task, interrupt stack**
  - Code = 10.4K
  - Data = 3.7K (including 1.5K for stacks)

- **Typical App – 7 tasks + idle, lightweight events, queues**
  - Code = 27K
  - Data = 10K (5K for stacks)

MQX Lite and Processor Expert Integration

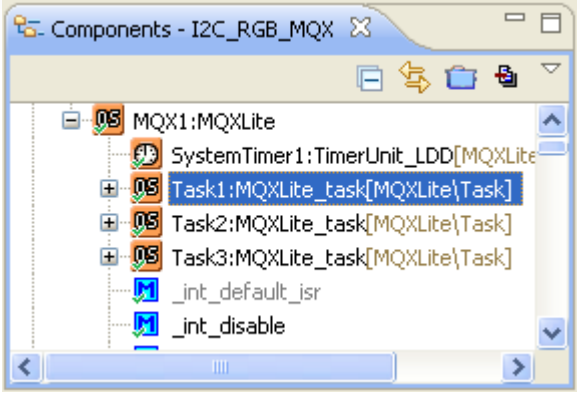
- **MQX Lite is an RTOS adapter**
  - Interrupt mechanism in MQX is unchanged
  - Processor Expert LDDs work with the RTOS
- **All I/O provided by LDD components**
- **Set up number of tasks in the task template list**
- **Easy to add MQX Lite to existing app**
  - Just drop in the MQX Lite component

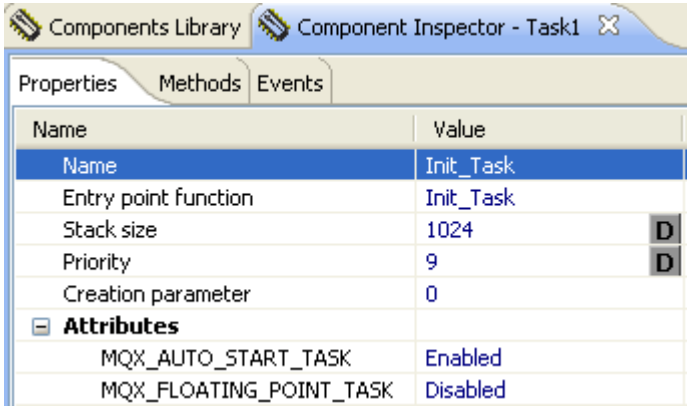


Configuring a Task

- Tasks are separate components contained within the RTOS component

- Configure in the component inspector
  - Set name of task function
  - Priority
  - Stack size
  - All the parameters of an MQX task template list





Generated Code for Tasks

- **In file mqx_tasks.c (this is new to Processor Expert)**
  - Creates a function stub, and a comment in the middle

```
void Task1_task(uint32_t task_init_data)
{
  int counter = 0;
  while(1) {
    counter++;
    /* Write your code here ... */
  }
}
```

- **There will be a function stub per task, as you name them**

Generated Code for Interrupts

- **Interrupts are handled in the driver, but call out to “event” handlers**

- **Event handlers are in events.c**
  - Function stub for each handler, you create the functionality
  - With a comment, **`/* Write your code here ... */`**

- **For example…**

```
void BlinkRateCounter_OnCounterRestart(LDD_TUserData *UserDataPtr)
{
  BlinkFlag ^= 1;
  if (BlinkFlag) {
    PWMTimerB_Disable(PWMTimerB_DeviceData);  //Disable LEDs for a blink period.
    PWMTimerRG_Disable(PWMTimerRG_DeviceData);
  }
}
```

Generated Code for main()

- **The main() function is in processorexpert.c**
  - Same as always for Processor Expert, no change here

- **It initializes Processor Expert, and passes control to the RTOS**
  - You can do your own initialization in between

```
PE_low_level_init();

/* Write your code here */

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

#ifdef PEX_RTOS_START
  PEX_RTOS_START(); /* Macro defined by RTOS component. */
#endif
```

LAB 4: Create New Project

Using Processor Expert & MQX-Lite

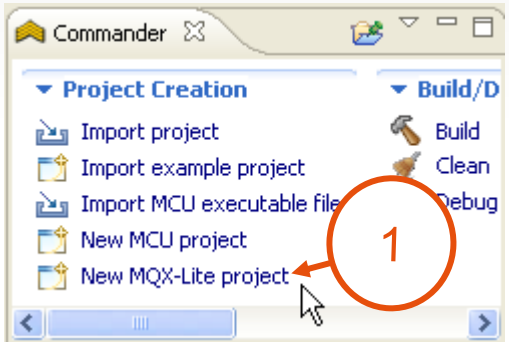
LAB 4 – Create An MQX Lite Project

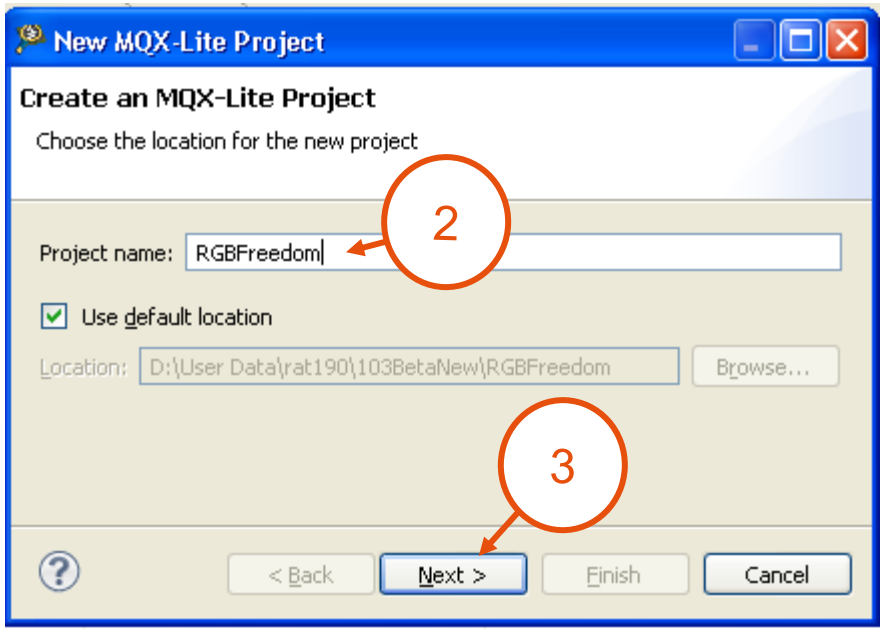
- This hands-on lab shows how to…
  - ☐Create an MQX Lite Project and configure the CPU (48 MHz)
  - ☐Configure the RTOS timer and add tasks
  - ☐Configure the tasks: Init_Task, ColorTask
  - ☐Add and configure components
  - ☐Generate code and walk through the code model
  - ☐Add files and walk through the task code
  - ☐Build, debug, play with the pretty colors

- We'll use the KL25Z Freedom board

Create a New MQX Lite Project

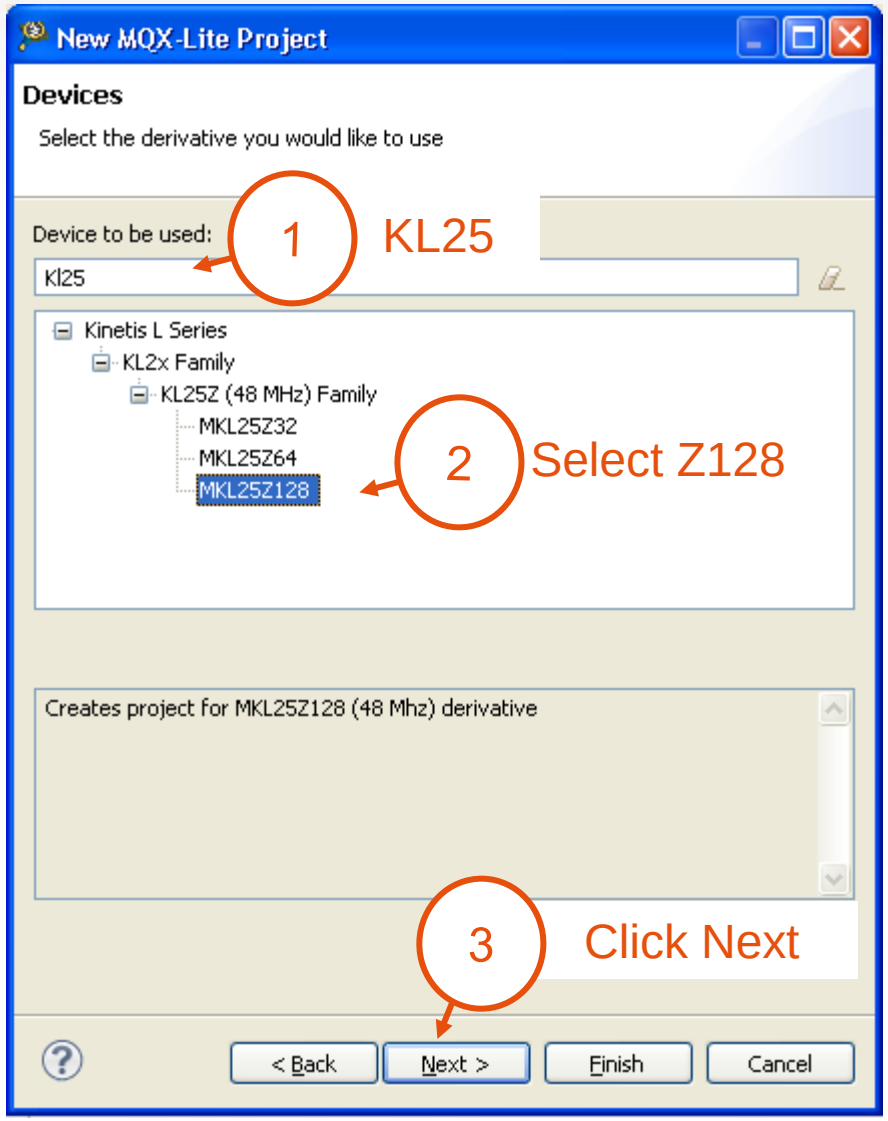
- Click New MQX-Lite project in Commander View
- Name it – RGBFreedom
- Click Next





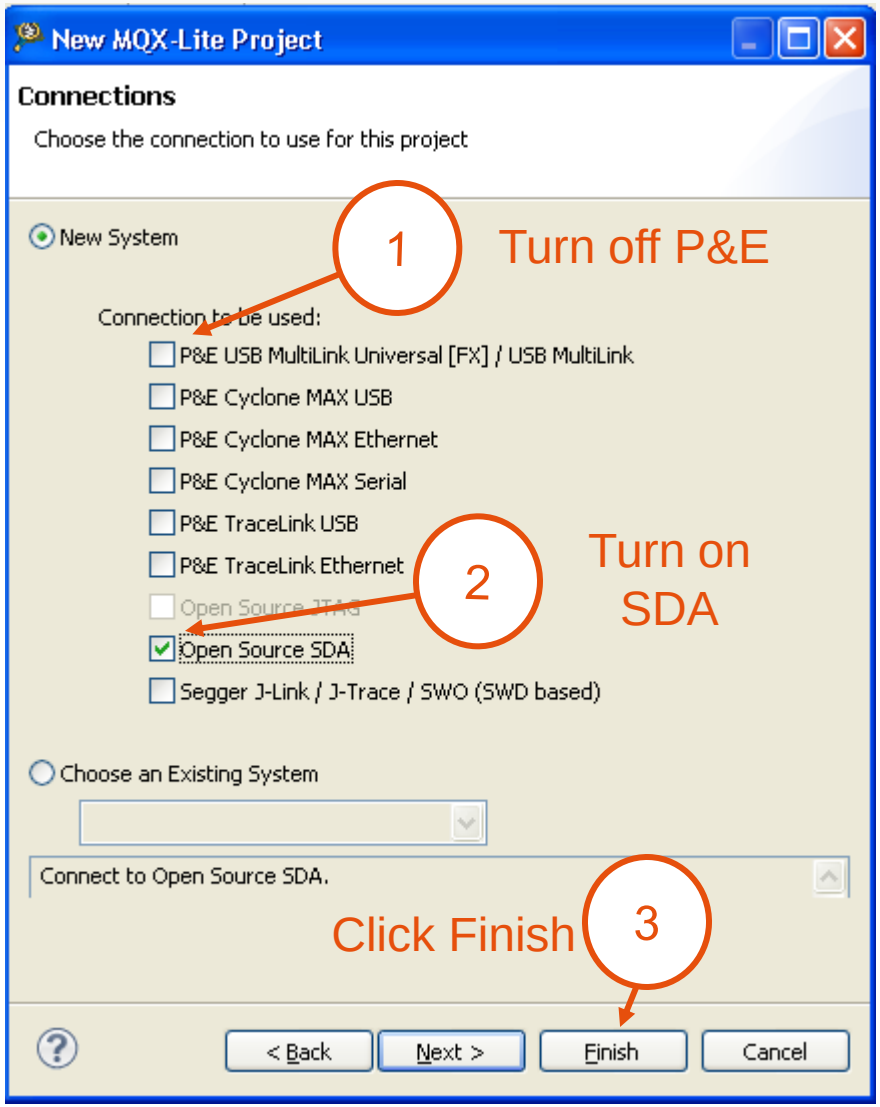
Pick the Device

- Type in the filter
- Select Z128
  - That's what's on the Freedom board
- Click Next



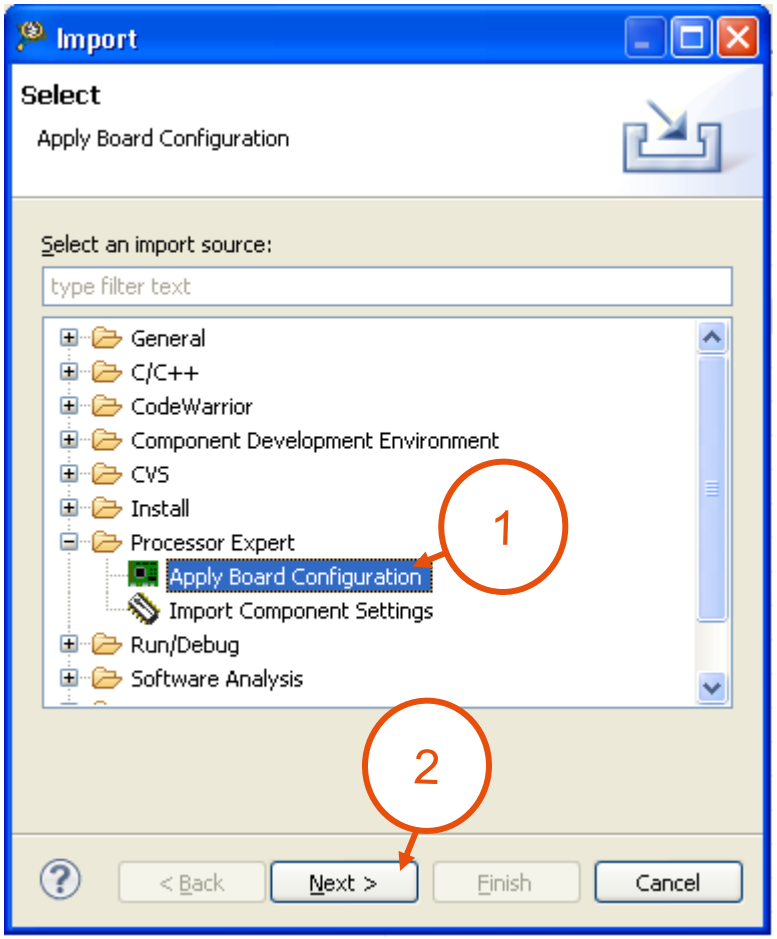
Set the Debug Connection

- Turn off P&E
- Turn on Open SDA
- Click Finish
- We accept default values from here on out
- The project appears



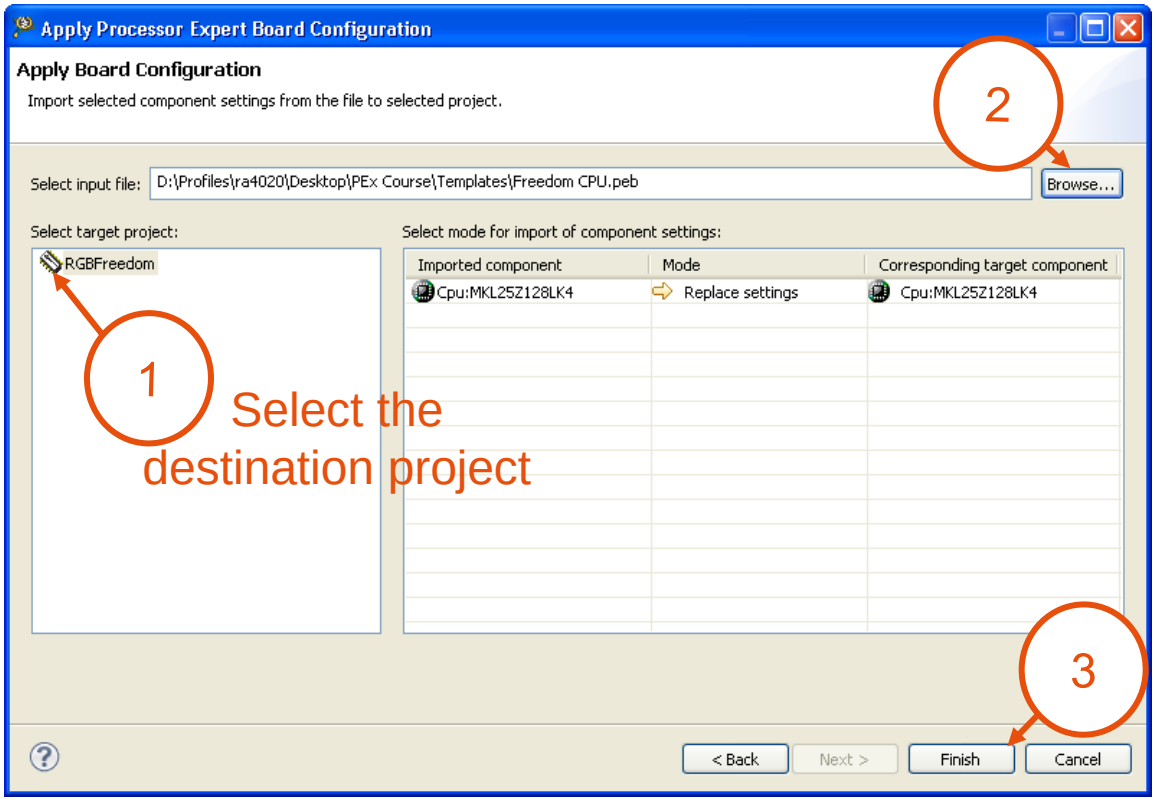
Configure the CPU – the Easy Way

- By default, clocks are not set for a Freedom board
- Select File→Import
- Select Processor Expert→ Apply Board Configuration
- Click Next



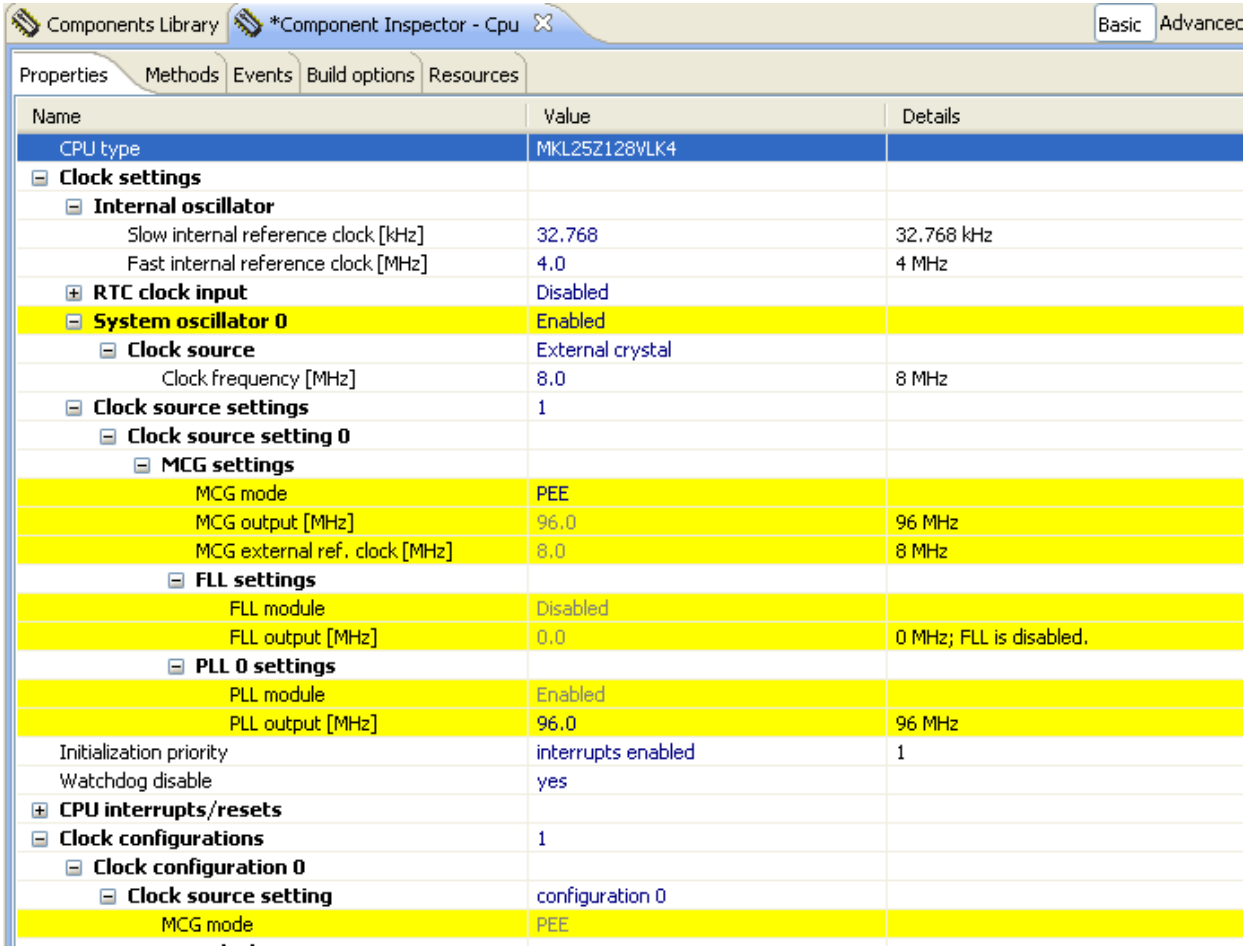
Open a Board Configuration File

- Select the project to which to apply the component
- Navigate to ..\PEx Course\Templates folder on Desktop
- Look for Freedom CPU.peb



CPU is Now Properly Configured

- Scroll to see what was set – all the yellow items changed



Name	Value	Details
CPU type	MKL25Z128VLK4	
Clock settings		
Internal oscillator		
Slow internal reference clock [kHz]	32.768	32.768 kHz
Fast internal reference clock [MHz]	4.0	4 MHz
RTC clock input	Disabled	
System oscillator 0	Enabled	
Clock source	External crystal	
Clock frequency [MHz]	8.0	8 MHz
Clock source settings	1	
Clock source setting 0		
MCG settings		
MCG mode	PEE	
MCG output [MHz]	96.0	96 MHz
MCG external ref. clock [MHz]	8.0	8 MHz
FLL settings		
FLL module	Disabled	
FLL output [MHz]	0.0	0 MHz; FLL is disabled.
PLL 0 settings		
PLL module	Enabled	
PLL output [MHz]	96.0	96 MHz
Initialization priority	interrupts enabled	1
Watchdog disable	yes	
CPU interrupts/resets		
Clock configurations	1	
Clock configuration 0		
Clock source setting	configuration 0	
MCG mode	PEE	

LAB 4 – Create An MQX Lite Project

- This hands-on lab shows how to…

☑ Create an MQX Lite Project and configure the CPU (48 MHz)

☐ Configure the RTOS timer and add tasks

☐ Configure the tasks: Init_Task, ColorTask

☐ Add and configure components

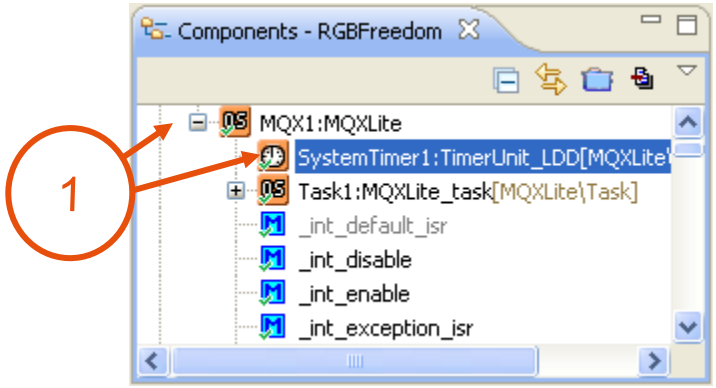
☐ Generate code and walk through the code model

☐ Add files and walk through the task code

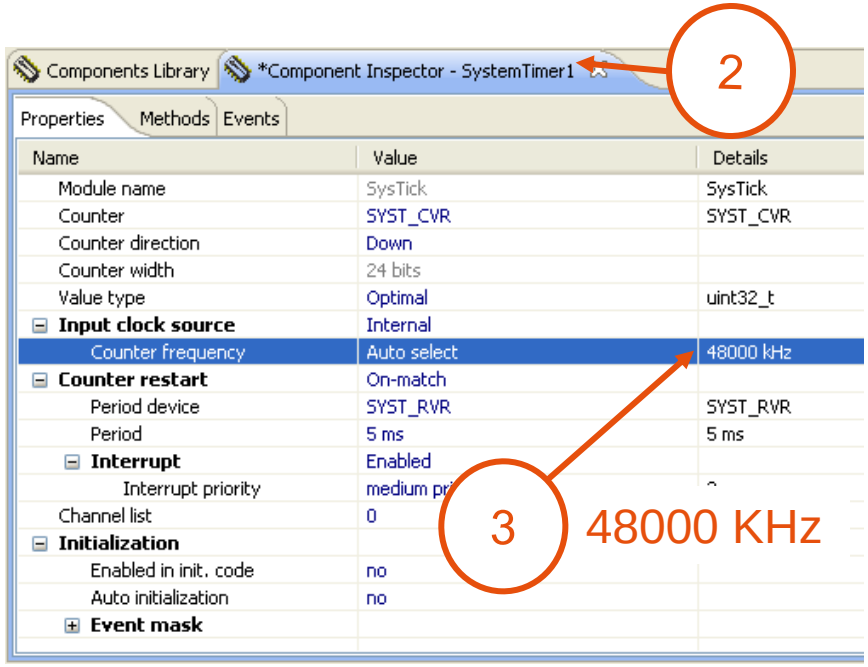
☐ Build, debug, play with the pretty colors

Confirm the RTOS Timer Configuration

- The Freedom board core clock runs at 48 MHz
- Expand the MQX component to expose the Timer, and select it

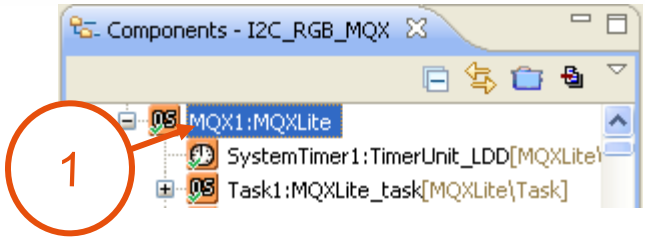


- Click Component Inspector
- Confirm the counter frequency is set to 48000 KHz

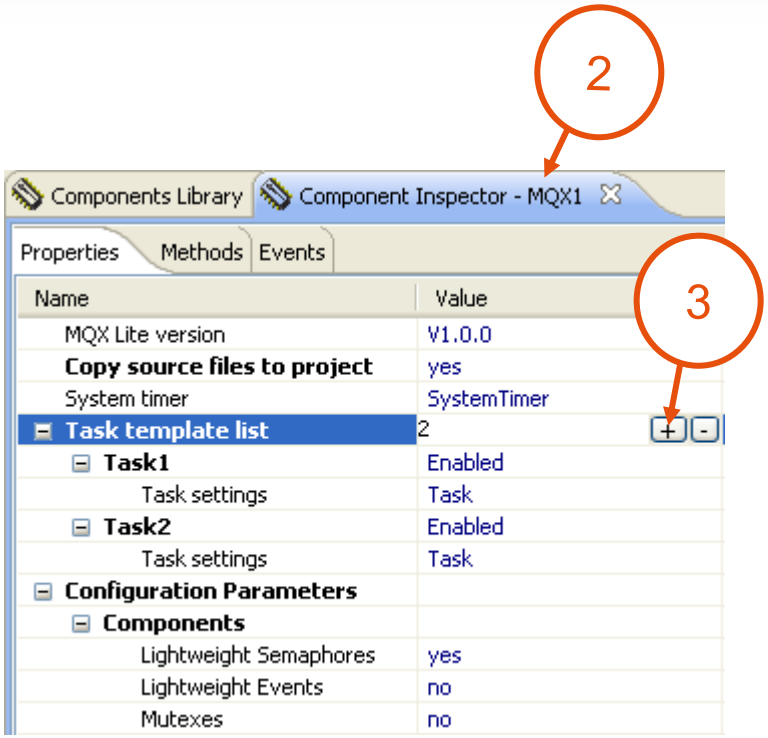


Add a Task to the Application

- In the Components view, select the MQX Lite component

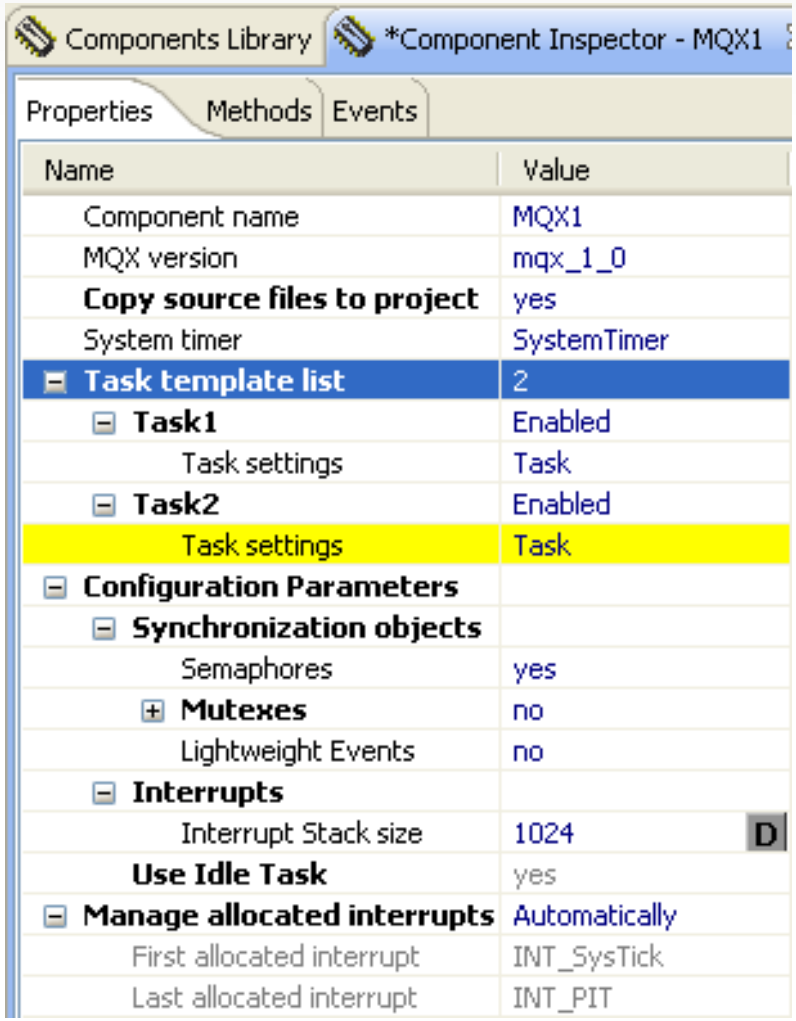


- Select Component Inspector
- Click value for Task Template List
  - A Plus-Minus tool appears
  - Click the Plus sign to increment to 2



Let's Look at the RTOS Options

- Copies source files to project
- Points to the name of the SystemTimer component
- There is one task minimum
- You can turn on/off events, mutexes, semaphores
- You can modify interrupt stack size to save RAM



MQX Lite Methods

- You control what methods are generated
- If you know a method will not be used, disable code generation to save memory
  - Don't rely on linker dead stripping

LAB 4 – Create An MQX Lite Project

- This hands-on lab shows how to…

☑Create an MQX Lite Project and configure the CPU (48 MHz)

☑Configure the RTOS timer and add tasks

☐Configure the tasks: Init_Task, ColorTask

☐Add and configure components

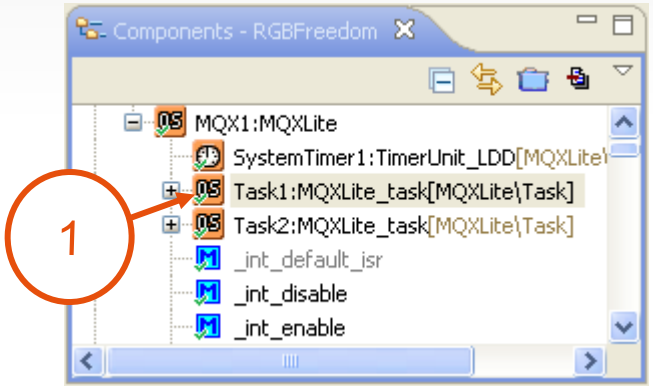
☐Generate code and walk through the code model

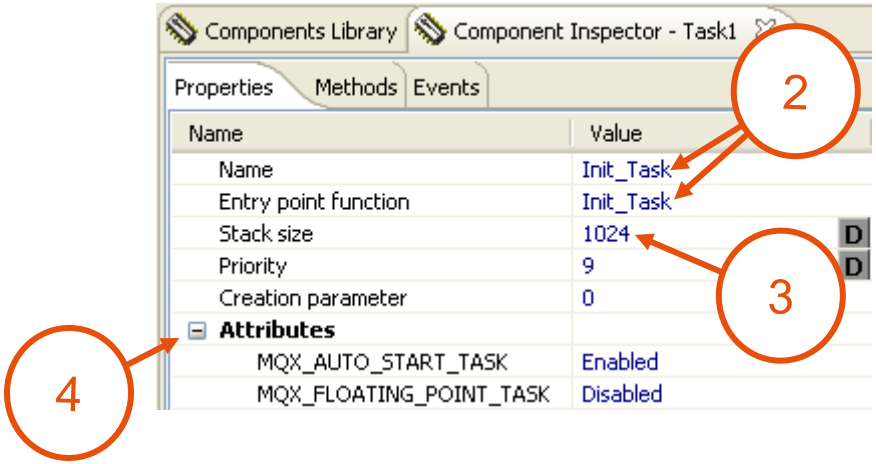
☐Add files and walk through the task code

☐Build, debug, play with the pretty colors

Configure Task1

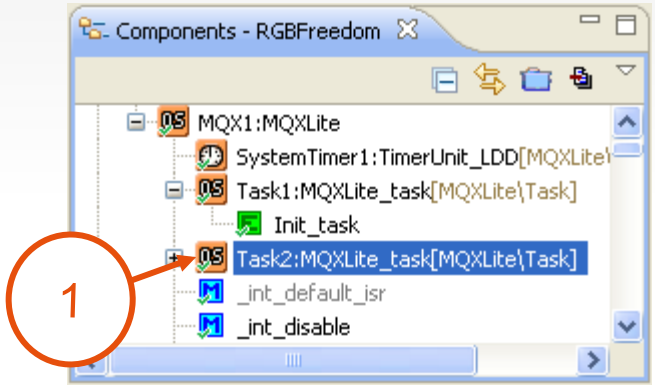
- Tasks are separate entities
  - Select Task1
- Names MATTER!
  - Code is generated based on names
- **Name = Init_Task**
- **Entry point function = Init_Task**
  - Names are case sensitive
- **Stack size = 1024**
- **MQX_AUTO_START_TASK = Enabled**
- Press Ctrl-S to save settings

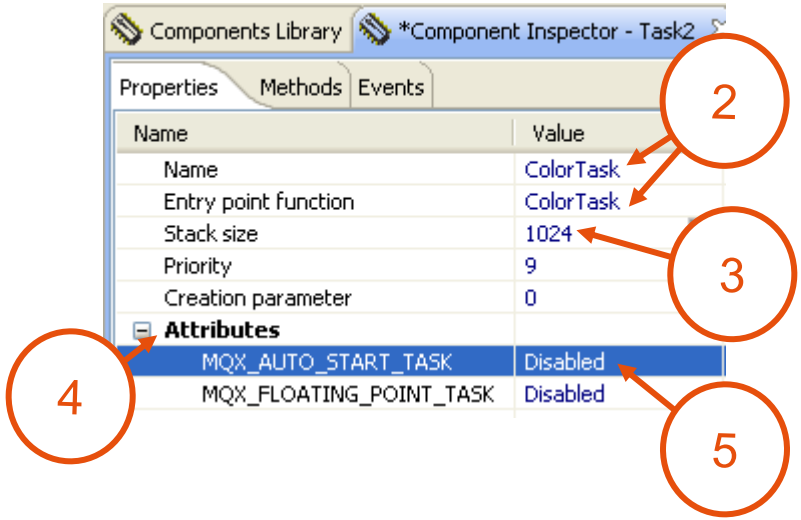




Configure Task2

- Select Task2
- **Name = ColorTask**
- **Entry point function = ColorTask**
- **Stack size = 1024**
- **MQX_AUTO_START_TASK = Disabled**
  - Init_Task will instantiate ColorTask
- Press Ctrl-S to save settings





LAB 4 – Create An MQX Lite Project

- This hands-on lab shows how to…

☑Create an MQX Lite Project and configure the CPU (48 MHz)

☑Configure the RTOS timer and add tasks

☑Configure the tasks: Init_Task, ColorTask

☐Add and configure components

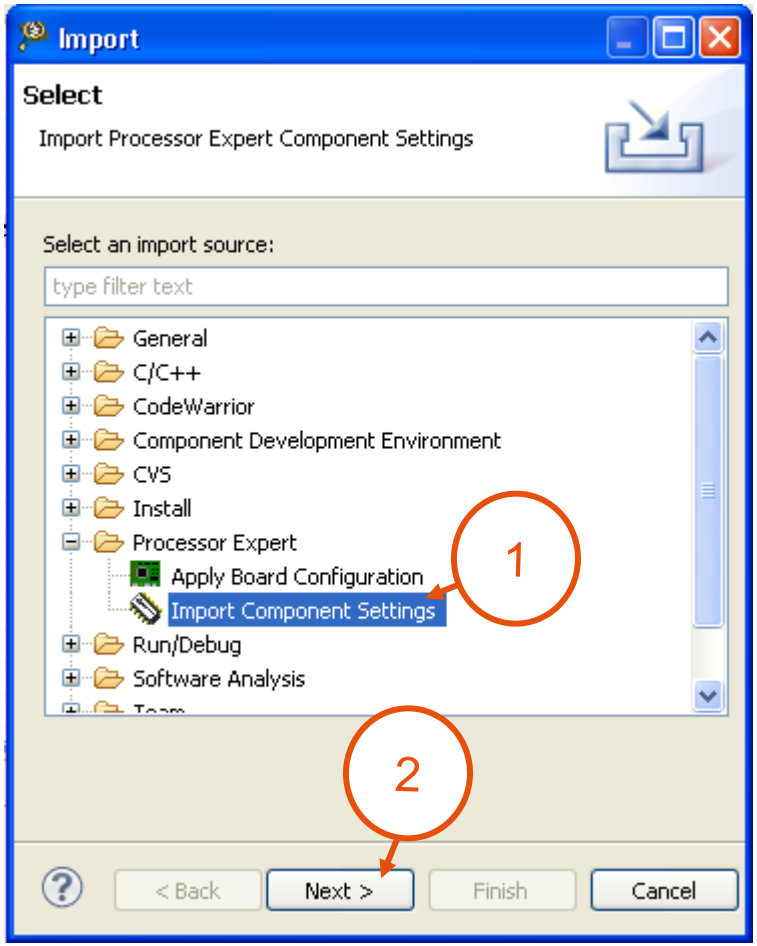
☐Generate code and walk through the code model

☐Add files and walk through the task code

☐Build, debug, play with the pretty colors

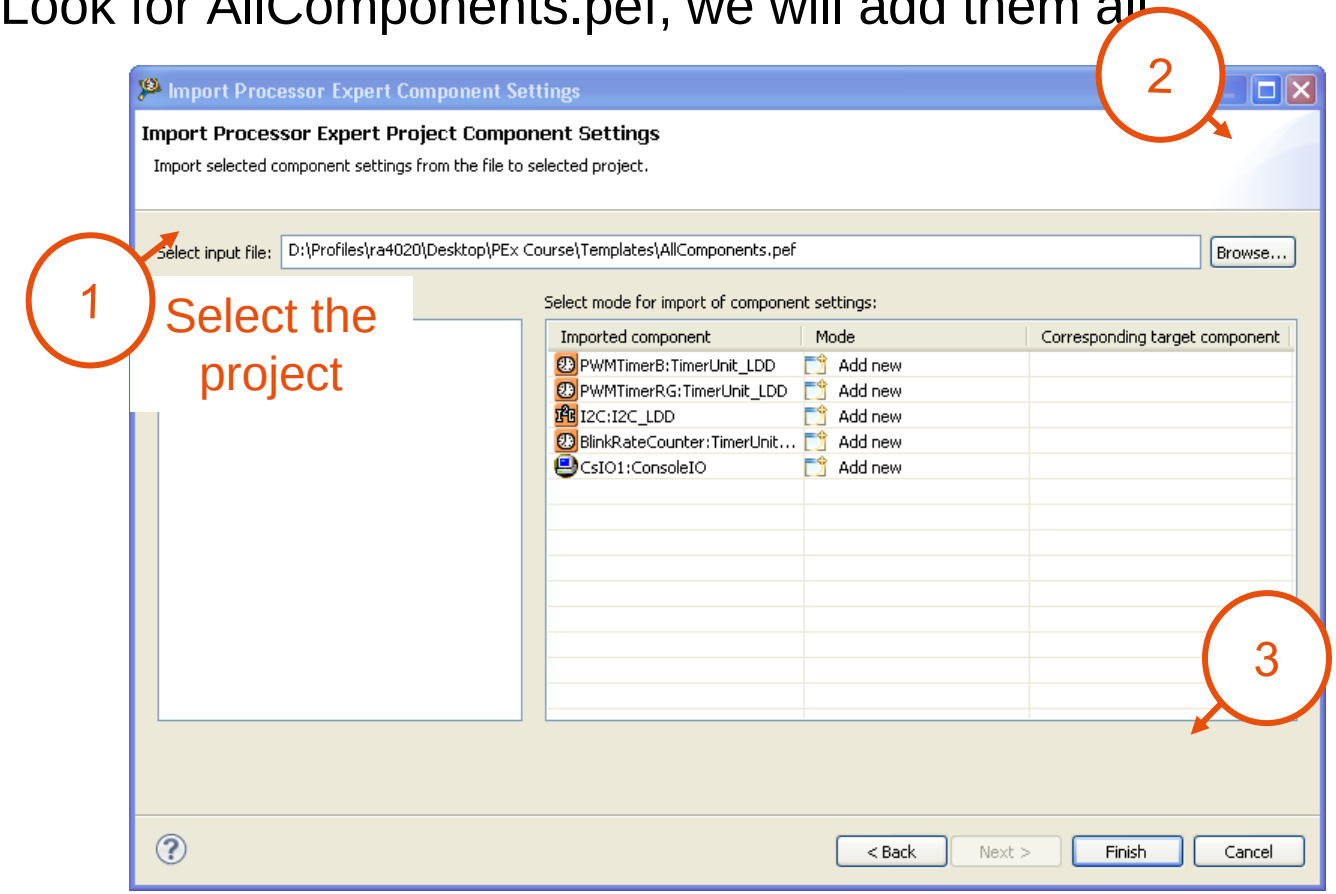
Add and Configure Components – the Easy Way

- Import Component Templates
- File→Import
- Processor Expert→
  Import Component Settings
- Click Next

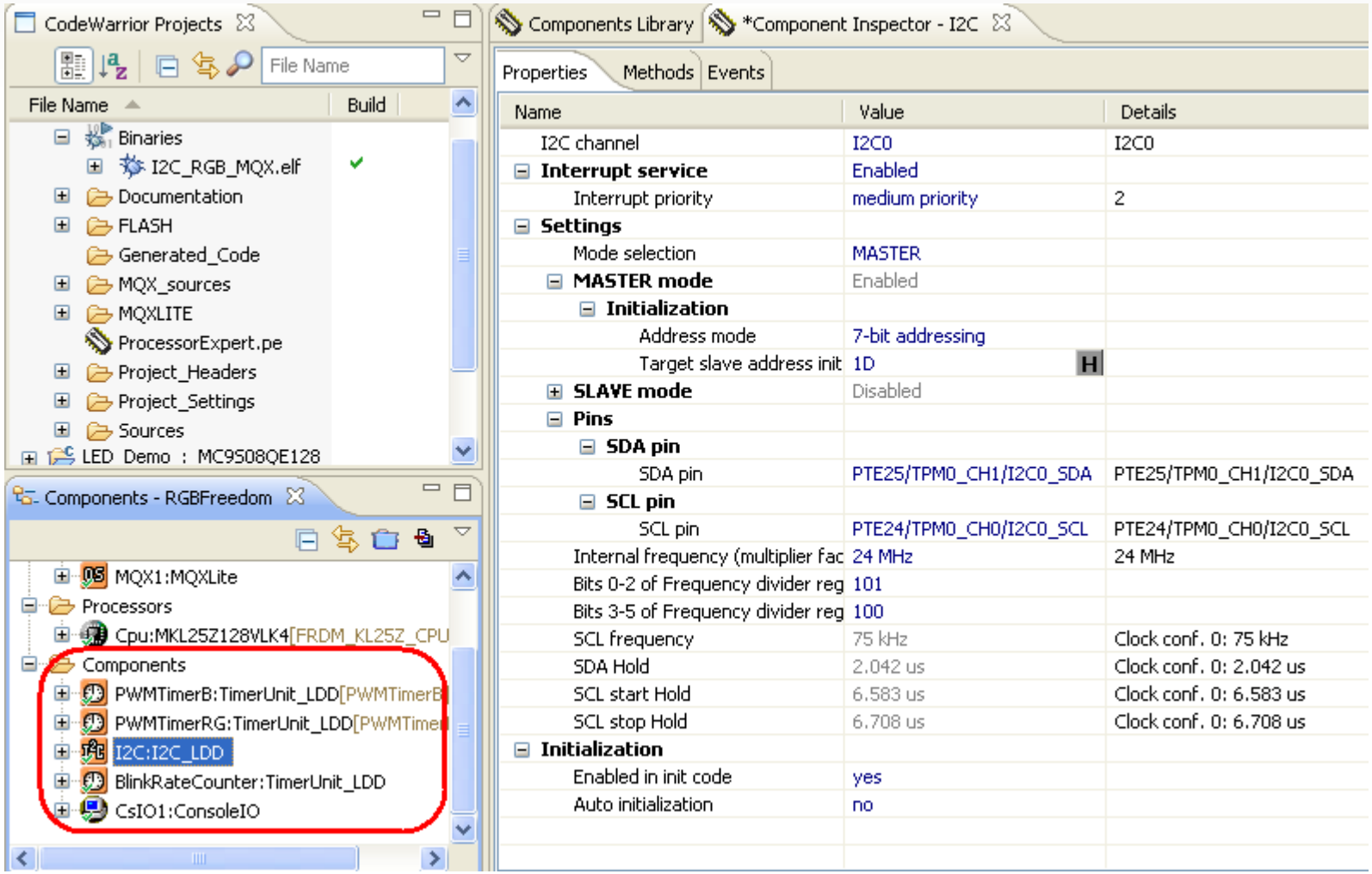


Open a Component Settings File

- Select the project to which to apply the components
- Navigate to the ..\PEx Course\Templates folder on Desktop
  - Look for AllComponents.pef, we will add them all



Components Appear Fully Configured



LAB 4 – Create An MQX Lite Project

- This hands-on lab shows how to…

☑Create an MQX Lite Project and configure the CPU (48 MHz)

☑Configure the RTOS timer and add tasks

☑Configure the tasks: Init_Task, ColorTask

☑Add and configure components

☐Generate code and walk through the code model

☐Add files and walk through the task code

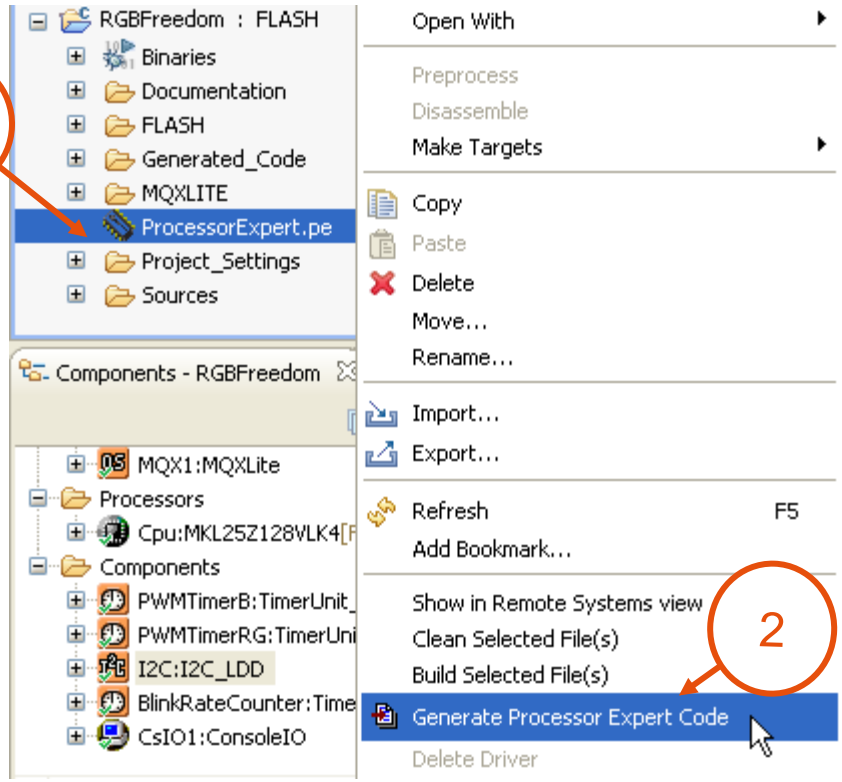
☐Build, debug, play with the pretty colors

Generate Code

- Click the “Generate Processor Expert Code” icon

- OR –

Right click



- Right click the .pe file
- Select “Generate Processor Expert Code”

Where is main()?

- main() is in ProcessorExpert.c in Sources folder
  - On line 51 in this case
- Double click on the file to open it in the Editor
- There is a call to PEX_RTOS_START() in line 66
  - Let's track that down
    - Right-click on PEX_RTOS_START()
    - Select Open Declaration (F3)
  - It is a call to `_mqxlite()`
    - Right-click on `_mqxlite()`
    - Select Open Declaration (F3)
- `_mqxlite()` is the init function for the OS
  - Sets up timer
  - Creates idle task
  - Creates autostart tasks

mqxlite.c Creates the Tasks

ProcessorExpert.c | MQX1.h | mqxlite.c

```
281     /* Create the idle task */
282 #if MQX_USE_IDLE_TASK
283     td_ptr = _task_init_internal((TASK_TEMPLATE_STRUCT_PTR)&kernel_data->IDLE_TASK_TEMPLATE,
284                                  kernel_data->ACTIVE_PTR->TASK_ID,
285                                  (uint_32)0,
286                                  FALSE,
287                                  (pointer)&idle_task_stack[0],
288                                  (_mem_size)kernel_data->IDLE_TASK_TEMPLATE.TASK_STACKSIZE);
289 #if MQX_CHECK_ERRORS
290     if (td_ptr == NULL) {
291         _mqx_exit(MQX_OUT_OF_MEMORY);
292     } /* Endif */
293 #endif
294     _task_ready_internal(td_ptr);
295 #endif
296
297     /* Check here for auto-create tasks, and create them here */
298     {
299         TASK_TEMPLATE_STRUCT_PTR    template_ptr;
300         int                         task_index = 0;
301
302         template_ptr = kernel_data->INIT.TASK_TEMPLATE_LIST;
303
304         while (template_ptr->TASK_TEMPLATE_INDEX) {
305             if (template_ptr->TASK_ATTRIBUTES & MQX_AUTO_START_TASK) {
306                 td_ptr = _task_init_internal(template_ptr,
307                                              kernel_data->ACTIVE_PTR->TASK_ID,
308                                              template_ptr->CREATION_PARAMETER,
309                                              FALSE,
310                                              (pointer)task_stack_pointers[task_index],
311                                              (_mem_size)template_ptr->TASK_STACKSIZE);
```

Task Templates are Generated from Component

- Open task_template_list.c in Generated_Code folder
- Here are our tasks
  - Based on the properties set in the component
  - One is an auto-start task as we specified

*task_template_list.c

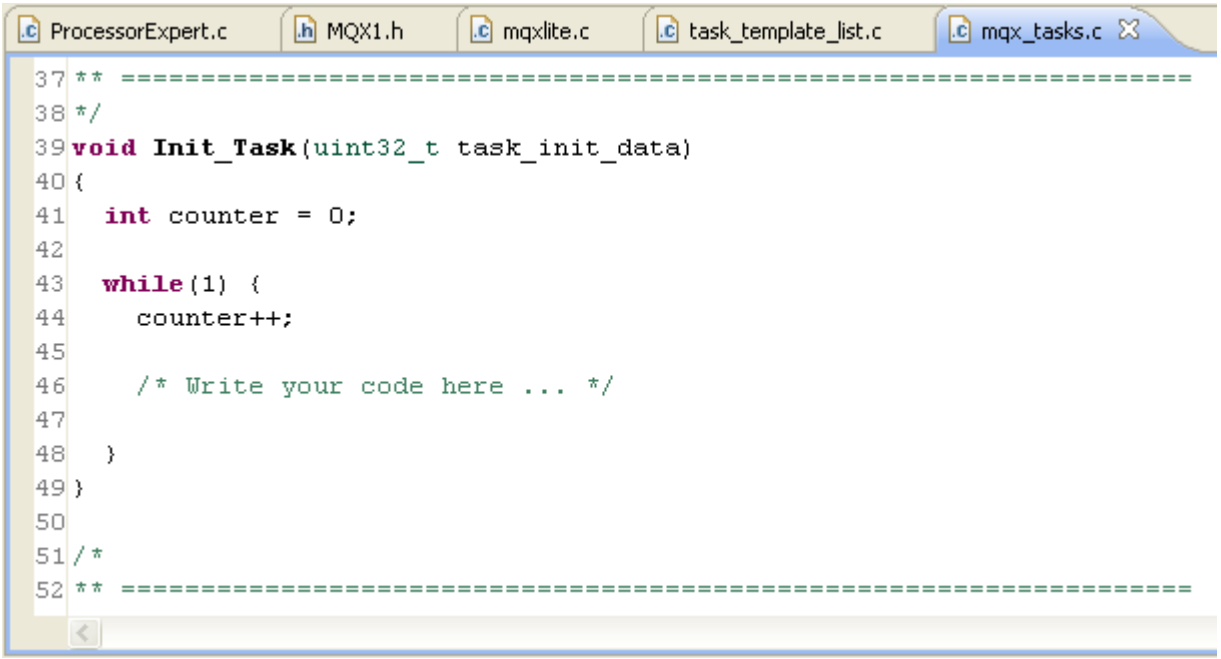
```
#define TASK_TEMPLATE_LIST_END    {0, 0, 0, 0, 0, 0, 0}

/* MQX task template list */
const TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
  /* Task: Init_Task */
  {
  /* Task number            */  INIT_TASK_TASK,
  /* Entry point            */  (TASK_FPTR)Init_Task,
  /* Stack size             */  INIT_TASK_TASK_STACK_SIZE,
  /* Task prior             */  9U,
  /* Task name              */  "init_task",
  /* Task attributes        */  (MQX_AUTO_START_TASK),
  /* Task parameter         */  (uint32_t)(0)
  },
  /* Task: ColorTask */
  {
  /* Task number            */  COLORTASK_TASK,
  /* Entry point            */  (TASK_FPTR)ColorTask,
  /* Stack size             */  COLORTASK_TASK_STACK_SIZE,
  /* Task priority          */  9U,
  /* Task name              */  "colortask",
  /* Task attributes        */  (0),
  /* Task parameter         */  (uint32_t)(0)
  },
  TASK_TEMPLATE_LIST_END
};
```

Where Is the Task Code?

- Task code is in mqx_tasks.c in Sources folder
  - Each task entry point function has a stub

```
/*Write your code here */
```

  - You create the functionality



Where Is the ISR Code?

- Interrupt handlers are part of driver code
  - Driver code can be found in Generated Code folder
  - Driver code calls out to event handlers
- Event handlers are in events.c in Sources folder
  - Each event has a stub

```
/*Write your code here */
```

  - You create the functionality

```
60 ** ===================================================================
61 */
62 void BlinkRateCounter_OnCounterRestart(LDD_TUserData *UserDataPtr)
63 {
64   /* Write your code here ... */
65 }
66
67 /*
```

LAB 4 – Create An MQX Lite Project

- This hands-on lab shows how to…

☑Create an MQX Lite Project and configure the CPU (48 MHz)

☑Configure the RTOS timer and Add tasks

☑Configure the tasks: Init_Task, ColorTask,

☑Import components

☑Generate code and walk through the code model

☐Add files and walk through the task code

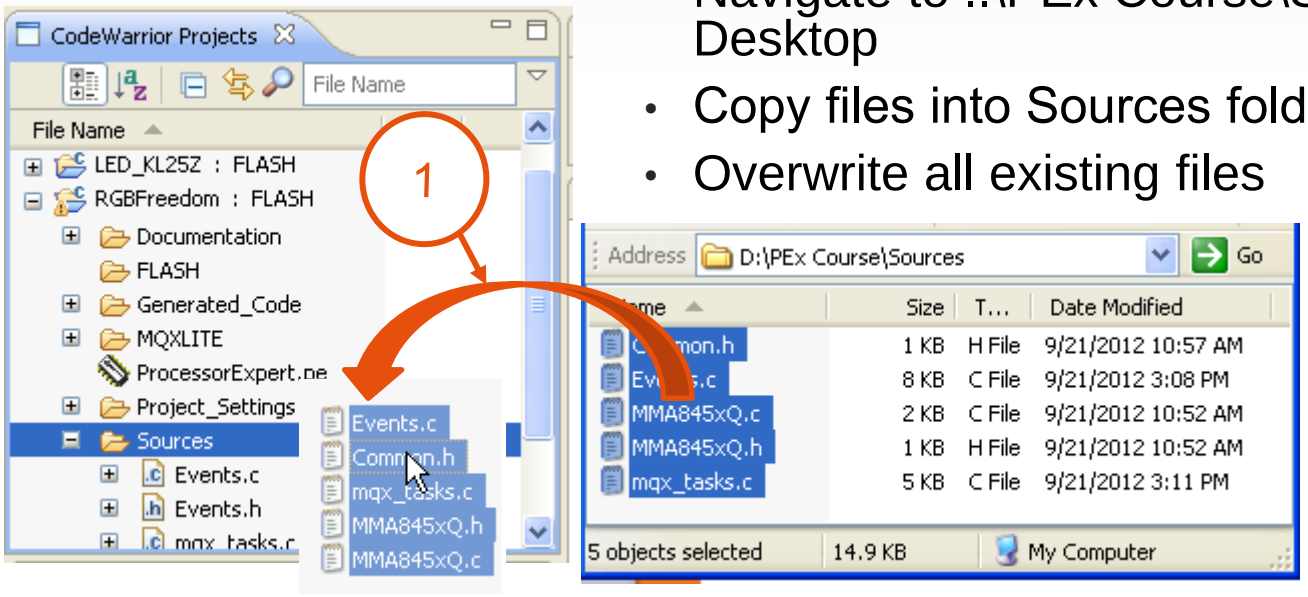
☐Build, debug, play with the pretty colors

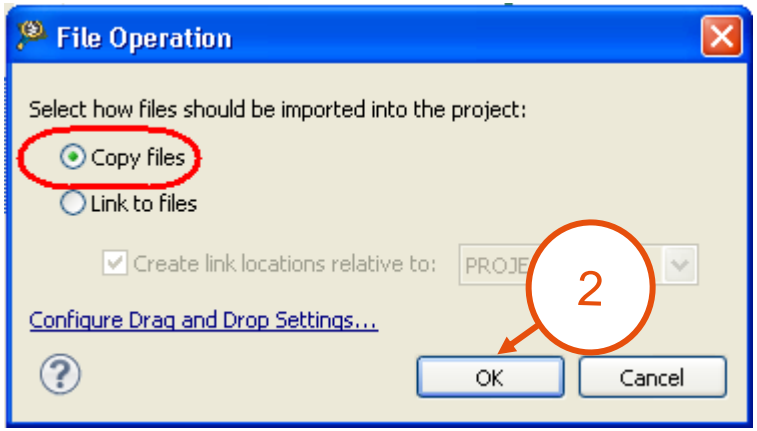
We Will Add Files to Implement Functionality

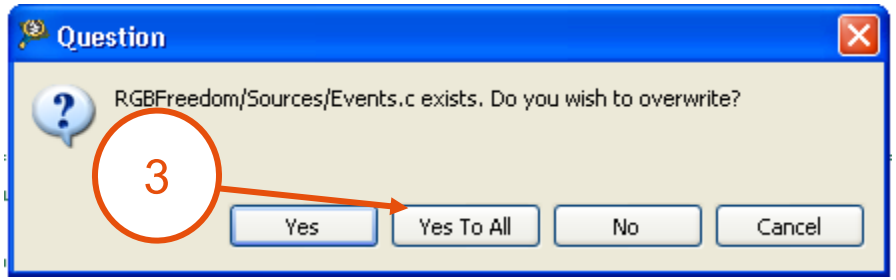
- This lab is about configuring MQX Lite
  - As we teach you about Processor Expert in general
- This course is not about how to write an application
- BUT… we want to show you how a task is implemented
  - So we're going to give you the task code
  - And walk through how it all works

Copy Files into Project, Overwrite All Existing Files

- Navigate to ..\PEx Course\Sources on Desktop
- Copy files into Sources folder by drag ‘n drop.
- Overwrite all existing files







What did we just do?

- mqx_tasks.c has the task implementations
  - Replaces the function stubs that code generation created
  - Processor Expert will not overwrite these functions
- Events.c has implementation of event handlers
  - Replaces the function stubs that code generation created
  - Processor Expert will not overwrite these functions
- MMA845xQ.c is the code for the accelerometer
  - This is just a “gimme”
  - If this accelerometer was a component, we would not need this

Study Init_Task

- Open mqx_tasks.c and go to Init_Task()
  - Lines 69-79 test the accelerometer before the app starts
  - Lines 95-97 initialize the LED timers and the blink rate

```
PWMTimerRG_DeviceData = PWMTimerRG_Init(NULL);
PWMTimerB_DeviceData = PWMTimerB_Init(NULL);
PeriodicTimer = BlinkRateCounter_Init(NULL);
```

  - Then we initialize the color task

```
_task_create_at(0, COLORTASK_TASK, 0, ColorTask_task_stack,
COLORTASK_TASK_STACK_SIZE);
```

Study ColorTask

```
Error = !ReadAccRegs(I2C_DeviceData, &DataState, OUT_X_MSB, 3 *
  ACC_REG_SIZE, (uint8_t*) Color);  // Read accelerometer
   if (!Error) {
    if (!BlinkFlag) {
      PWMTimerRG_Enable(PWMTimerRG_DeviceData);
      PWMTimerB_Enable(PWMTimerB_DeviceData);
      PWMTimerRG_SetOffsetTicks(PWMTimerRG_DeviceData,
  0,1000*(1<<(abs(Color[0]/10)))); // x axis - red LED
      PWMTimerRG_SetOffsetTicks(PWMTimerRG_DeviceData, 1,
  1000*(1<<(abs(Color[1]/10)))); // y axis - green LED
      PWMTimerB_SetOffsetTicks(PWMTimerB_DeviceData, 0,
  1000*(1<<(abs(Color[2]/10)))); // z axis - blue LED
    }
   }
```

Study BlinkRateCounter_OnCounterRestart

- Open events.c and go to BlinkRateCounter_OnCounterRestart

```
BlinkFlag ^= 1;
  if (BlinkFlag) {
    PWMTimerB_Disable(PWMTimerB_DeviceData);  //Disable LEDs for a
blink period.
    PWMTimerRG_Disable(PWMTimerRG_DeviceData);
  }
```

- The event handler uses methods from components

LAB 4 – Create An MQX Lite Project

- This hands-on lab shows how to…

☑Create an MQX Lite Project and configure the CPU (48 MHz)

☑Configure the RTOS timer and Add tasks

☑Configure the tasks: InitTask, ColorTask, TSSTrigger

☑Import components

☑Generate code and walk through the code model

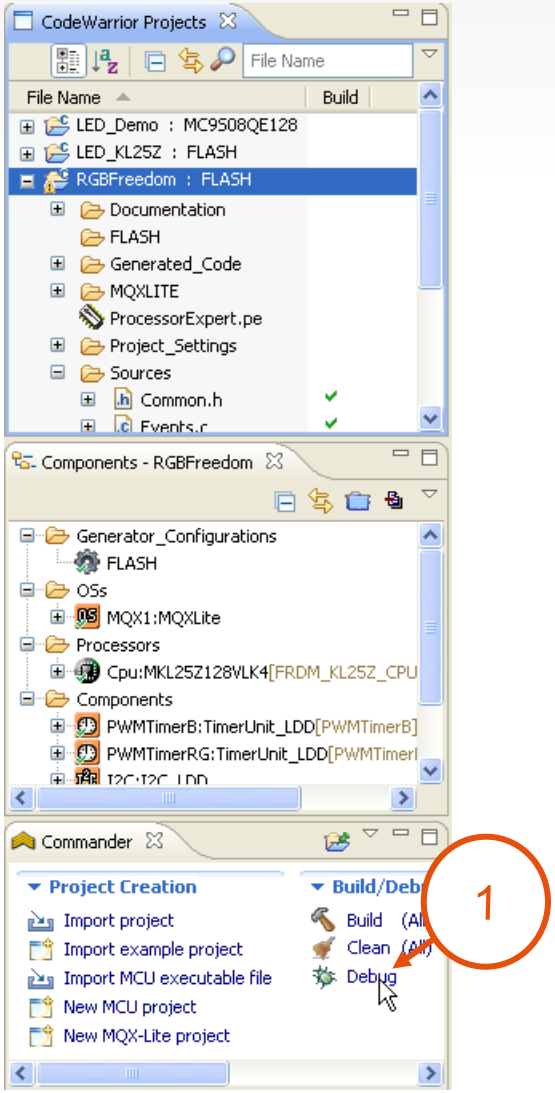
☑Add files and walk through the task code

☐Build, debug, play with the pretty colors

Debug

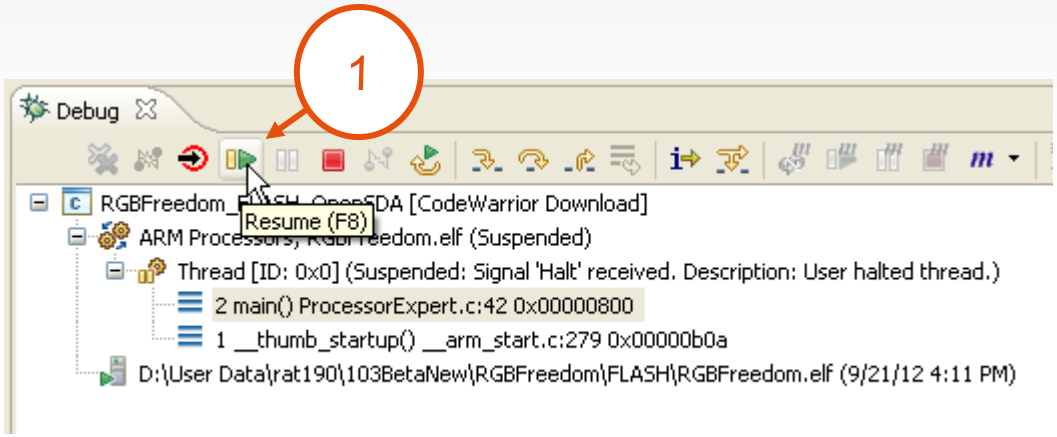
- Click Debug in the Commander View
  - One click will do it all
    - Generate code
    - Build Code
    - Download
    - Launch Debugger

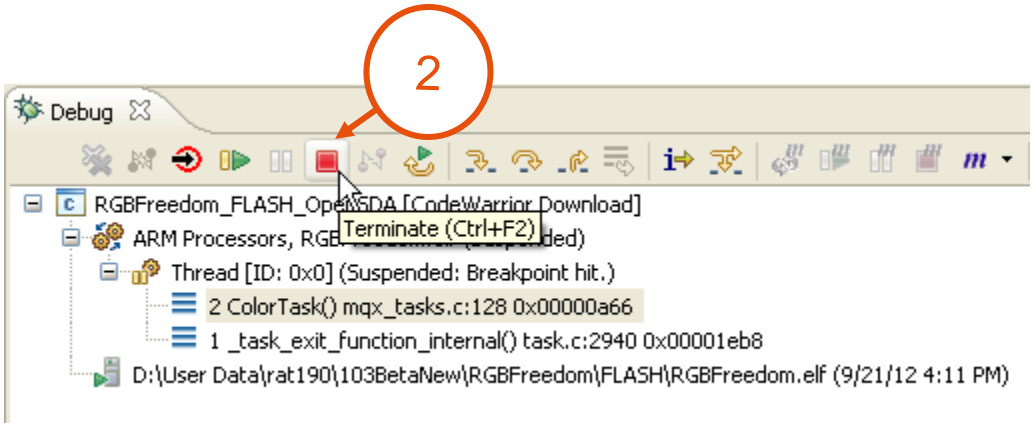
    (This works because there is only one debug configuration)



Run the Code

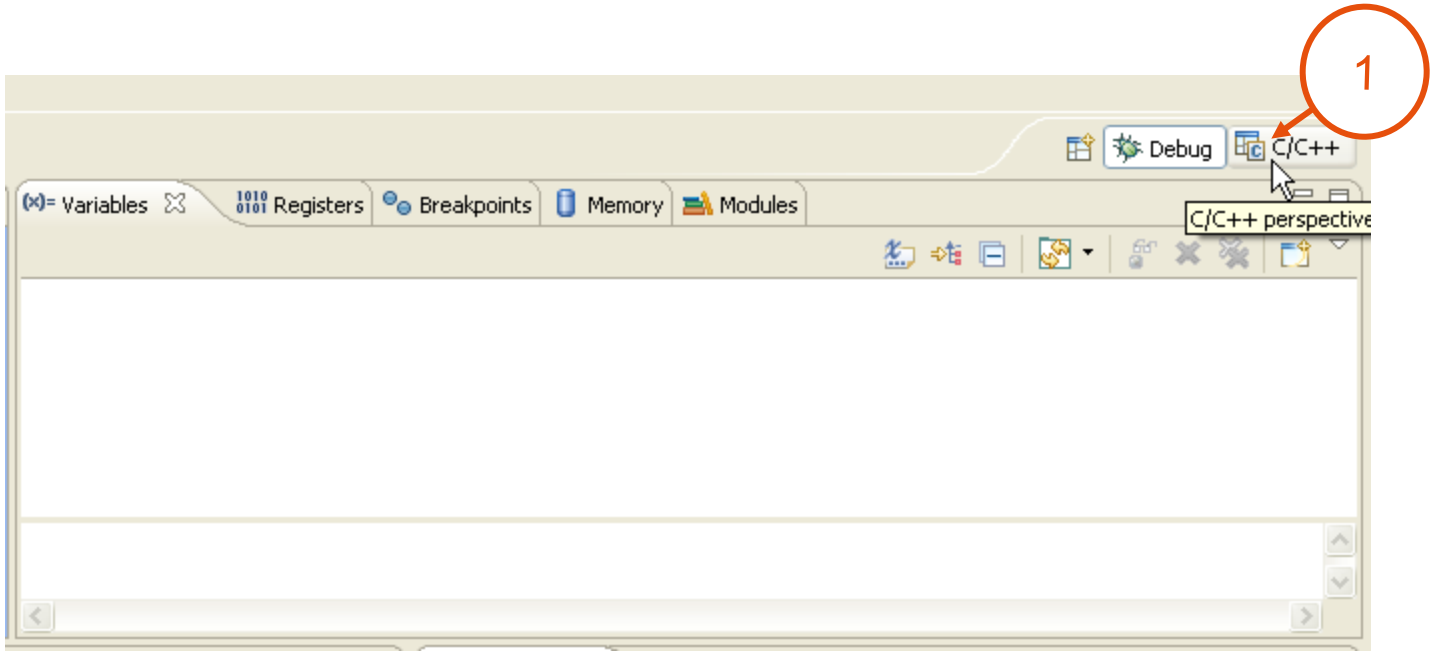
- Click Resume
- The LED flashes
- Tilt the board and watch the colors change
  - The ColorTask reads the accelerometer and adjusts
- Click Terminate when done





Return to the C/C++ Perspective

- Just to get ready for the next lab



What You Accomplished

- Created an MQX Lite application
- Saw how easy it is to configure and instantiate tasks
- Saw where to implement code (the code model)
  - In mqx_tasks.c
- Got more experience with Processor Expert including
  - TEMPLATES for boards, processors, and components

LAB 5: Create New Project

Using Board Configuration File

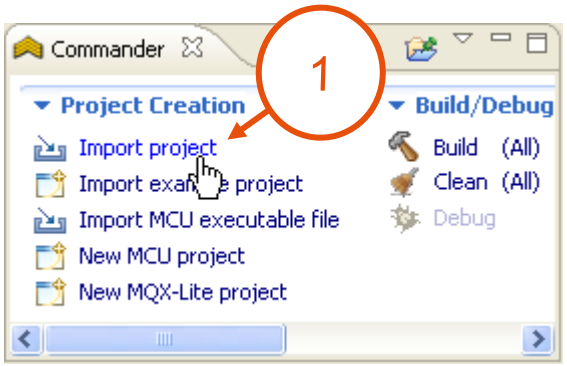
LAB 5 – Create a Board Configuration File

- This hands-on lab shows how to…
  - ☐Export a board configuration file
  - ☐Import an empty project for a Freedom board
  - ☐Configure the entire platform from the configuration file

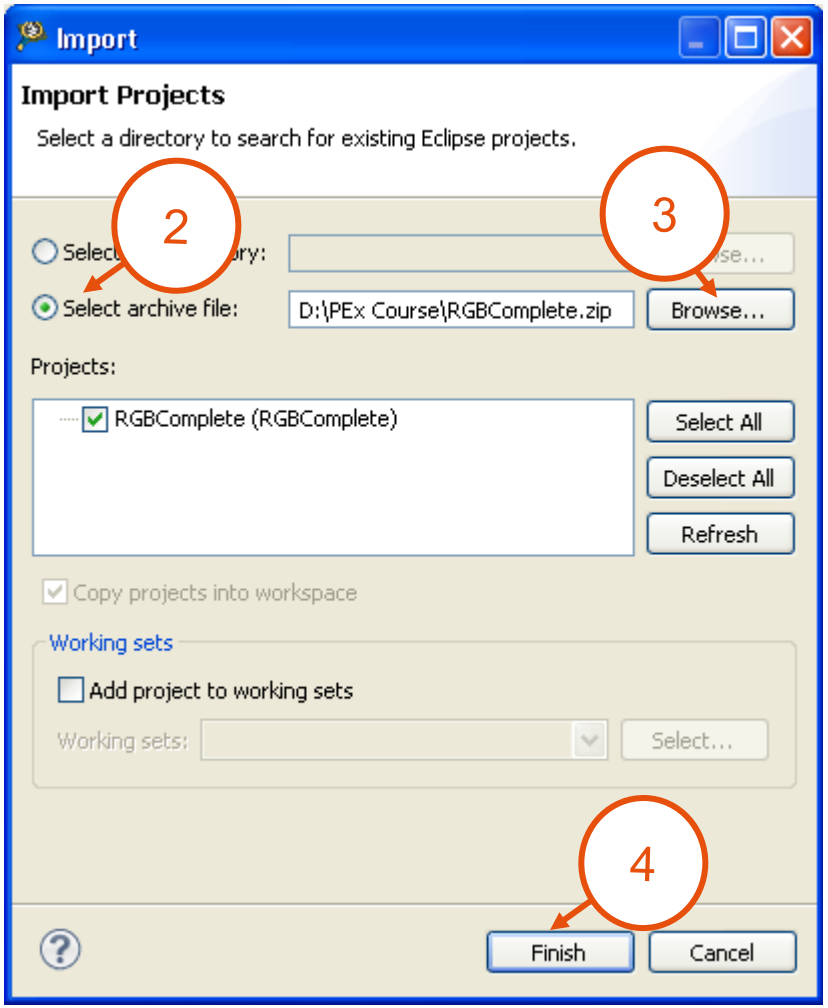
- We'll use the Freedom board

Open a fully configured Project

- Select Import Project in Commander View



- Select archive file
- Browse to RGBComplete in ..\PE Course
- Select Finish

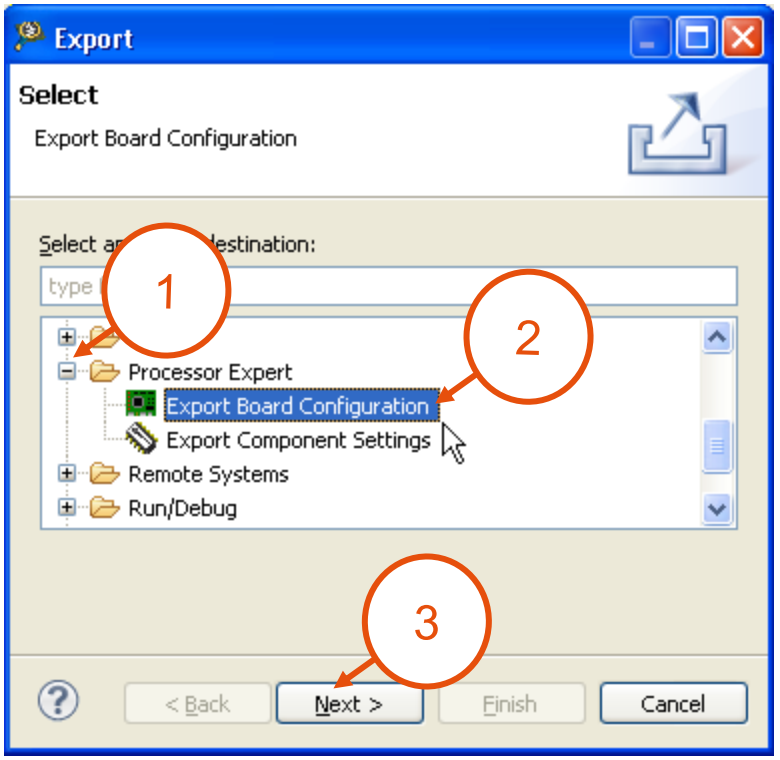


What the application does

- Three Tasks
  - Init Task to set things up
  - Color task changes color as you tilt the board
  - TSS task changes blink rate as you touch the slider
- Trust me?

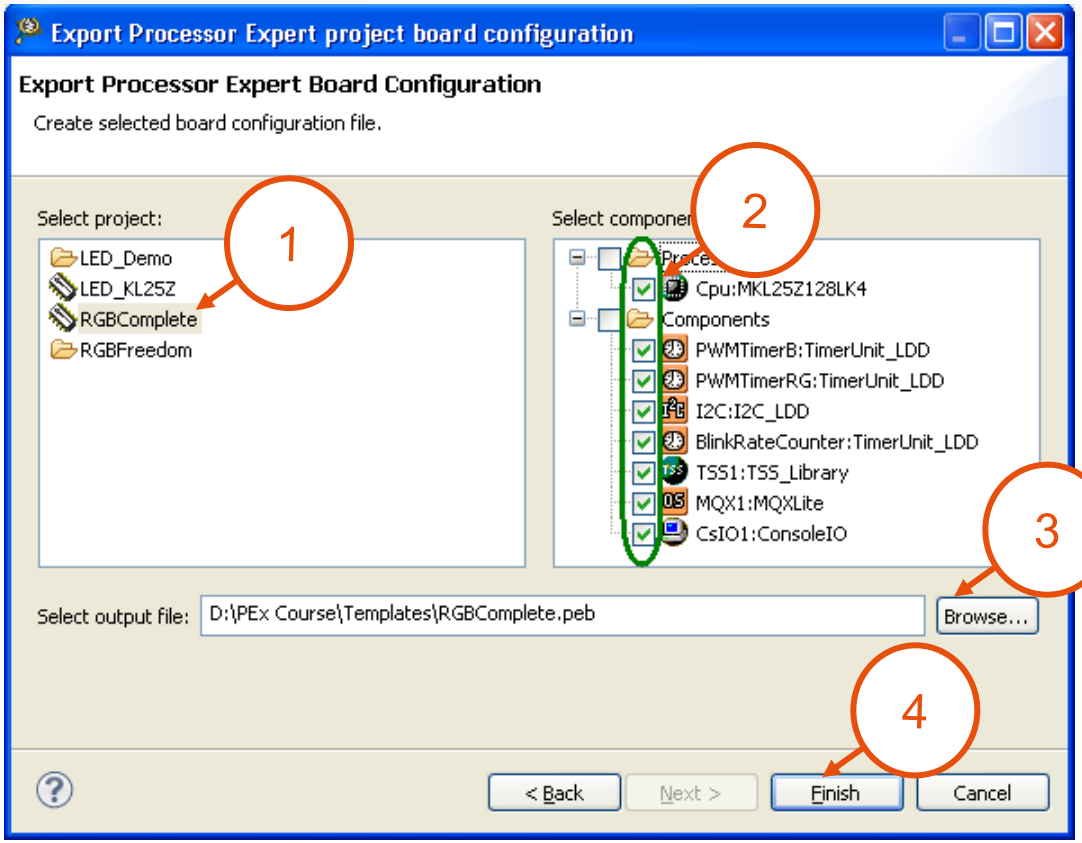
Export board configuration

- Assume your experts have configured a complete system
- Select File→Export
- Select Processor Expert
- Select Export Board Configuration
- Click Next



Save Entire Board Configuration

- Specify source project
- Select all components
- Specify file name
  - RGBComplete
  - Remember where you put it, you'll need it
- Select Finish



What did we just do?

- Exported a file that defines
  - The CPU and its configuration
  - Every component and its configuration
- File is available for you to use as you see fit
- Are you seeing the potential here?

LAB 5 – Create a Board Configuration File

- This hands-on lab shows how to…

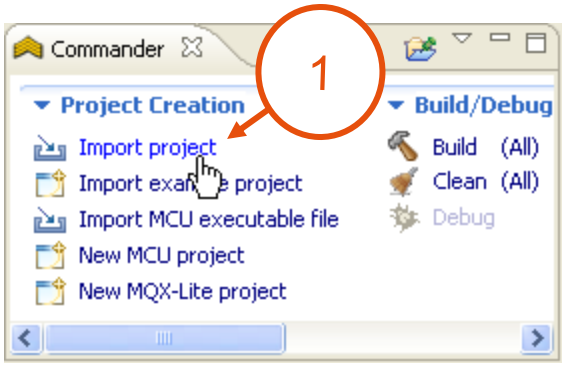
  ☑ Export a board configuration file

  ☐ Import an empty project for a Freedom board

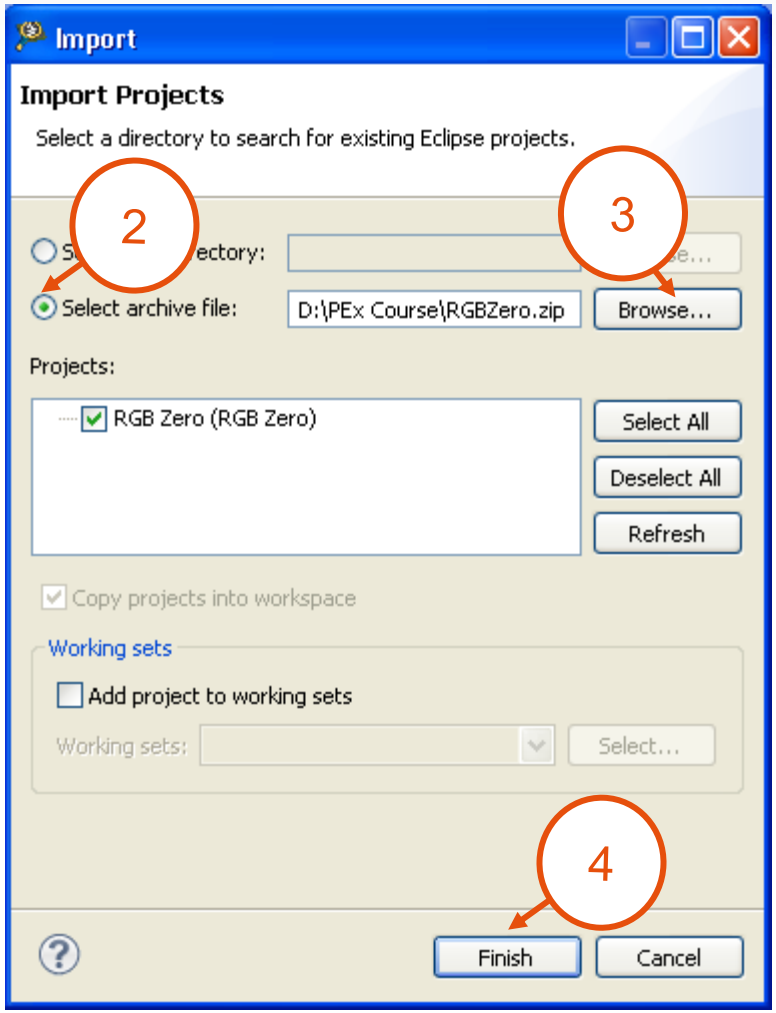
  ☐ Configure the entire platform from the configuration file

Open an Unconfigured Project

- Select Import Project in Commander View



- Select archive file
- Browse to RGBZero in ..\PEx Course on Desktop
  - Project has no components
  - Project has required source files
- Click Finish



What have we got?

- A project with the right processor and package
- Nothing else matters
- We don’t even need to configure the processor
  - Clock settings, interrupts… the works, completely irrelevant
- As the name of our demo project implies, you can start from zero
- NOTE: for the application to work, *we give you source files*!

LAB 5 – Create a Board Configuration File

- This hands-on lab shows how to…

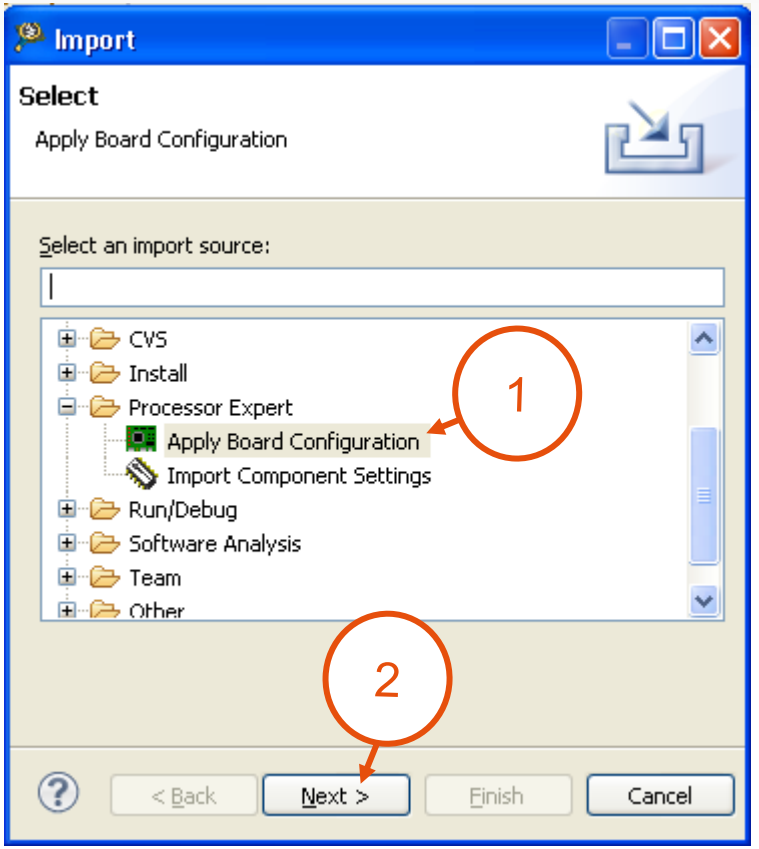
  ☑Export a board configuration file

  ☑Import an empty project for a Freedom board

  ☐Configure the entire platform from the configuration file

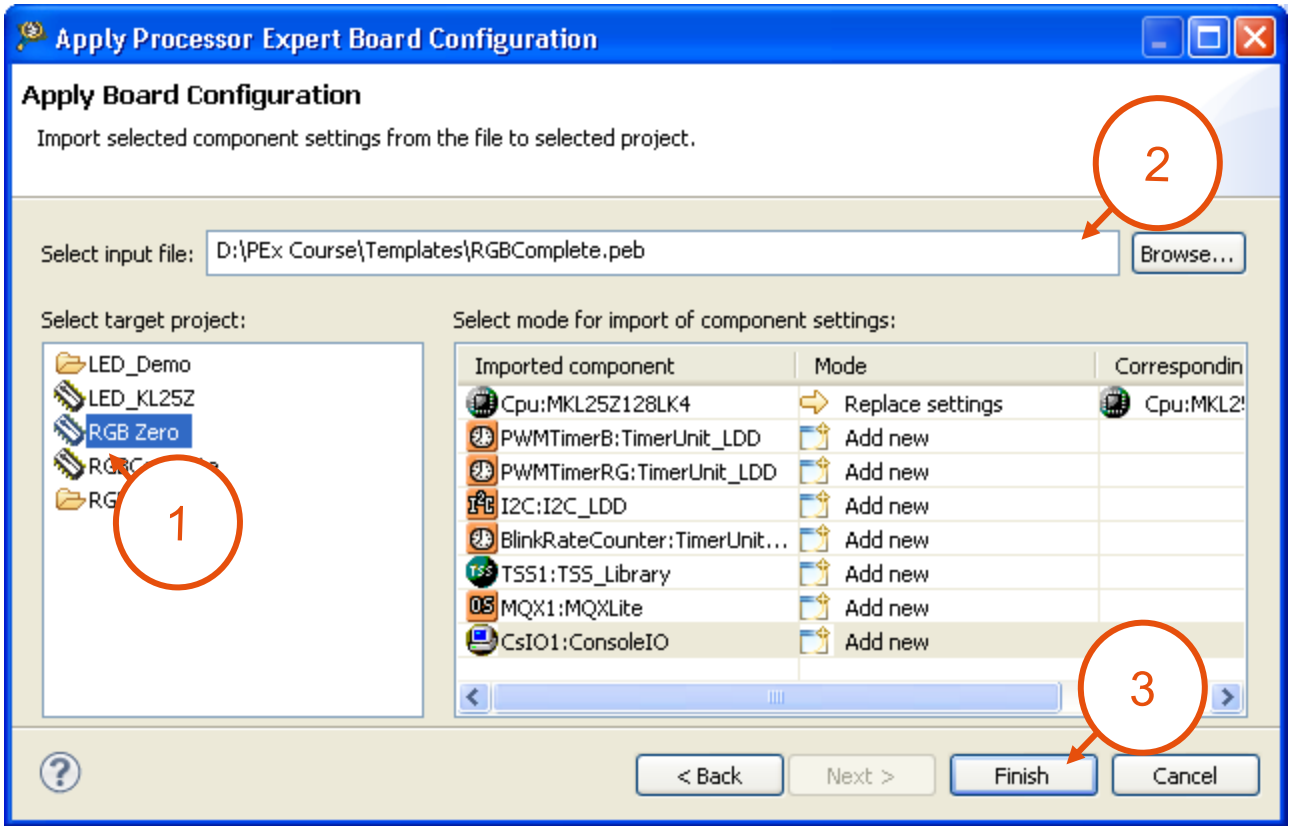
Import a Board Configuration

- Select File→Import
- Select Processor Expert
- Select Apply Board Configuration
- Click Next



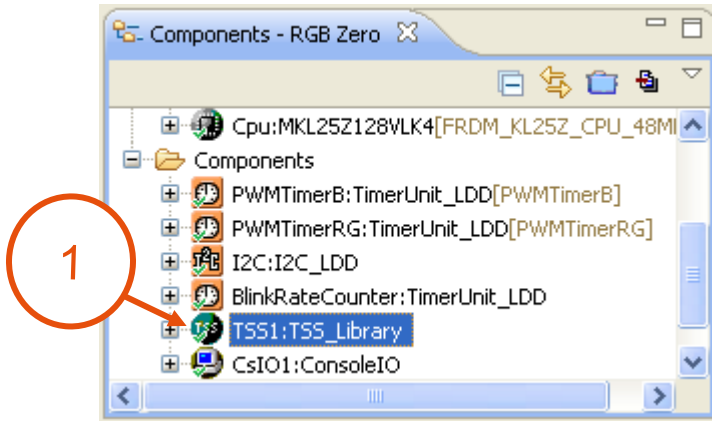
Apply Board Configuration File

- Select project to which the configuration file should be applied
- Navigate to the configuration file you saved and apply it



Project is Fully Configured

- All the components appear
  - Investigate any component



- TSS is a good example
  - Very complex
  - Encapsulates the entire touch sensing library

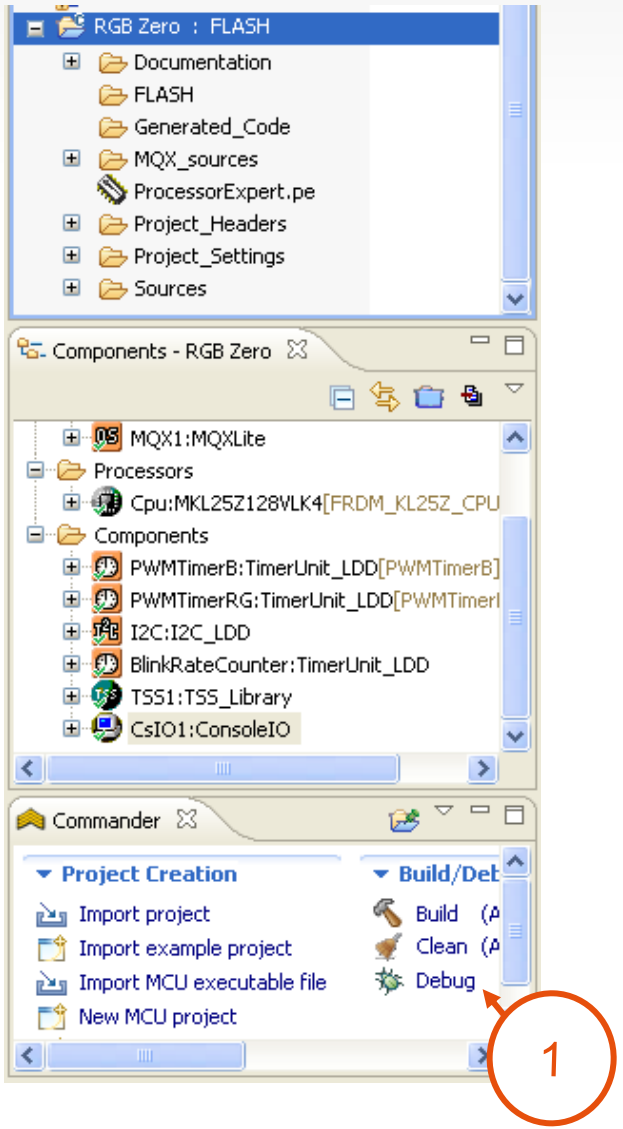
Components Library | Component Inspector - TSS1

Properties | Methods | Events

Name	Value	
Electrode Sensitivity	40	D
Number of Controls	1	
Control0		
Control Type	ASLIDER	
Number of Electrodes	2	D
Config Registers Init	Enabled	
Control Configuration Re		
Event Control and Statu		
Auto Repeat Rate	0	D
Movement Timeout	2	D
Range	32	D
Structure Name	AnalogSlider	
Sensor Settings		
Timers	Disabled	
TSI Module	Enabled	
Auto Calibration		
Resolution	4	D
Calibration Limits	Disabled	
Proximity Auto Calibration	Disabled	
Clock Settings	Enabled	
Active Mode Clock Source	Bus clock	
Active Mode Prescaler	divide by 1	
Active Mode Clock Divider	divide by 2048	
Low Power Clock Source	LPOCLK	
Scan Settings	Disabled	
TSS Config Registers Init	Enabled	
System Configuration Register		
System Enabled	yes	
SWI Enabled	no	

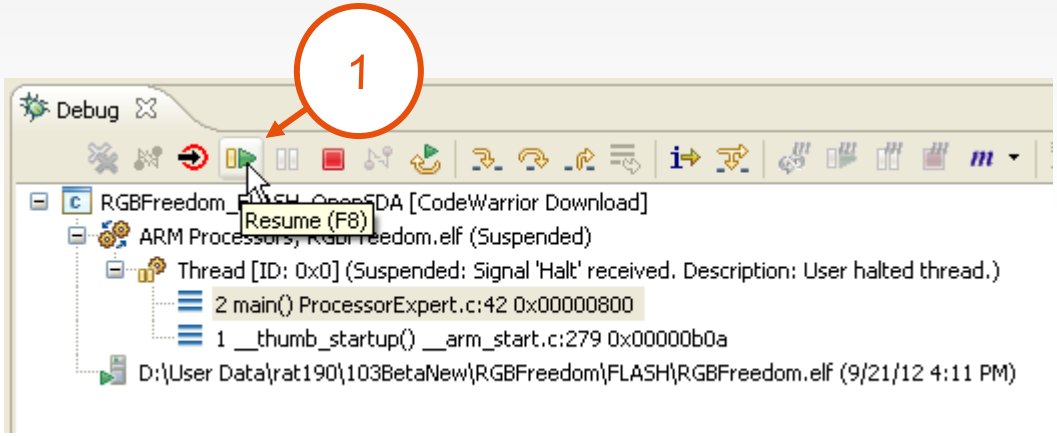
Run the application

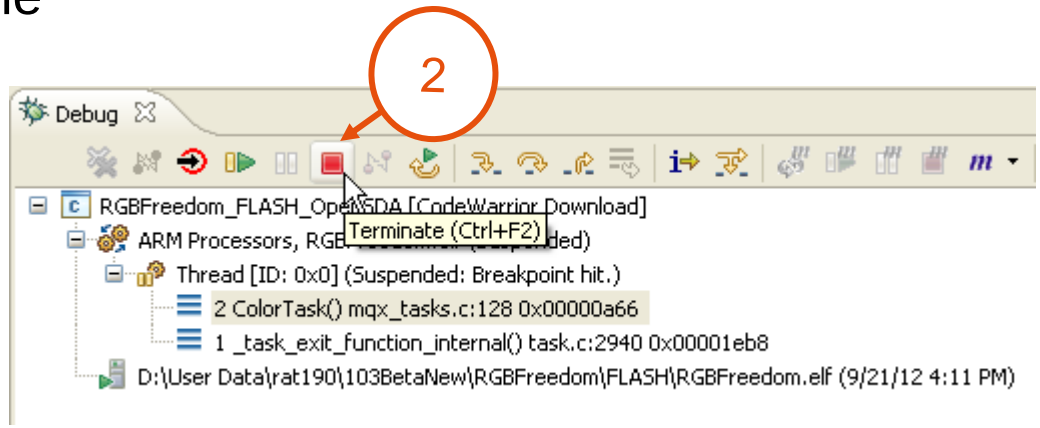
- Make sure the project is selected
  - Unless you have it pinned ☺
- Click Debug in Commander View
  - One click does it all
    - Generates code
    - Builds
    - Downloads
    - Stops at first line of main()



Run the Code

- Click Resume
- The LED flashes
- Tilt the board and watch the colors change
- Touch the slider to change the blink rate
- Click Terminate when done





LAB 5 – Create a Board Configuration File

- This hands-on lab shows how to…

  ☑Export a board configuration file

  ☑Import an empty project for a Freedom board

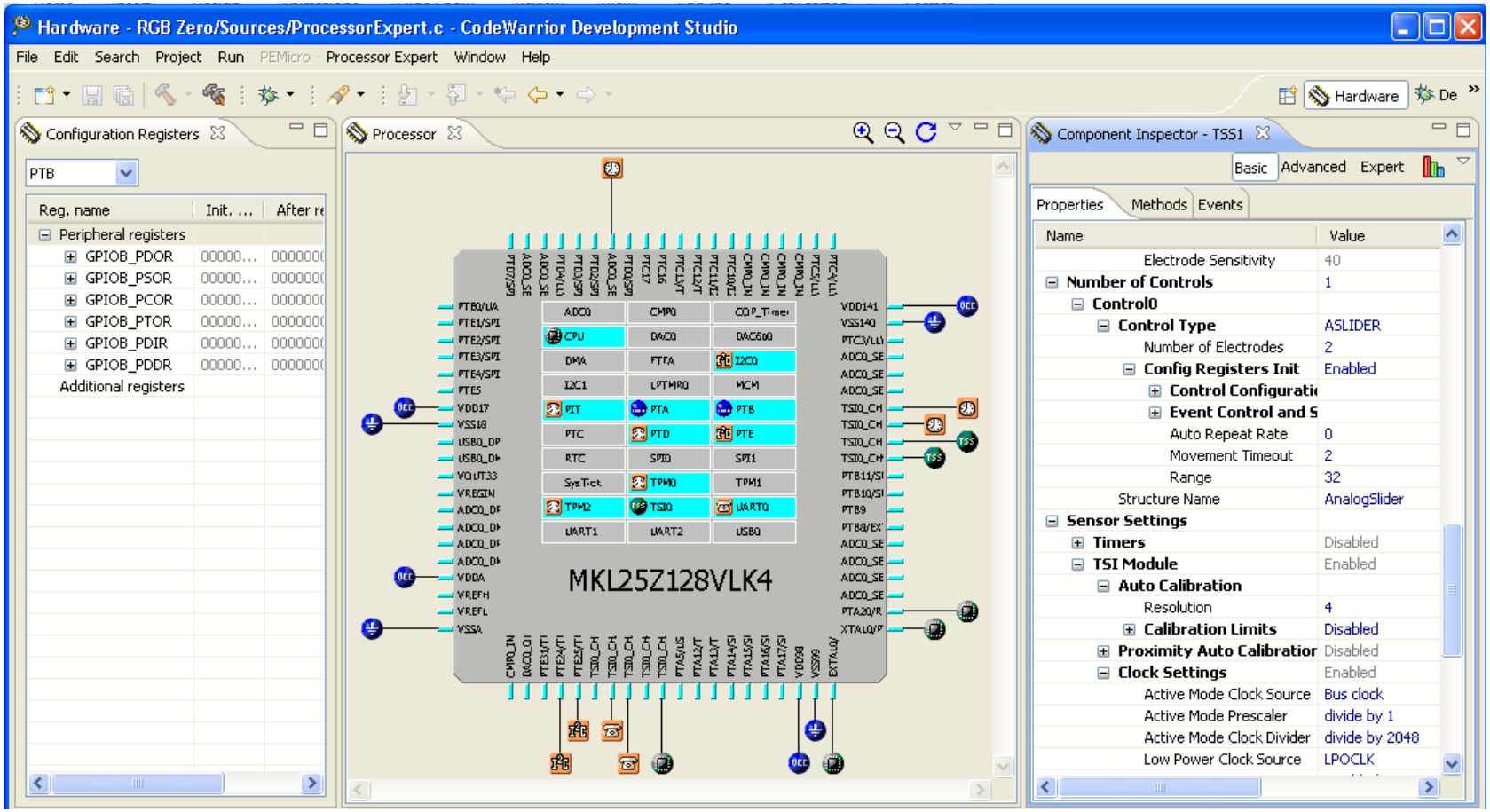
  ☑Configure the entire platform from the configuration file

What You Just Did

- Started from a nothing project
  - Default CPU – not even MQX-Lite-specific
  - We gave you the software sources, e.g. the tasks were defined!
- Configured the processor, the clocks, peripherals, TSS software library, the RTOS, the entire system!
- By importing one file, clicking one button
- You can do this for any arbitrarily complex system
  - With Processor Expert components

This IS the Power of Processor Expert Software

- We aren't making this up, it's real



freescale™

LAB 6: Hardware Perspective

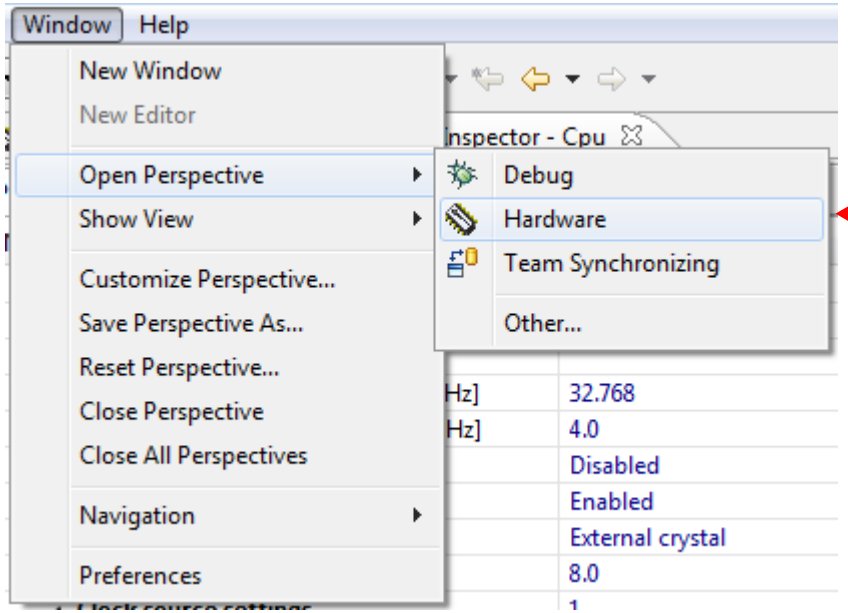
Adding Peripherals

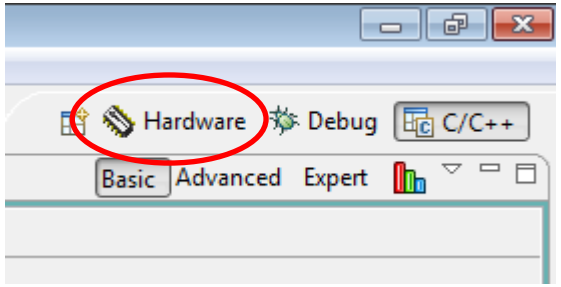
Hardware Perspective

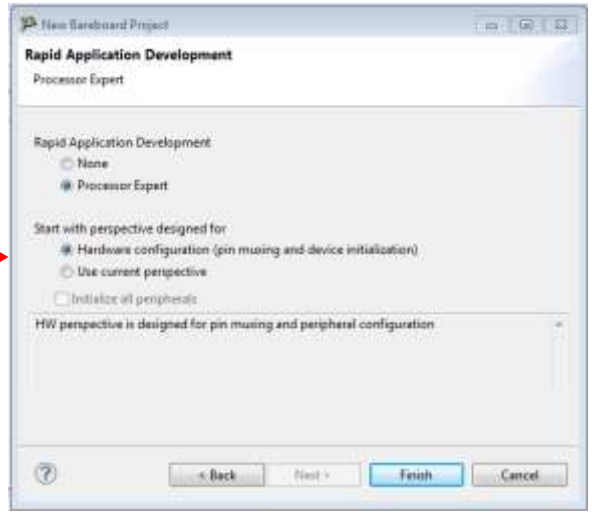
- Review Current Pin Settings
- Add UART Component to the mix
- Select the pins for HW Connection
- Set the parameters for UART
  - Baud rate, start bits, stop bits, etc
- Not going to enable or generate code (won't work with current hardware, Just for show)

Hardware Perspective

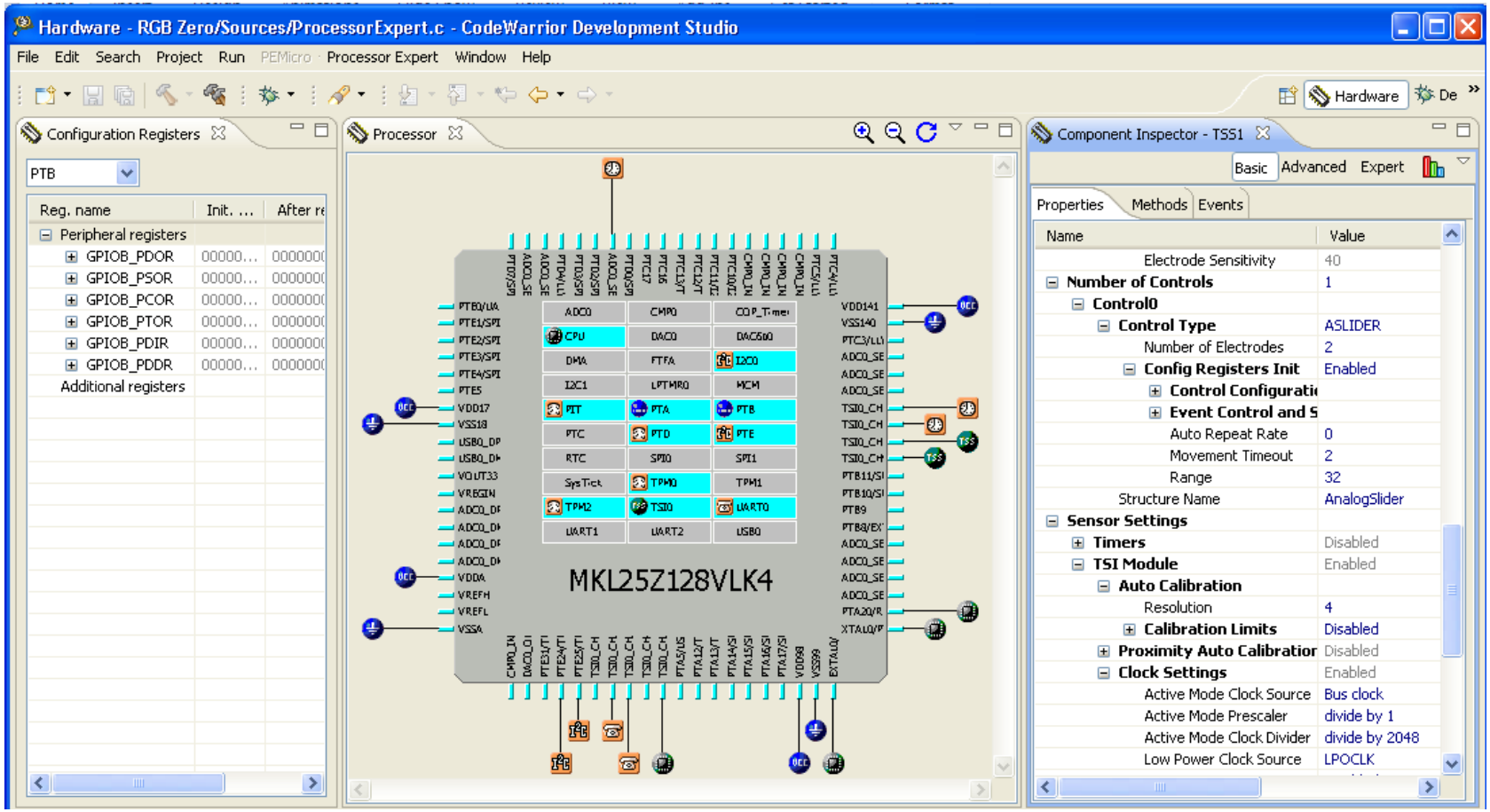
- Select Hardware Perspective



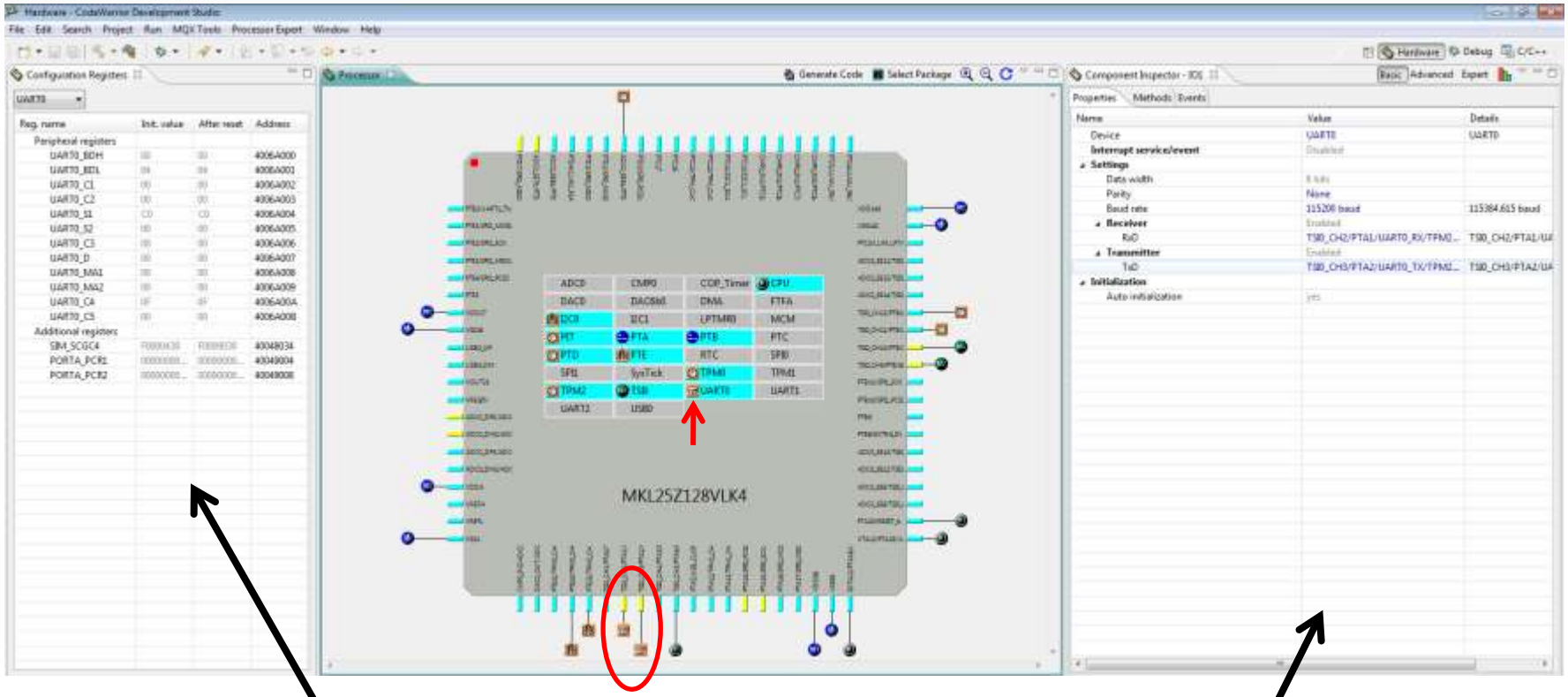




Hardware Perspective

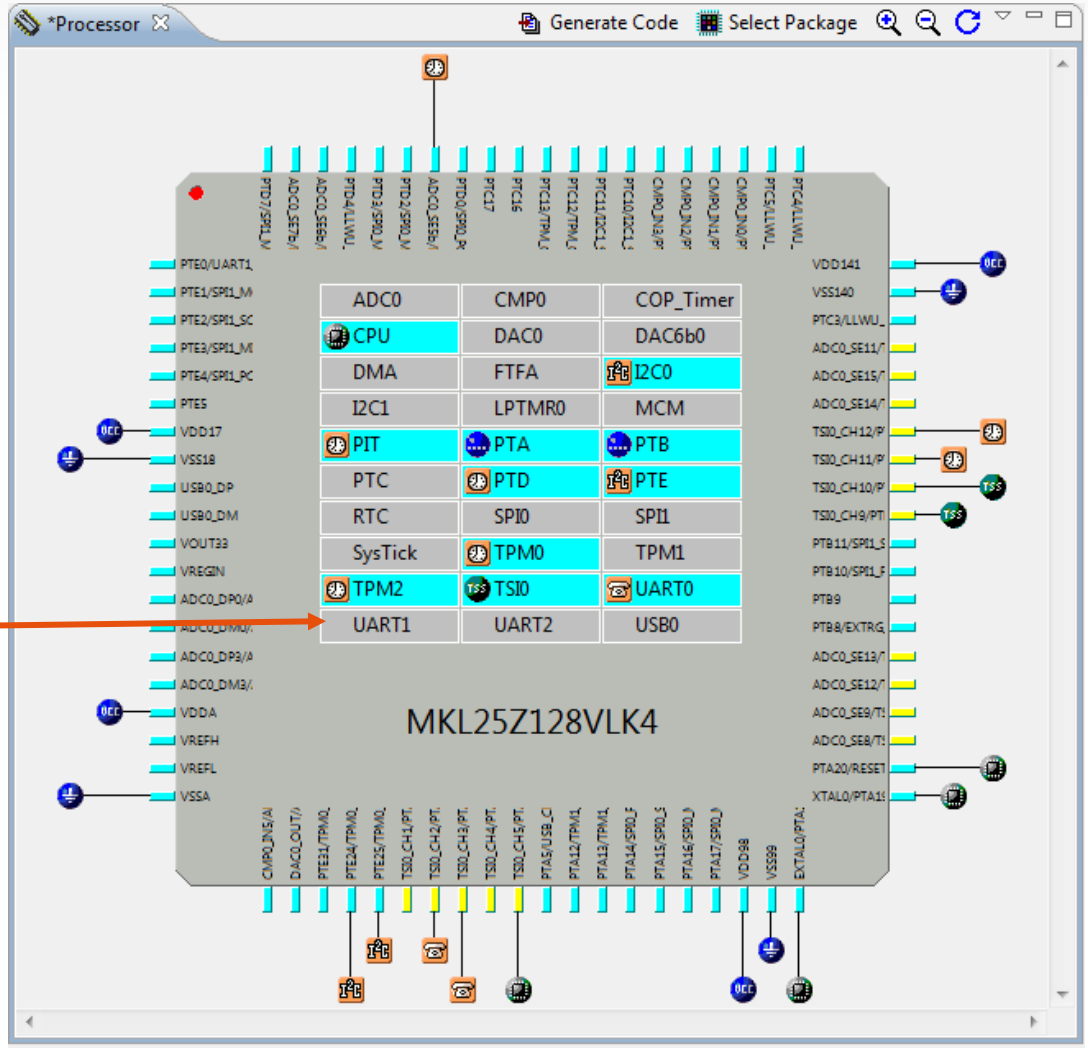


Select the UART 0

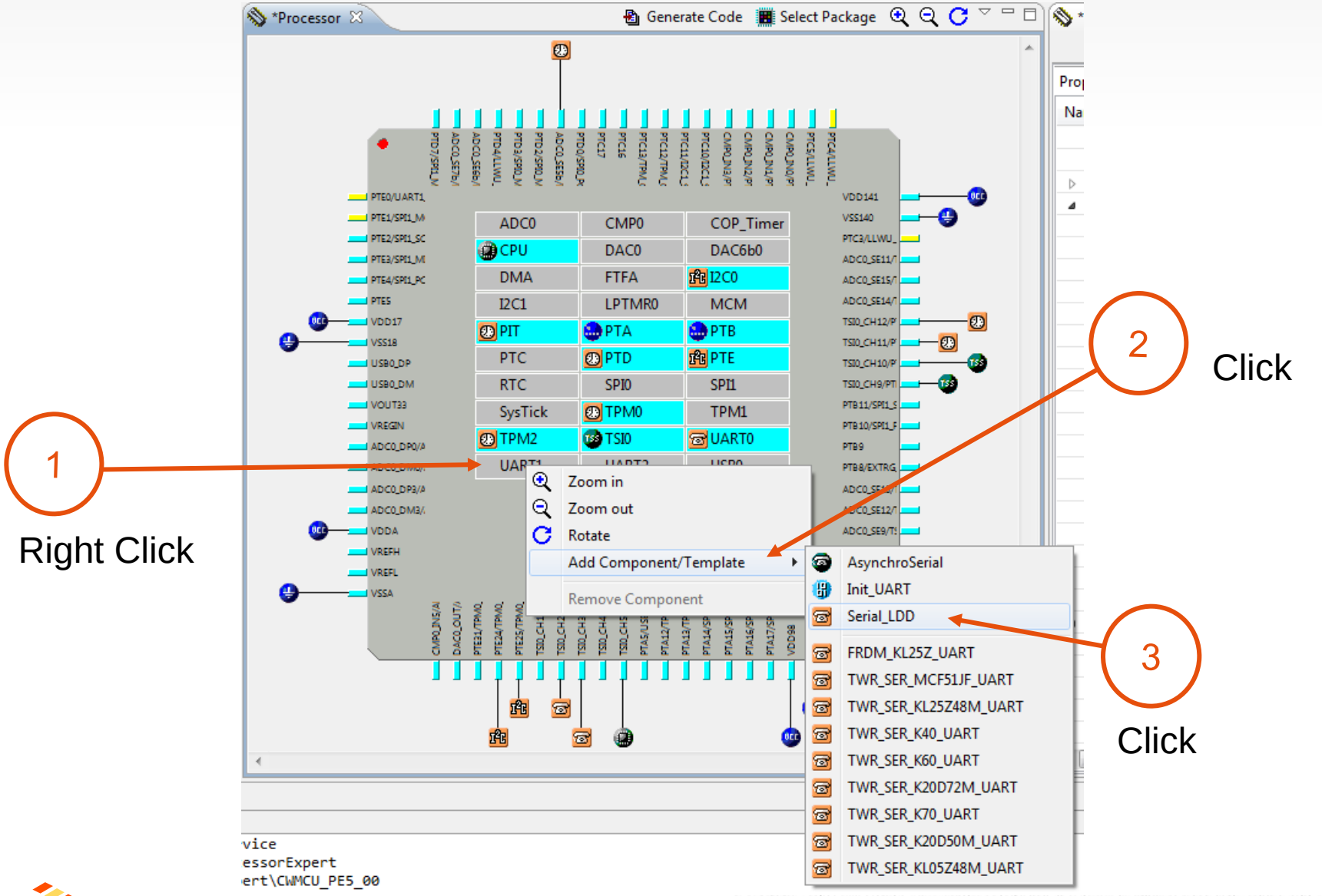


Notice these panes now reflect this component

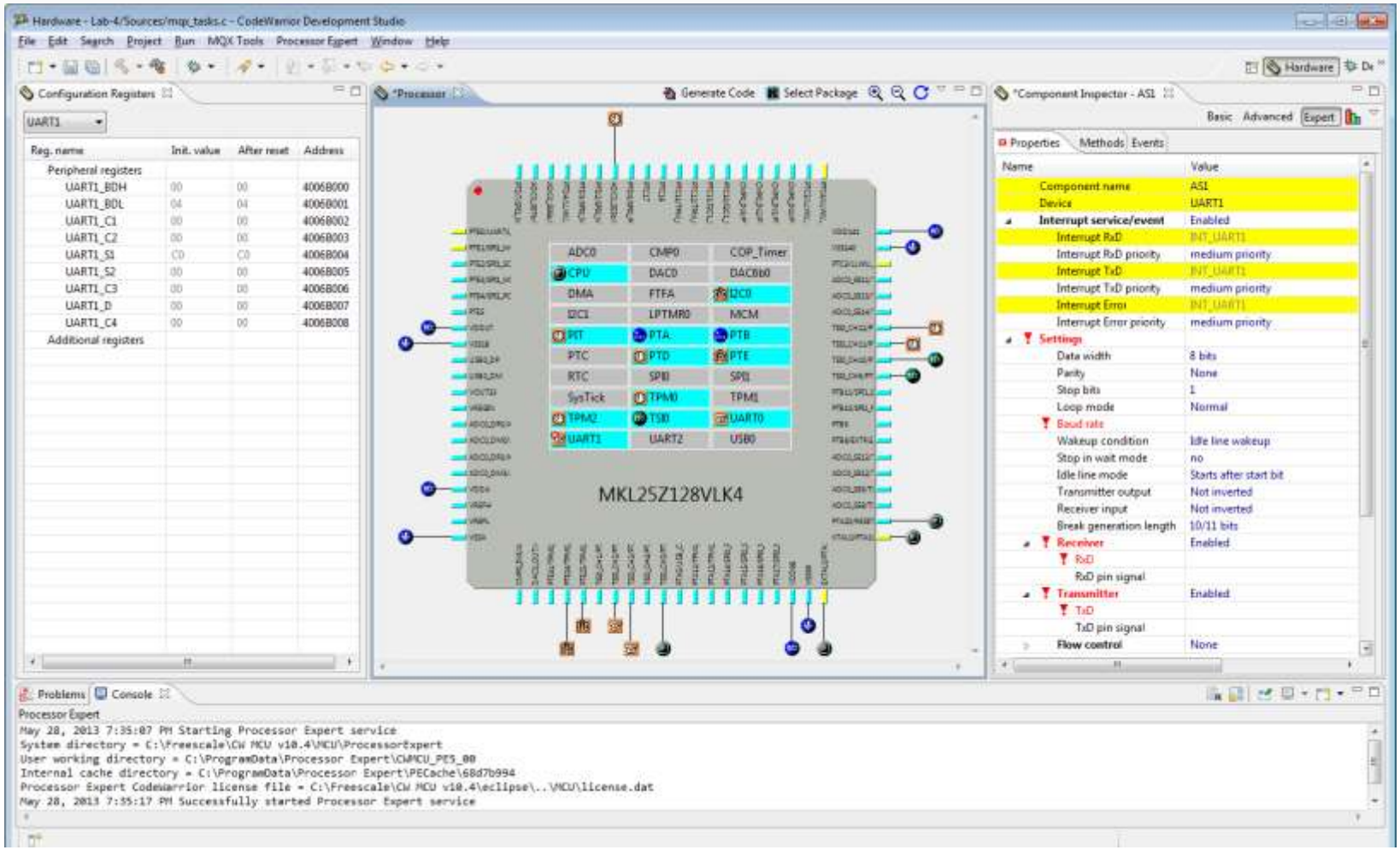
Add UART1


*Processor
Generate Code
Select Package
ADC0
CMP0
COP_Timer
CPU
DAC0
DAC6b0
DMA
FTFA
I2C0
I2C1
LPTMR0
MCM
PIT
PTA
PTB
PTC
PTD
PTE
RTC
SPI0
SPI1
SysTick
TPM0
TPM1
TPM2
TSI0
UART0
UART1
UART2
USB0
MKL25Z128VLK4
1
Right Click

Add UART1

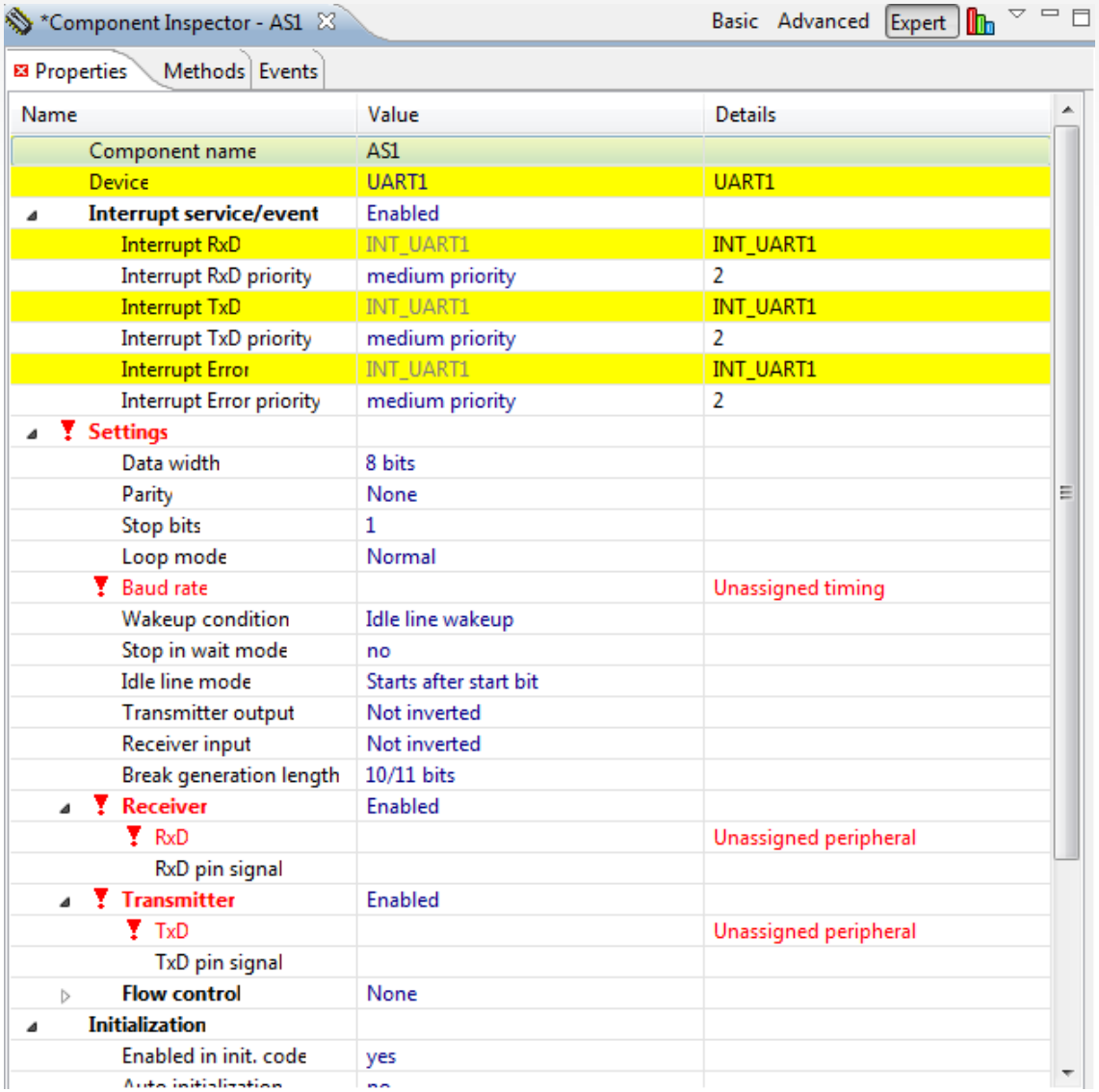

*Processor
Generate Code
Select Package
ADC0
CMP0
COP_Timer
CPU
DAC0
DAC6b0
DMA
FTFA
I2C0
I2C1
LPTMR0
MCM
PIT
PTA
PTB
PTC
PTD
PTE
RTC
SPI0
SPI1
SysTick
TPM0
TPM1
TPM2
TSI0
UART0
Zoom in
Zoom out
Rotate
Add Component/Template
Remove Component
AsynchroSerial
Init_UART
Serial_LDD
FRDM_KL25Z_UART
TWR_SER_MCF51JF_UART
TWR_SER_KL25Z48M_UART
TWR_SER_K40_UART
TWR_SER_K60_UART
TWR_SER_K20D72M_UART
TWR_SER_K70_UART
TWR_SER_K20D50M_UART
TWR_SER_KL05Z48M_UART
1
Right Click
2
Click
3
Click

Add UART1


Hardware - Lab-4/Sources/mqx_tasks.c - CodeWarrior Development Studio
File Edit Search Project Run MQX Tools Processor Expert Window Help
Configuration Registers
UART1
Reg. name
Init. value
After reset
Address
Peripheral registers
UART1_BDH
UART1_BDL
UART1_C1
UART1_C2
UART1_S1
UART1_S2
UART1_C3
UART1_D
UART1_C4
Additional registers
*Processor
Generate Code
Select Package
ADC0
CMP0
COP_Timer
CPU
DAC0
DAC0b0
DMA
FTFA
I2C0
I2C1
LPTMR0
MCM
PIT
PTA
PTB
PTC
PTD
PTE
RTC
SPI0
SPI1
SysTick
TPM0
TPM1
TPM2
TSI0
UART0
UART1
UART2
USB0
MKL25Z128VLK4
*Component Inspector - AS1
Basic
Advanced
Expert
Properties
Methods
Events
Name
Value
Component name
AS1
Device
UART1
Interrupt service/event
Enabled
Interrupt RxD
INT_UART1
Interrupt RxD priority
medium priority
Interrupt TxD
INT_UART1
Interrupt TxD priority
medium priority
Interrupt Error
INT_UART1
Interrupt Error priority
medium priority
Settings
Data width
8 bits
Parity
None
Stop bits
1
Loop mode
Normal
Baud rate
Wakeup condition
Idle line wakeup
Stop in wait mode
no
Idle line mode
Starts after start bit
Transmitter output
Not inverted
Receiver input
Not inverted
Break generation length
10/11 bits
Receiver
Enabled
RxD
RxD pin signal
Transmitter
Enabled
TxD
TxD pin signal
Flow control
None
Problems
Console
Processor Expert
May 28, 2013 7:35:07 PM Starting Processor Expert service
System directory = C:\Freescale\CW MCU v10.4\MCU\ProcessorExpert
User working directory = C:\ProgramData\Processor Expert\CWMCU_PE5_00
Internal cache directory = C:\ProgramData\Processor Expert\PECache\68d7b994
Processor Expert CodeWarrior license file = C:\Freescale\CW MCU v10.4\eclipse\..\MCU\license.dat
May 28, 2013 7:35:17 PM Successfully started Processor Expert service

Add UART1

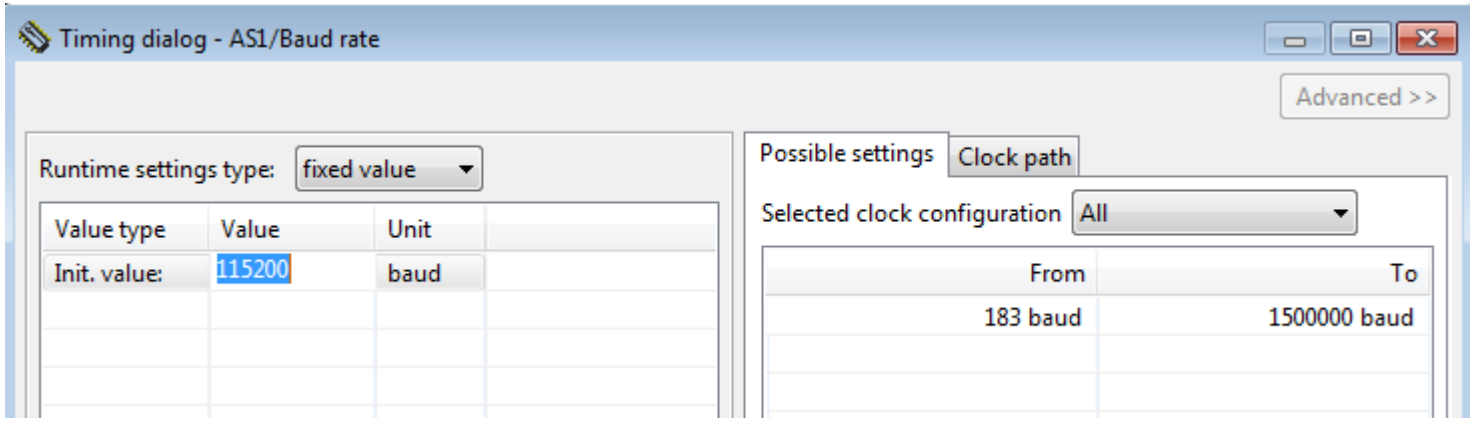
- Need Baud Rate
- Need RxD Pin
- Need TxD Pin



Select Baud Rate

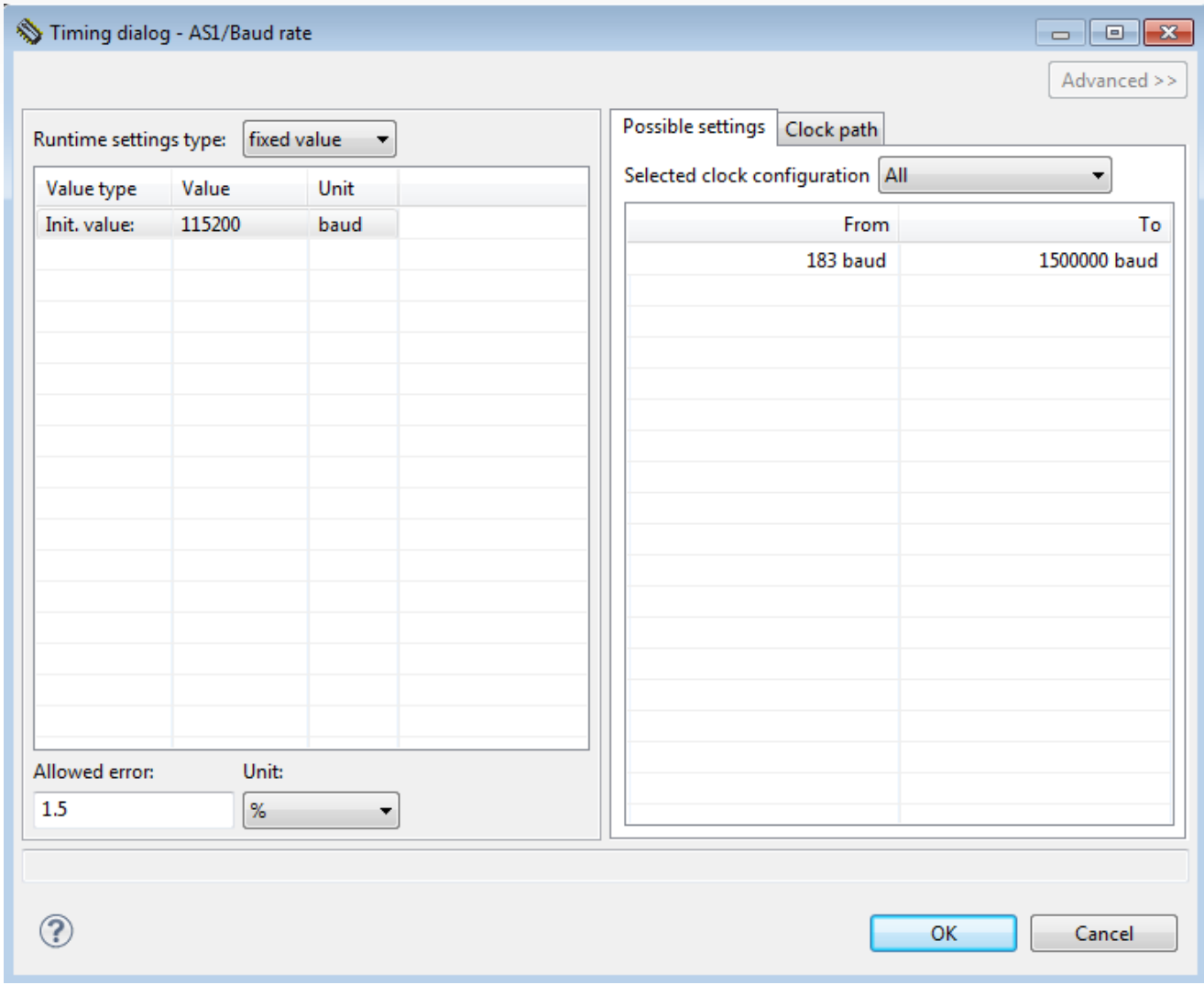
Settings		
Data width	8 bits	
Parity	None	
Stop bits	1	
Loop mode	Normal	
Baud rate	...	Unassigned timing
Wakeup condition	Idle line wakeup	
Stop in wait mode	no	
Idle line mode	Starts after start bit	
Transmitter output	Not inverted	
Receiver input	Not inverted	
Break generation length	10/11 bits	

1



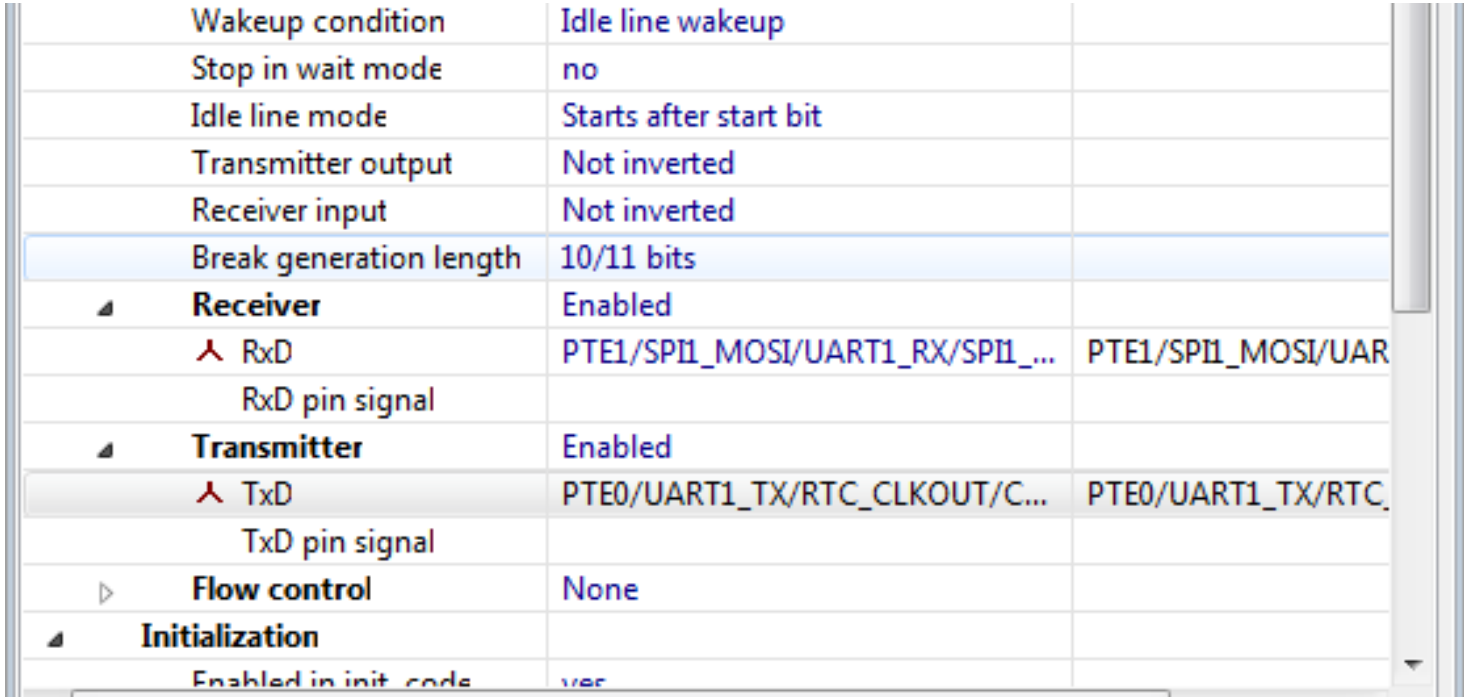
Select Baud Rate

- Enter 115200 for Value
- OK



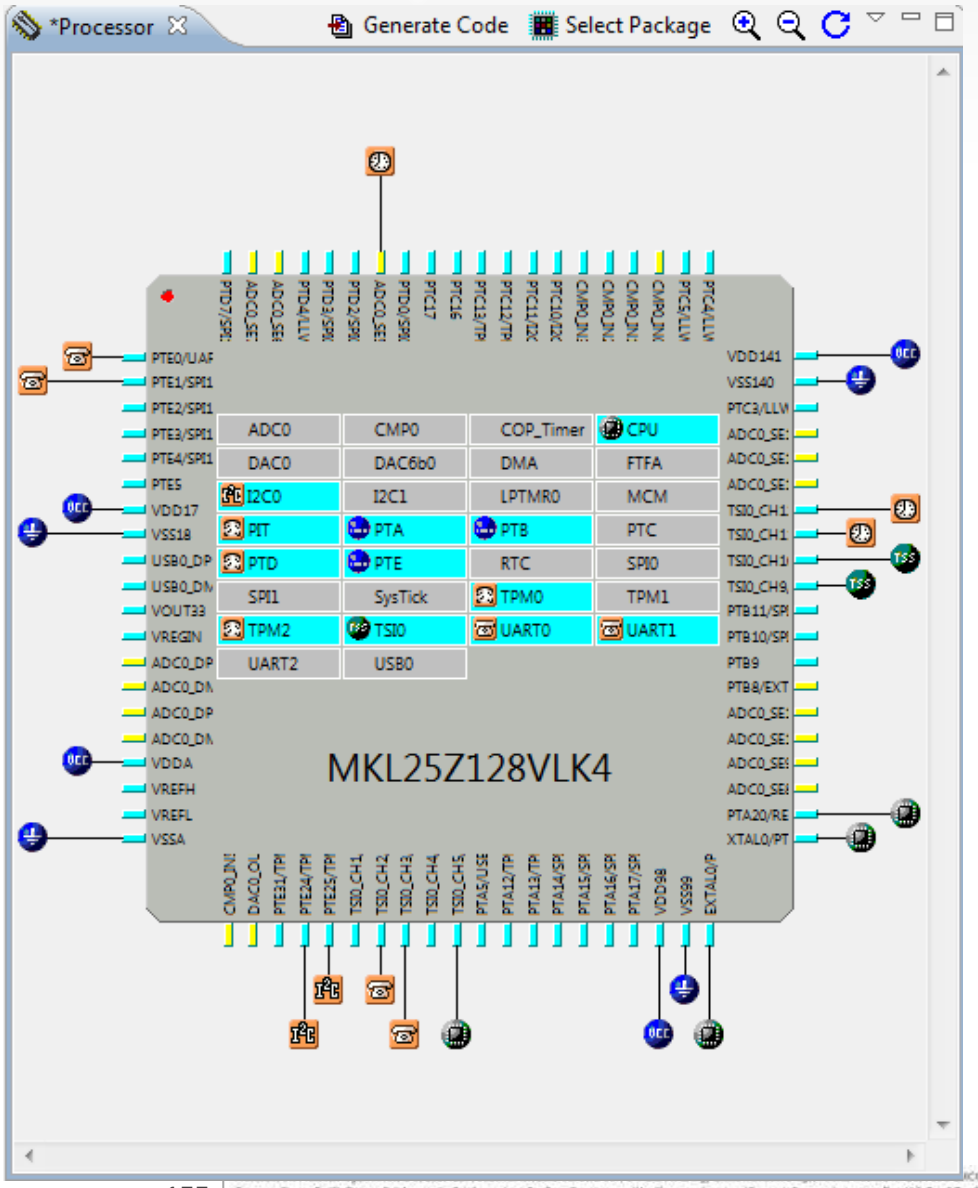
Select Rx/Tx Pins for the Functions

- Select PTE1 for RxD
- Select PTE0 for TxD



UART1 Configuration is now Complete

- Note the on the pins



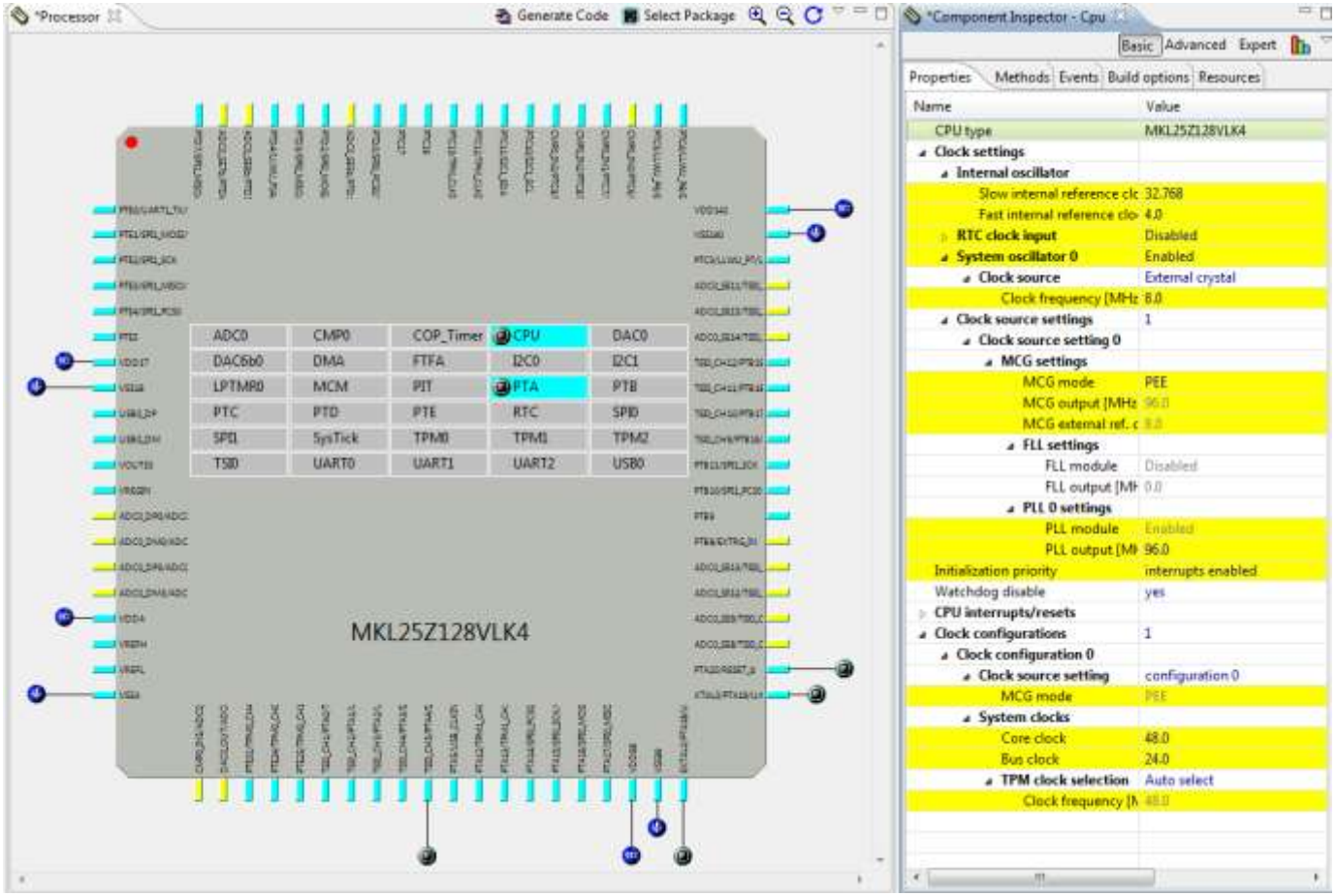
LAB 7: Create Your Own PEF

Lab 7 Generate Your Own PEB

- Create “BareBoard” Project
  - Use Either Commander View “New MCU project” or
  - File -> New -> Bareboard Project
- Select MKL25Z128 chip (as done in earlier Lab)
- Open Hardware View
- Set up the clocks, etc.
- Save the component as a PEF (remember where you put it and what you called it)
- Now repeat Lab 2 but use this PEF file for the CPU component.
- All is well!
- This could be repeated for all the components.

Lab 7 Generate Your Own PEB

- Set Clocks and Pins



Questions?

- Please fill out the survey so we can improve the class for next year. Thanks!


freescale™