Hands-on Workshop: Using CodeWarrior IDE and Processor Expert Software Modeling Tool – Part 1

AMF-ENT-T0711

Mark Ruthenbeck
Ruth Rhoades Product Manager

September 2013

Agenda

- CodeWarrior Development Studio
  - Features, Suites, Pricing, Downloads, Upgrade Policy
- Basic Processor Expert Concepts & Terms
- Lab 1: FRDM-KL46Z Board Project
  - Create a new project with the New Project Wizard
  - Import existing components and files
  - Build the project
  - Test the application's functionality

CodeWarrior Development Studio

- CodeWarrior Development Studio for Microcontrollers v10.4 integrates the development tools for ColdFire, ColdFire+, DSC, Kinetis L Series, Kinetis K Series, Qorivva, PX Series, RS08, S08 and S12Z architectures into a single product based on the Eclipse open development platform.

3

CodeWarrior Differentiating Features

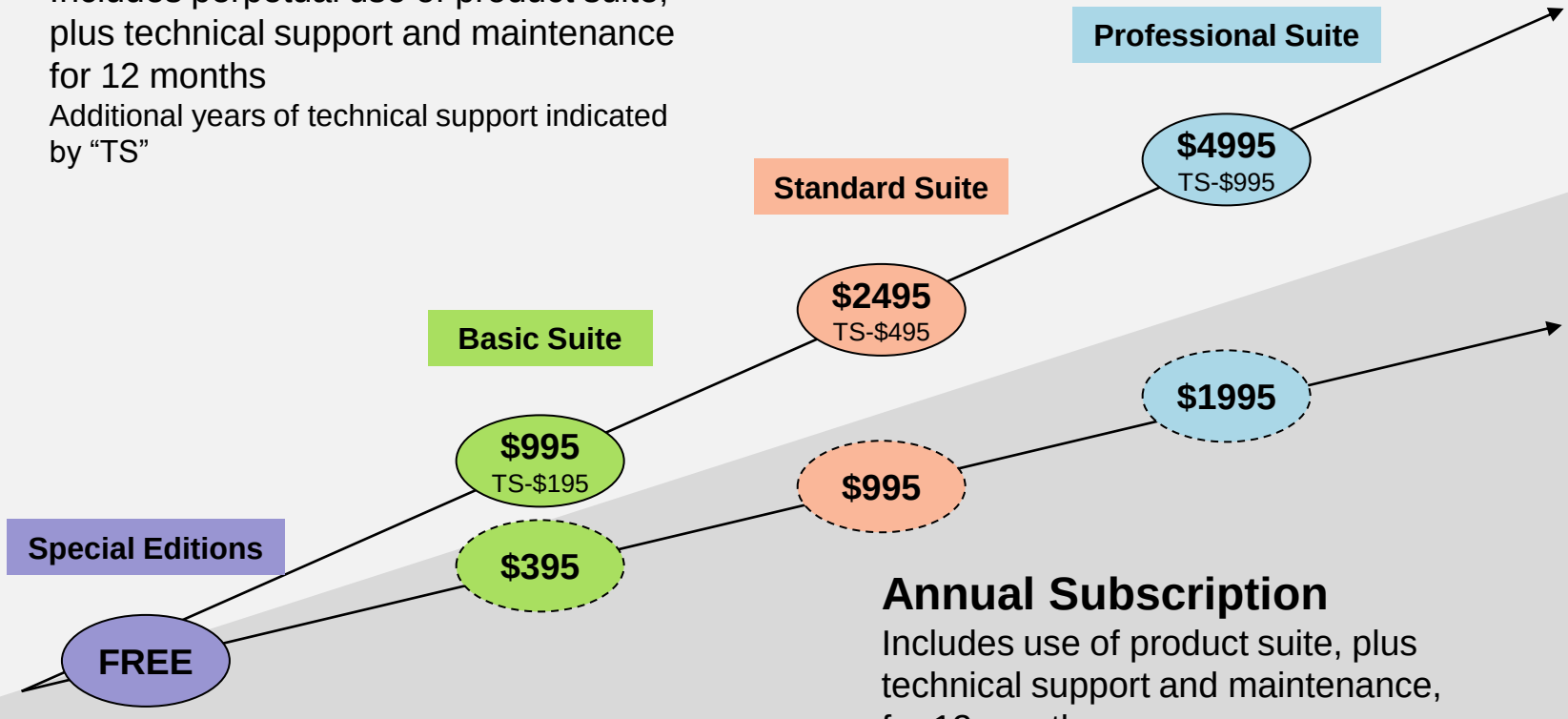
- **Special Edition** – Available at no charge and allows projects to be developed with code size limitations
- **New Project Wizard** – Allows a project to be created in as few as six clicks
- **LiveView** – Allows registers, memory and global variables to be monitored without stopping the processor
- **Processor Expert** – Creates tested, optimized initialization code and low-level drivers tuned to application needs and selected Freescale derivative
  - Built-in knowledge base immediately flags resource conflicts and incorrect settings, so errors are caught early in design cycle
  - Processor Expert for ColdFire+ and Kinetis works with and extends the capabilities of MQX
- **Trace and Profile** support for on-chip trace buffer – provides sophisticated emulator-like debug capability without additional trace capture hardware

CodeWarrior Pricing and Packaging Model

Perpetual

Includes perpetual use of product suite, plus technical support and maintenance for 12 months

Additional years of technical support indicated by “TS”



Annual Subscription

Includes use of product suite, plus technical support and maintenance, for 12 months

Feature Summary & Comparison

	Special Edition	Basic Edition	Standard Edition	Professional Edition
Key Features				
Unlimited Project Files	X	X	X	X
Unlimited Assembler	X	X	X	X
Processor Expert Software	X	X	X	X
Component Development Environment	Non-commercial	Non-commercial	Non-commercial	Commercial
Task Aware Debug				X
C++				X
Licensing: Annual or Permanent		X	X	X
1 Year Support		X	X	X
Code/Debug Size Limits				
RS08, S08	64K	128K	None	None
S12Z	64K	128K	None	None
Coldfire V1, Coldfire+	64K	128K	None	None
Coldfire V2-V4	128K	512K	None	None
DSC	64K	256K	None	None
Kinetis L	64K	128K	None	None
Kinetis K	128K	512K	None	None
Qorrivva, PX	512K	1M	None	None
Pricing				
Pricing: Annual License/Support	Free	$395	$995	$1995
Pricing: Permanent License	Free	$995	$2495	$4995
Pricing: Annual Support Permanent License	Free	$195	$495	$995

CodeWarrior Upgrade Policy

Annual Subscription License

- If you have an active annual subscription license, you can download CodeWarrior MCU v10.4 at no cost
- If your annual subscription license has expired, you must purchase another license before downloading CodeWarrior MCU v10.4
  - Basic Suite: $395
  - Standard Suite: $995
  - Professional Suite: $1,995

Valid technical support agreement equals NO cost to upgrade

Perpetual License

- If you purchased a perpetual license within the last 12 months, you can download CodeWarrior MCU v10.4 at no cost
- If you have an active technical support agreement (after the first 12 months), you can download CodeWarrior MCU v10.4 at no cost
- If your technical support agreement has expired, you must purchase another technical support agreement before downloading CodeWarrior MCU v10.4
  - Basic Suite: $195
  - Standard Suite: $495
  - Professional Suite: $995

CodeWarrior License Registration/Renewal

A new license is required for CW MCU v10.4. Licenses for previous versions will NOT work.

- **New purchase:** If you just purchased a CodeWarrior Suite, the process to activate and license your product is simple:
  - Log in to “My Freescale”
  - Select “ CodeWarrior Licensing”
  - Navigate to your product in “Products you are entitled to license”
  - Click the “Get License” button to retrieve your license
- **Existing customer:** If you already own a CodeWarrior suite and have obtained previous versions of CodeWarrior for Microcontrollers (e.g., 10.0, 10.1, 10.2 or 10.3), the process is a little different:
  - Log in to “My Freescale”
  - Select “ CodeWarrior Licensing”
  - Navigate to your product in “Products that you have licensed”
  - Click the “Version Renewal” button next to your existing CodeWarrior for Microcontrollers product to retrieve your license

Processor Expert Software

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)

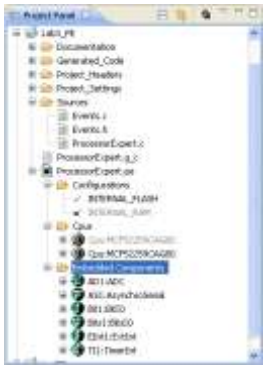
Embedded Components

- 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

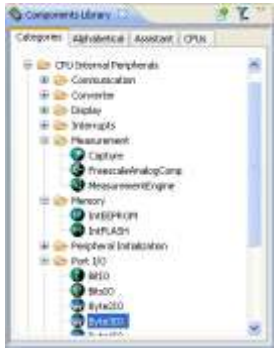


Generate Code
- Let Processor Expert generate all components drivers code

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

Processor Expert Project Design Timeline

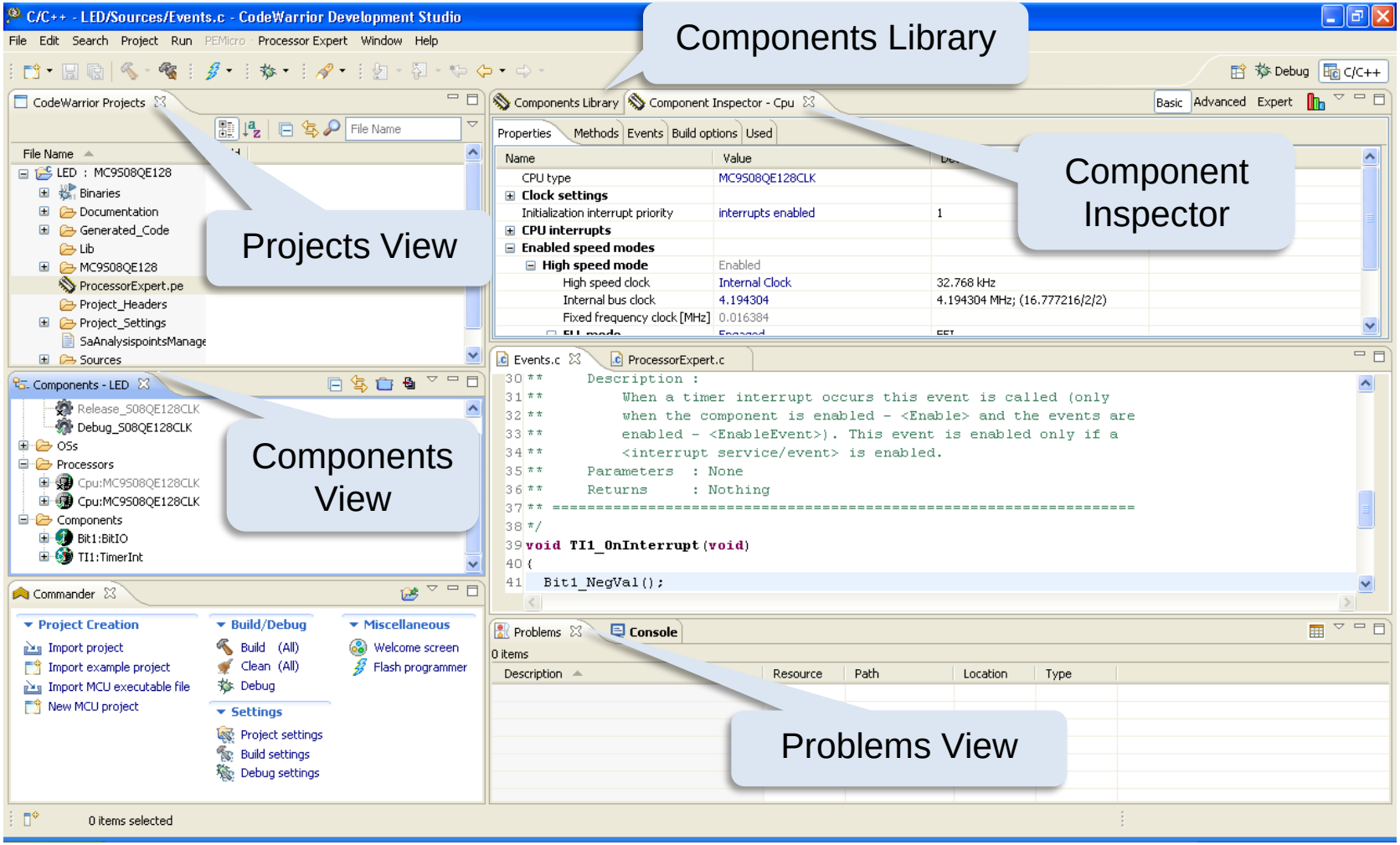
Add Components
- Select components required in the project from Components Library



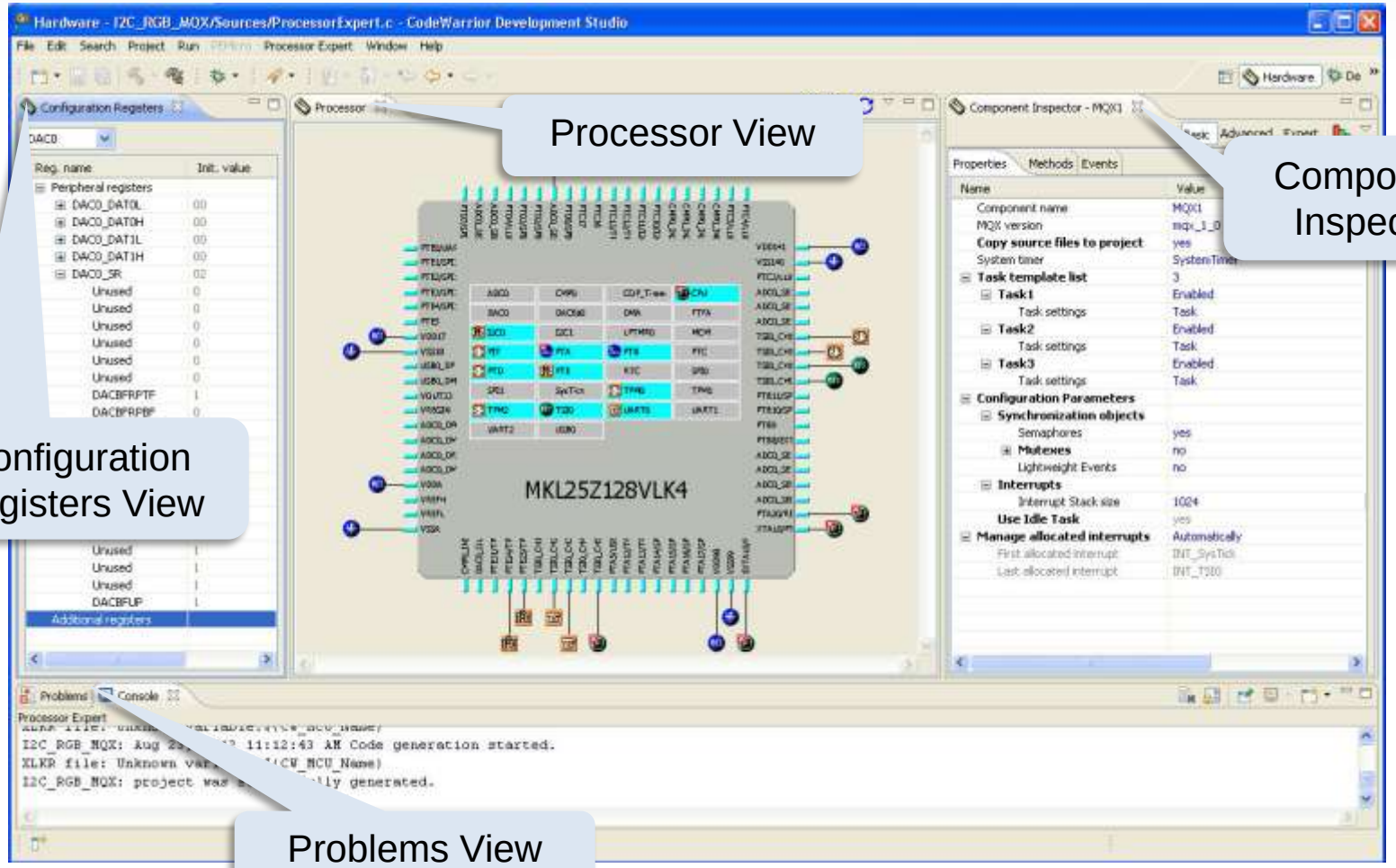
Verify Settings
- Make sure there are no design-time errors in the project

Write Code
- Write application code using code generated for components

Processor Expert Views – C/C++ Perspective

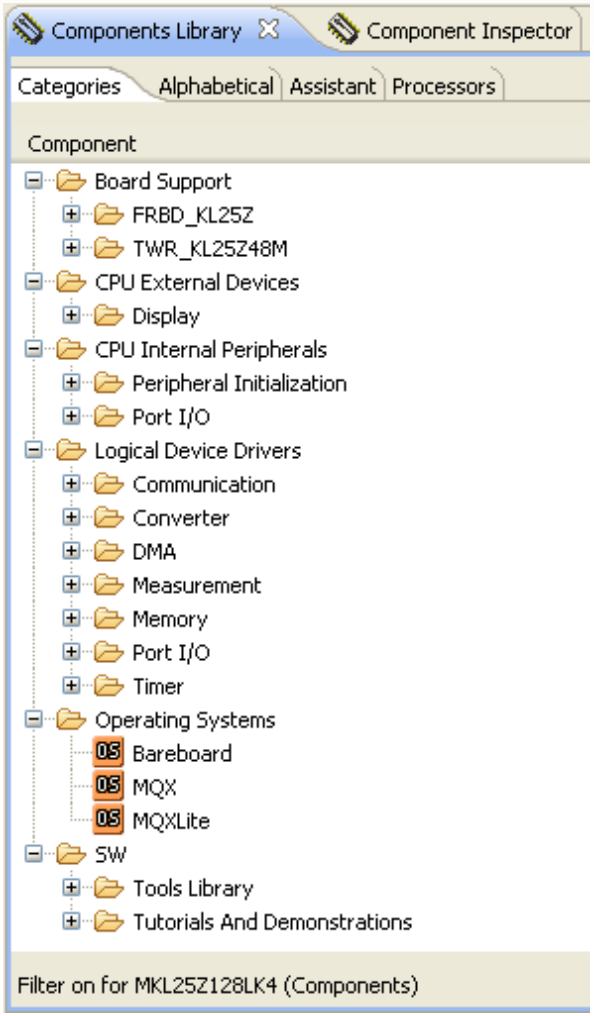


Processor Expert Views – Hardware Perspective

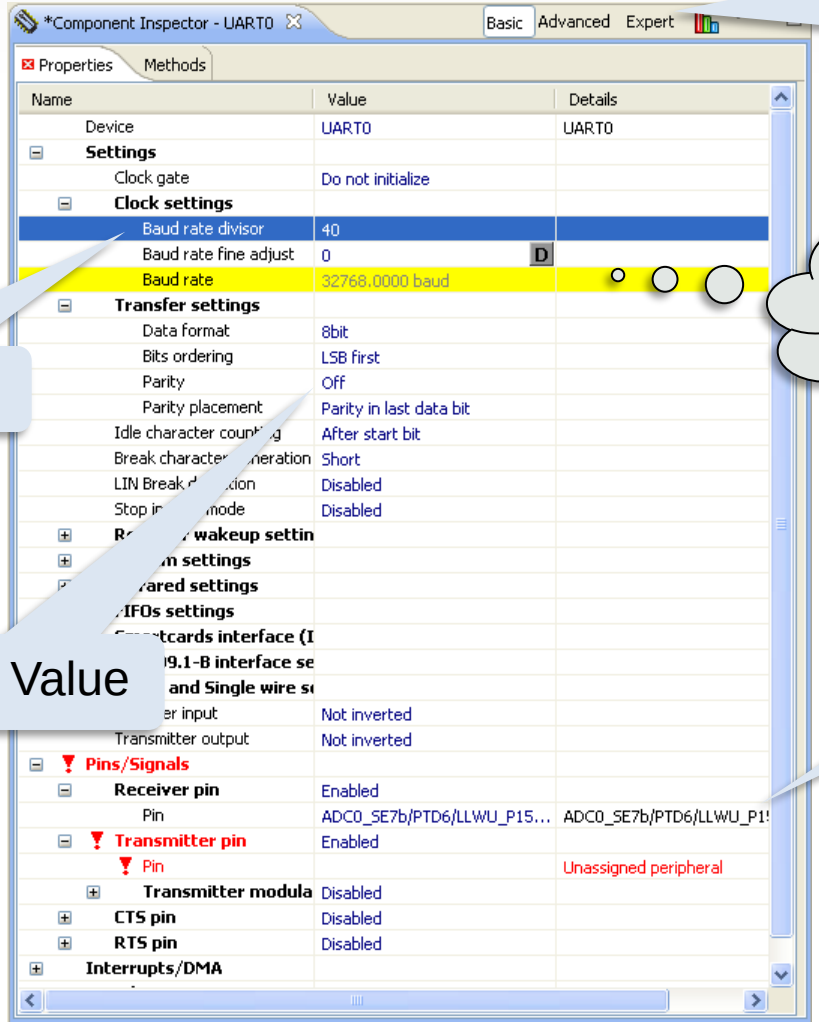


Components Library

- 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



Select level of information displayed

Recently changed items are highlighted

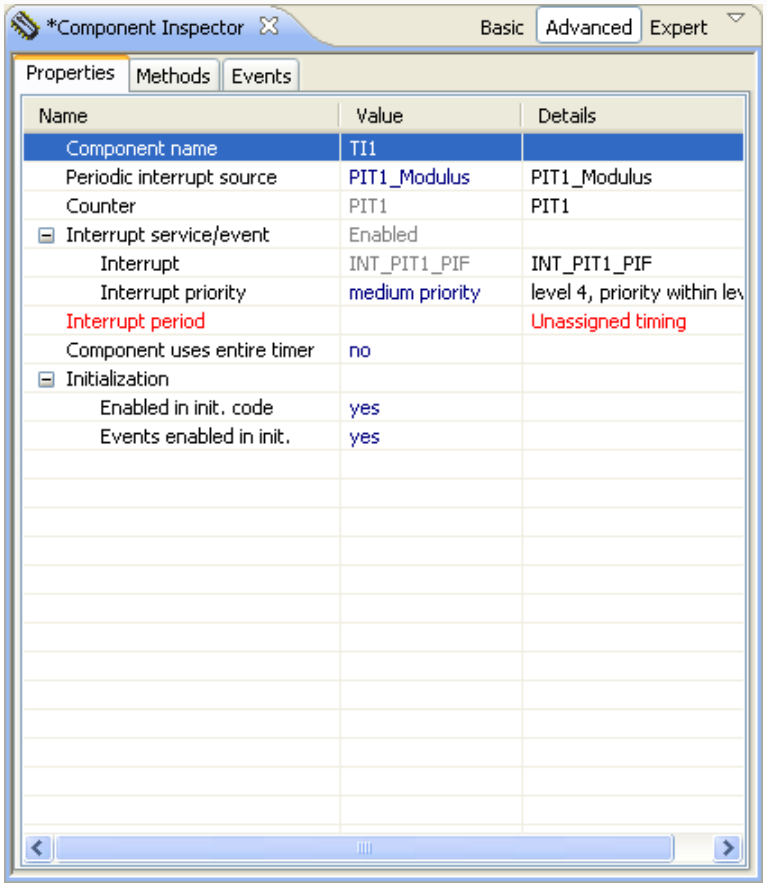
Properties

Property Value

Details

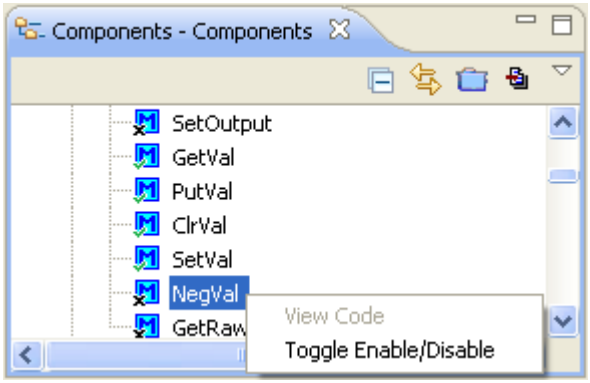
Component Inspector – Properties

- 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.

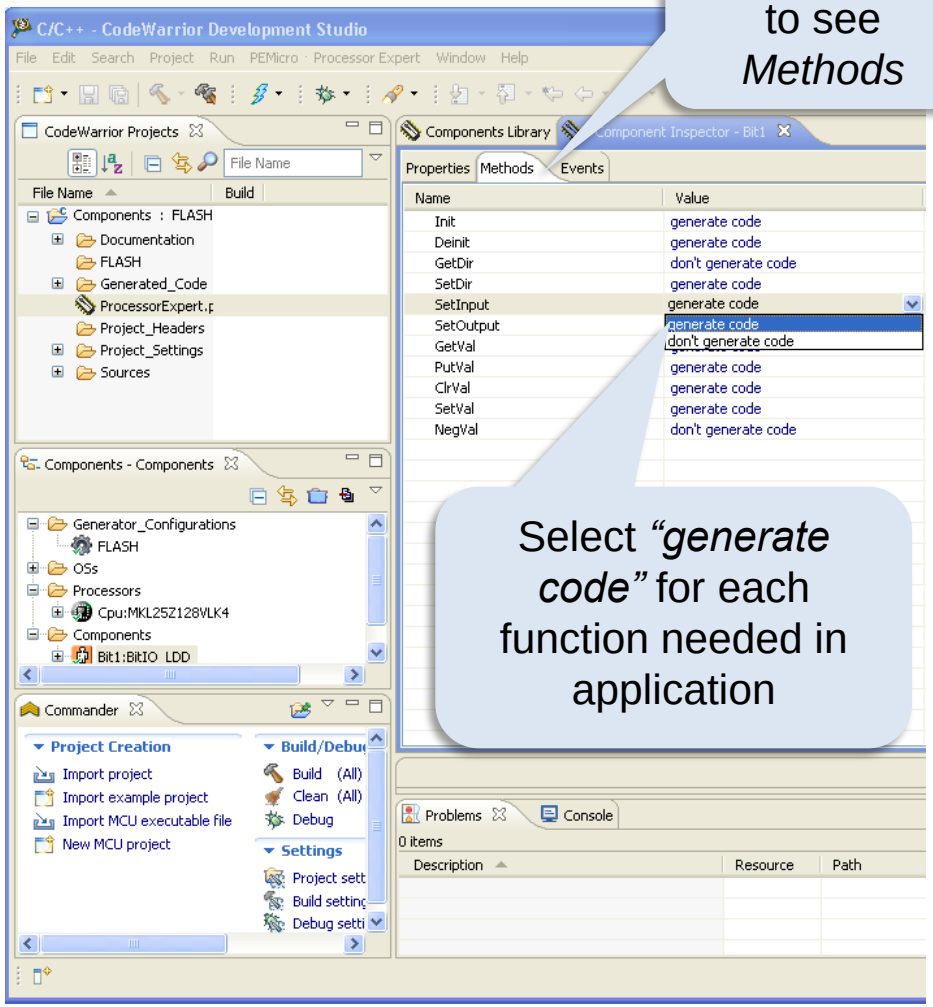


Component Inspector – Methods

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

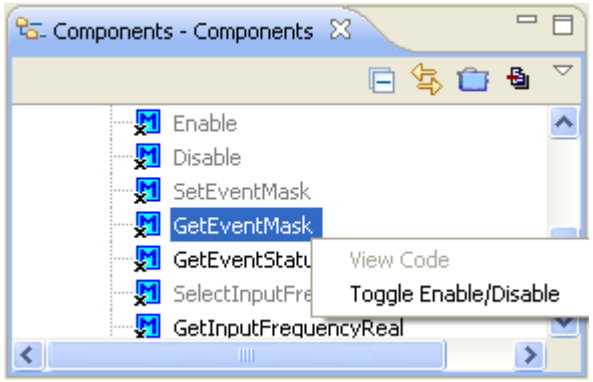


- To enable code generation for a method in the Components View, right-click on the method.
- Select *“Toggle Enable/Disable”*

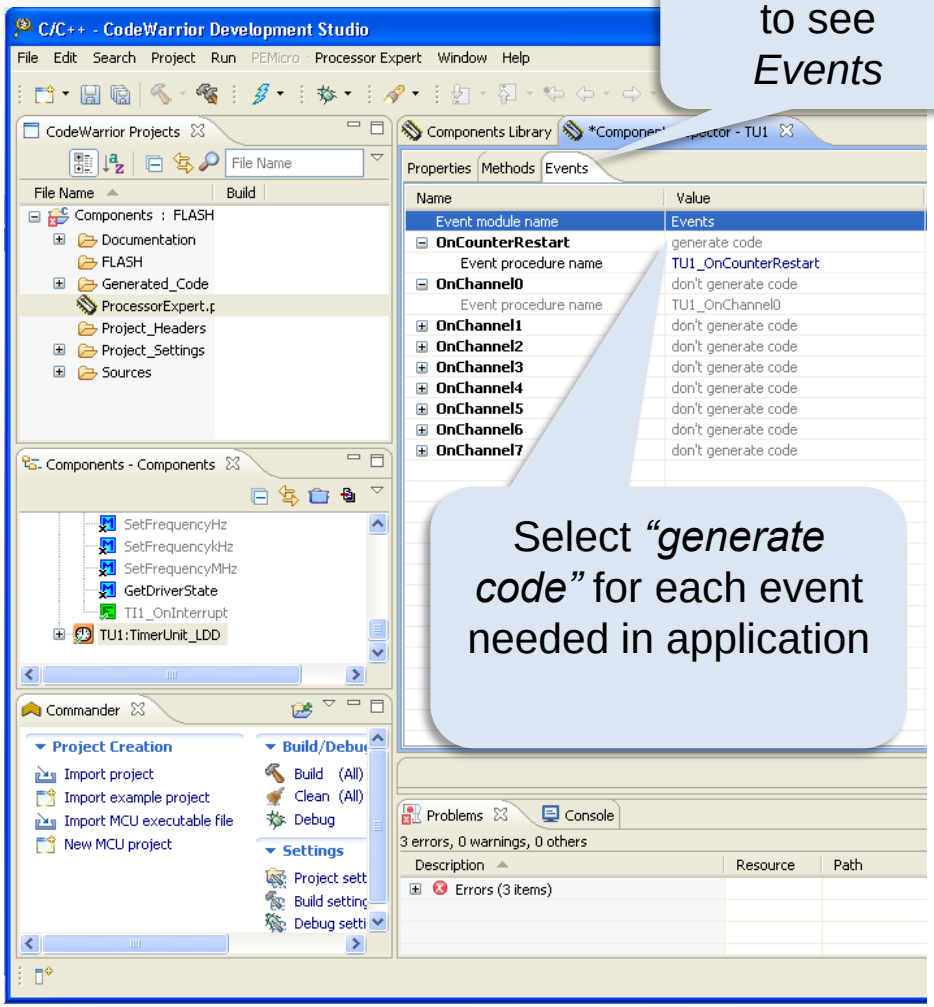


Component Inspector – Events

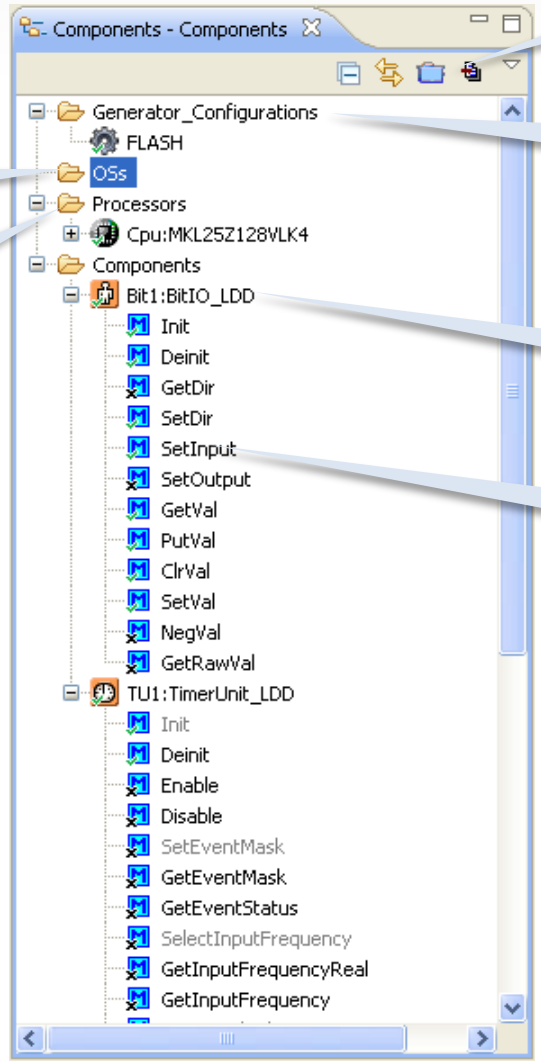
- Select the *Events TAB*
- Select “generate code” for each event required in the application
- Code will only be generated for the events selected



- 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

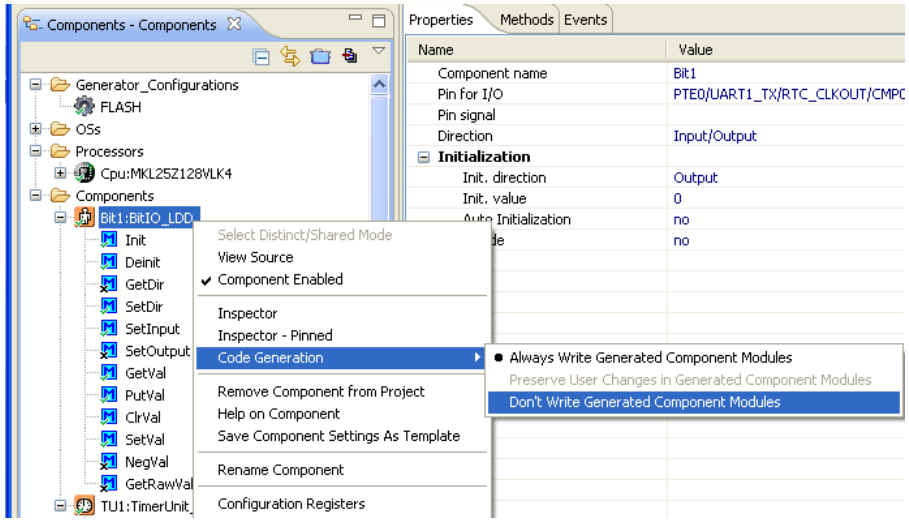
Components

Processors

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*

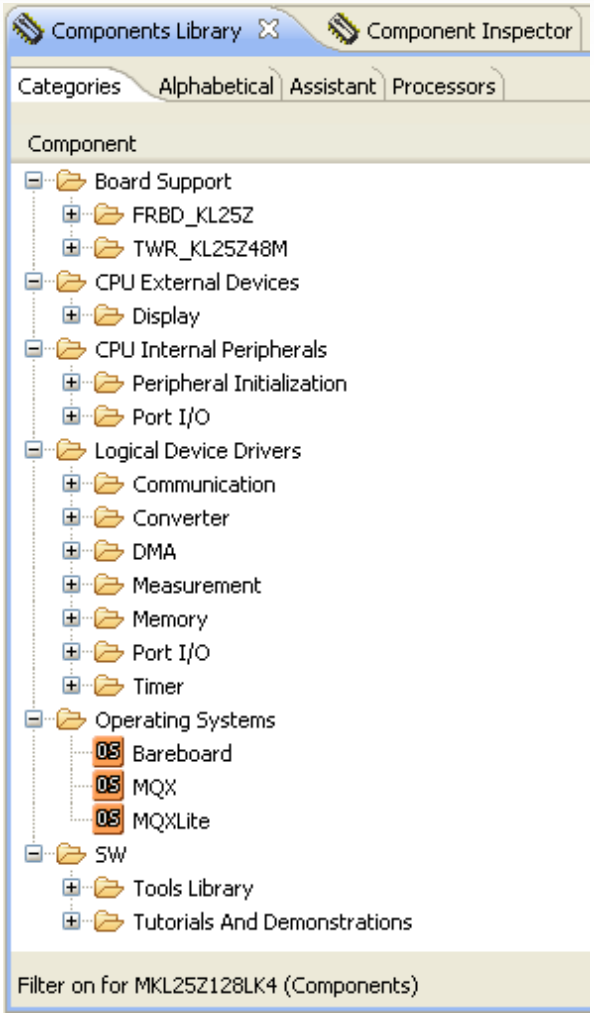


Processor Expert Software

Embedded Components

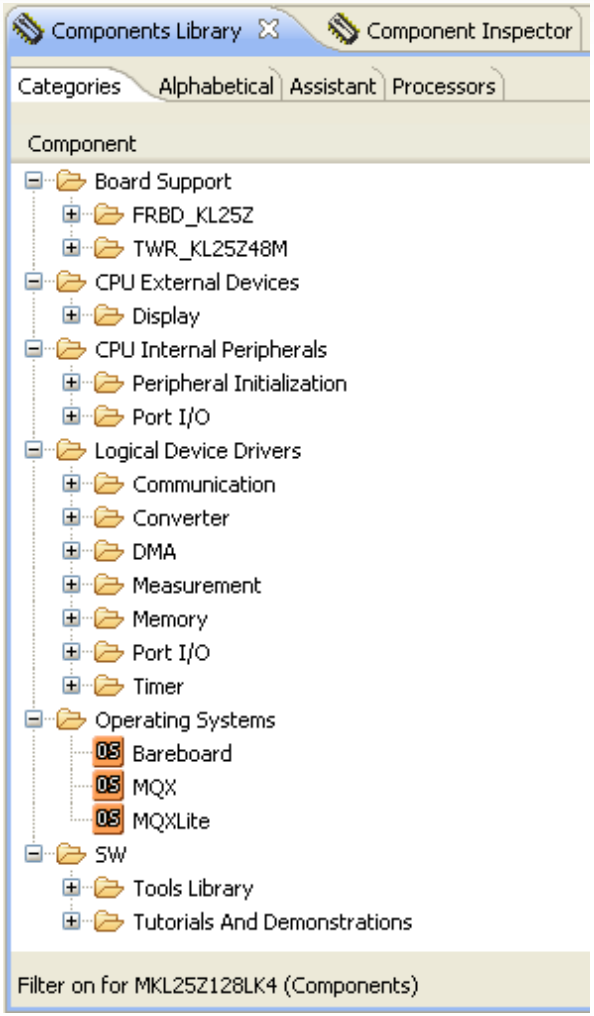
Component Categories

- **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

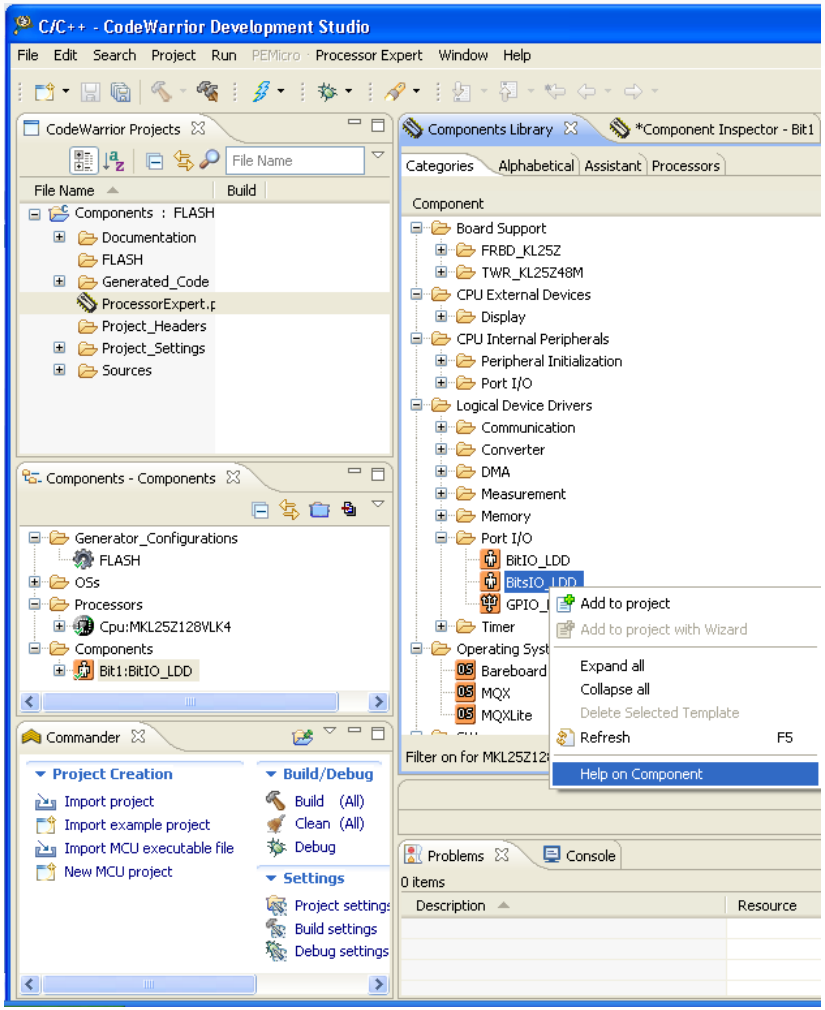


Component Categories

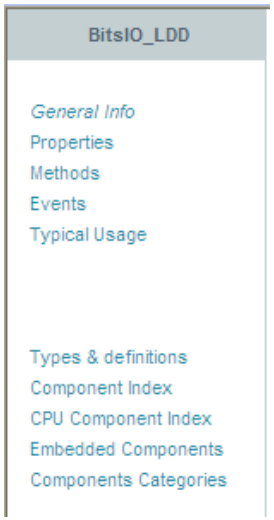
- **Logical Device Drivers**
  - Embedded Component support for internal peripherals
  - Uses a standard API, which allows generated device drivers to be adapted for used with bare-metal applications as well as RTOS applications.
- **Operating Systems**
  - RTOS adapter components
  - Adapter defines 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



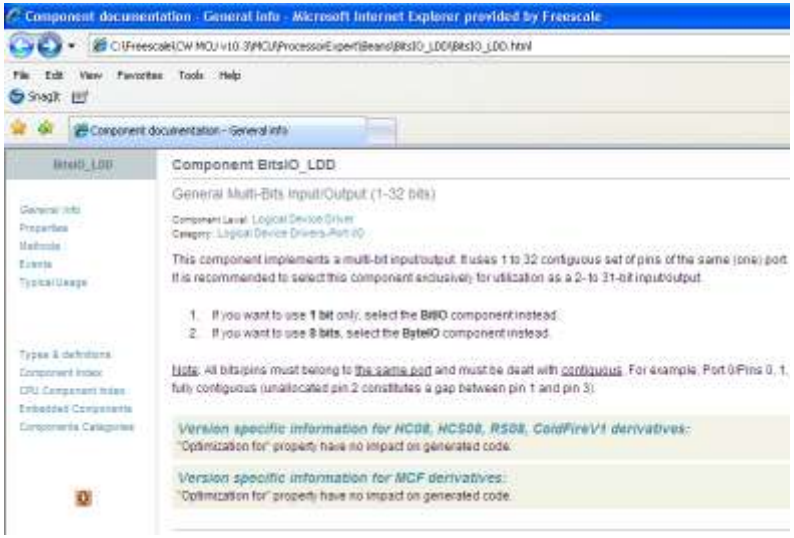
Help on Component

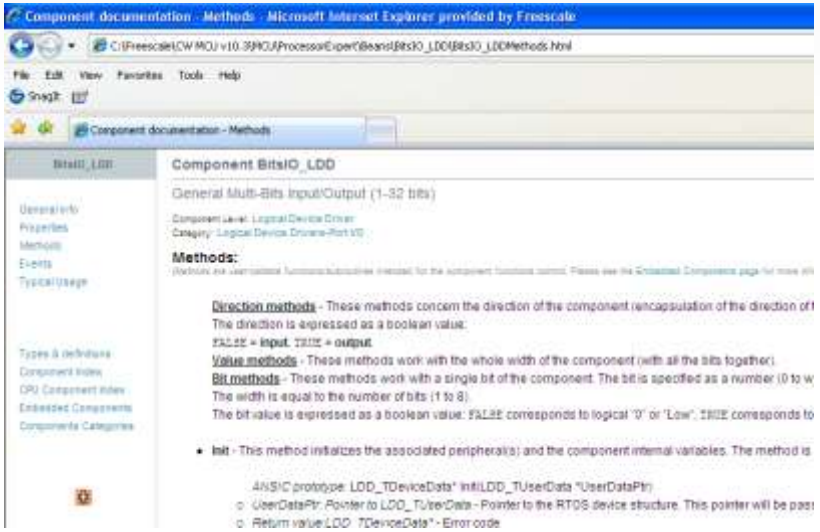


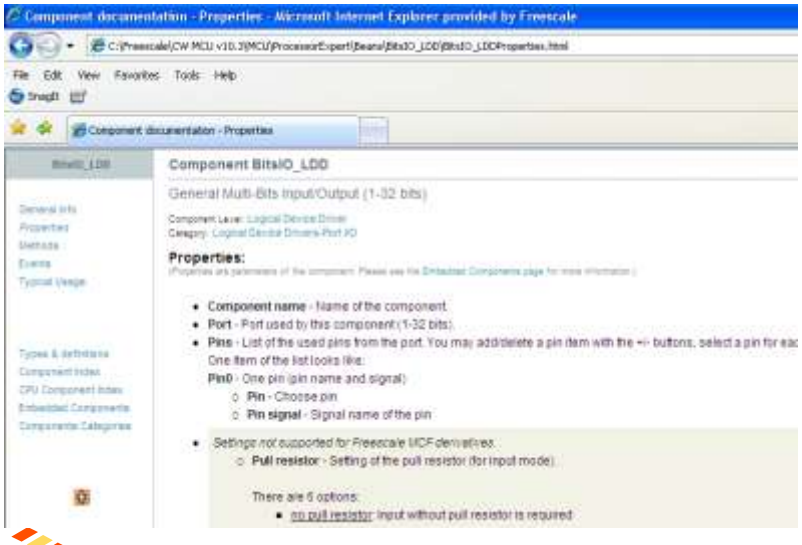
- Right-click on the Component in the *Components Library*
- Select *Help on Component* in the pop-up menu
- Component documentation will be displayed with the following topics
  - General Info
  - Properties
  - Methods
  - Events
  - Typical Usage

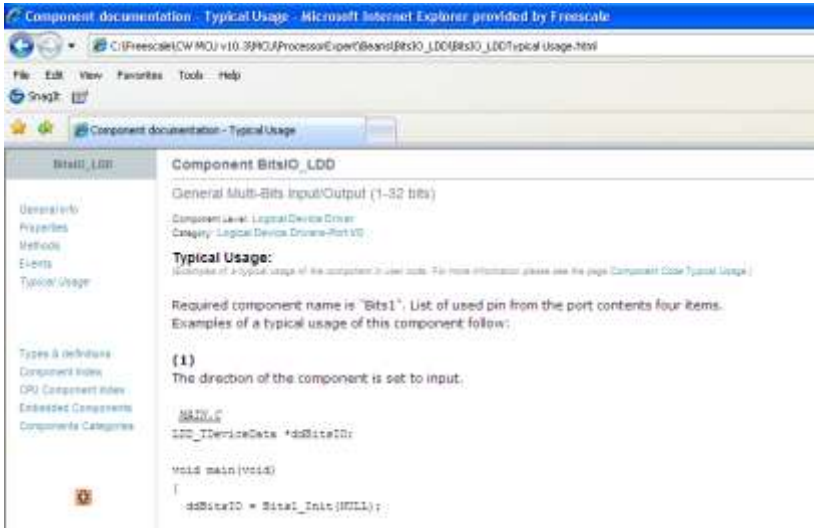


Help on Component





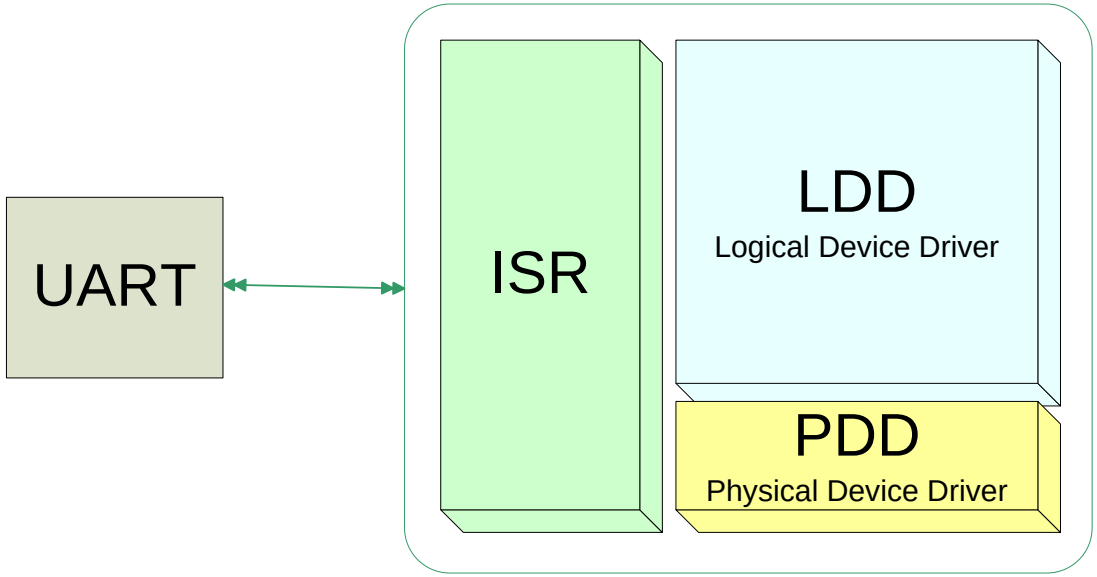




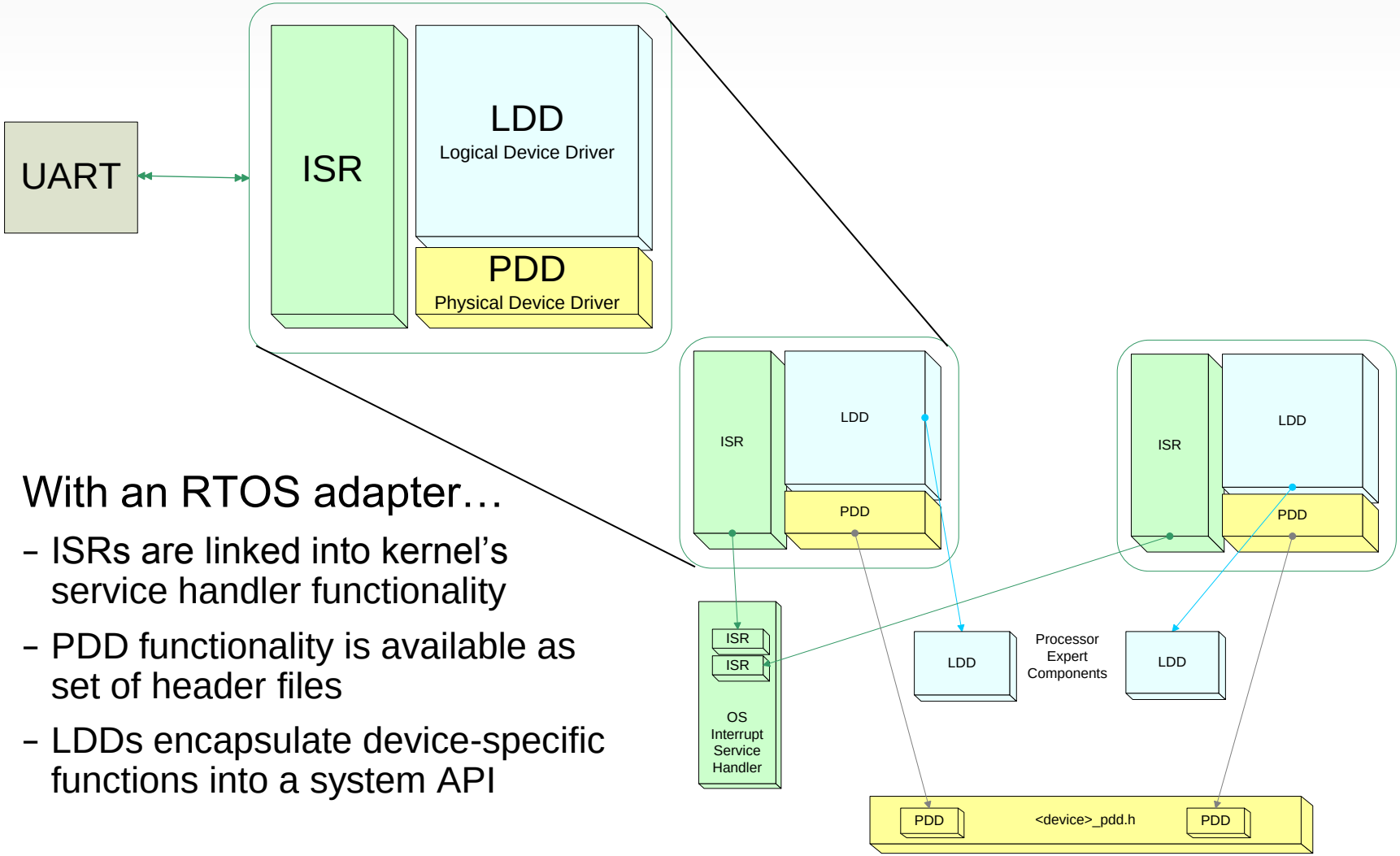
High Level and Logical Device Driver Components

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 (HLC)

- 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 HLC 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

Lab 0.5: Readying FRDM-KL46Z Board for Debugging

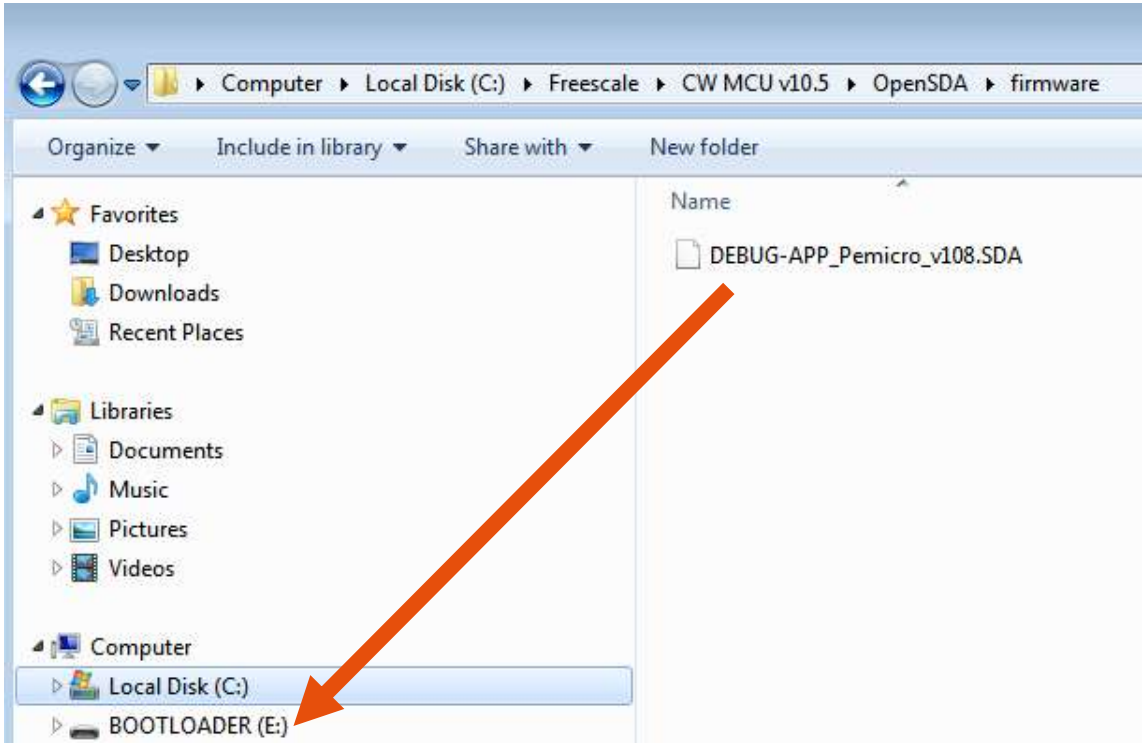
Readying FRDM-KL46Z Board

- Unplug the USB cable
- Press and hold the Reset button
- Plug in the USB Cable

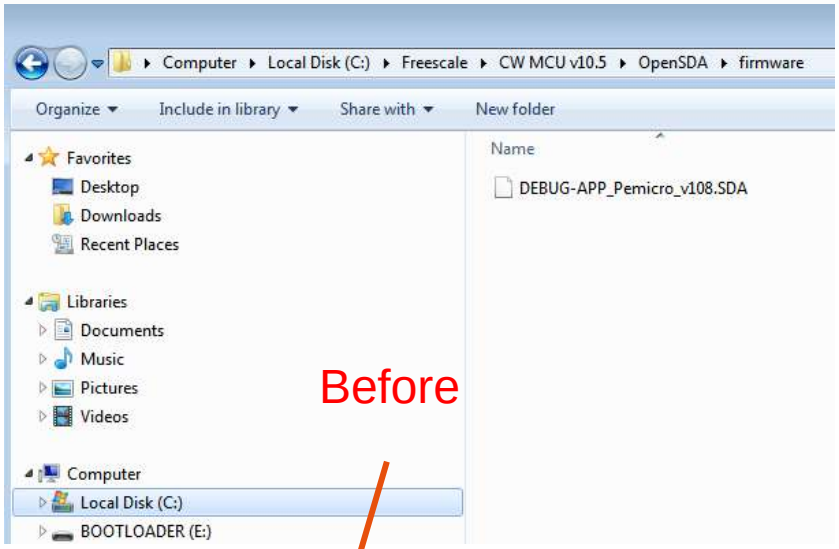
- Release the Reset button
- Green LED Blinking
- A removable drive should now be visible in the Windows Explorer with a volume label of BOOTLOADER

Readying FRDM-KL46Z Board

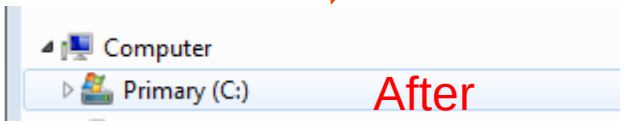
- Drag & Drop the DEBUG-APP-Pemicro_v10x.SDA to the BOOTLOADER
- C:\Freescale\CW MCU v10.5\OpenSDA\firmware



Readying FRDM-KL46Z Board



Before



After

- You will now see the “debug app” on the drive.
- Unplug the USB
- Re-Plug the USB
- BOOTLOADER Disappears
- Check for “solid” Green LED
- You are ready to go!

Green LED ON

Readying FRDM-KL46Z Board

- Check for “solid” Green LED
- You are ready to go!

Lab 1: Use New Project Wizard to Create Project

Create a new project to blink the LEDs

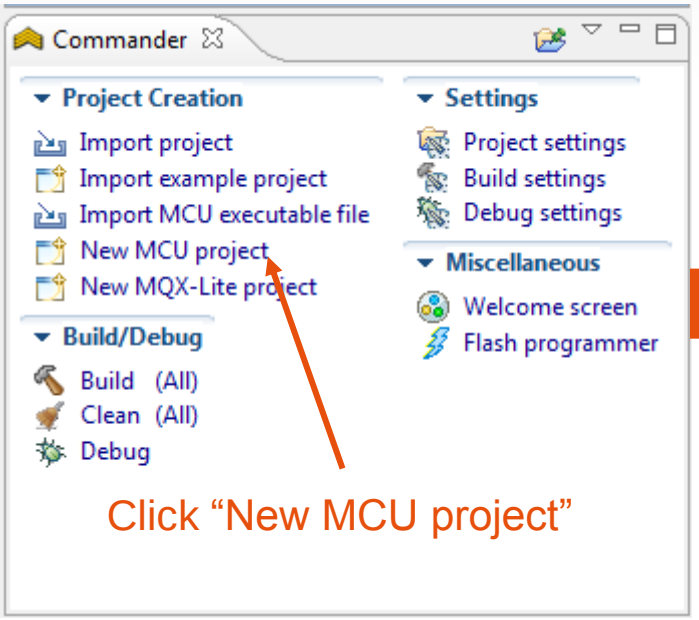
- This hands-on lab shows you how to…
  - Create a new project with the New Project Wizard
  - Configure Components with the Component Inspector
  - Use High Level Device functions
  - Import existing files
  - Build the project
  - Test the application's functionality
  - Trace the Code
- The lab uses the FRDM-KL46Z board
- The application will blink an LED periodically, and light a LED with button presses.

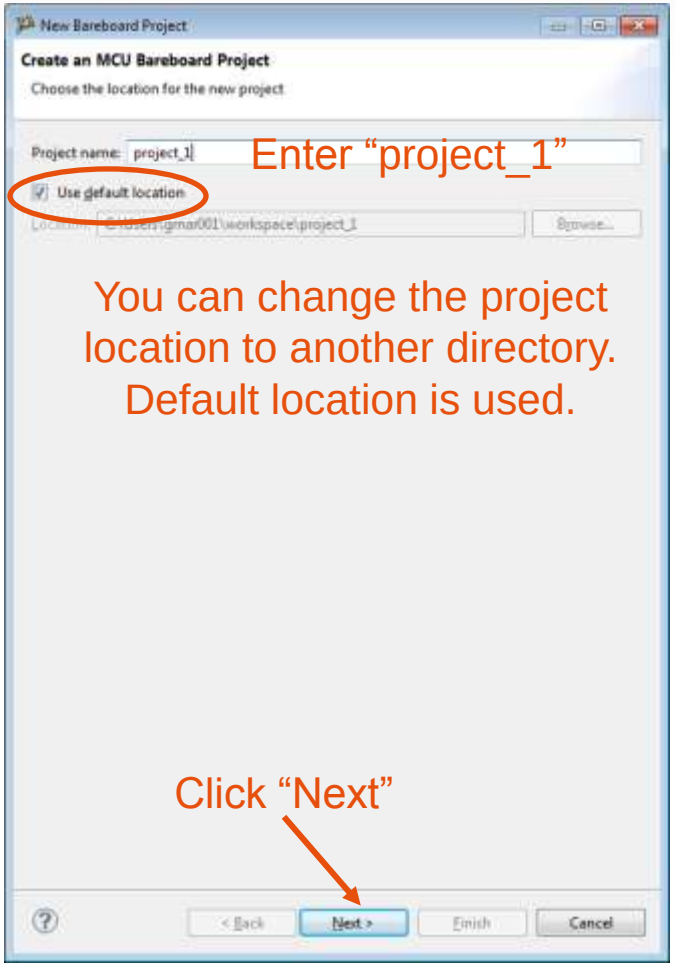
Check Point: Create a New Project to Blink an LED

- This hands-on lab shows you how to…
  - **Create a new project with the New Project Wizard**
  - Select & Configure Components
  - Configure Components with the Component Inspector
  - Use High Level Device functions
  - Import existing components
  - Build the project
  - Test the application's functionality
  - Trace the Code

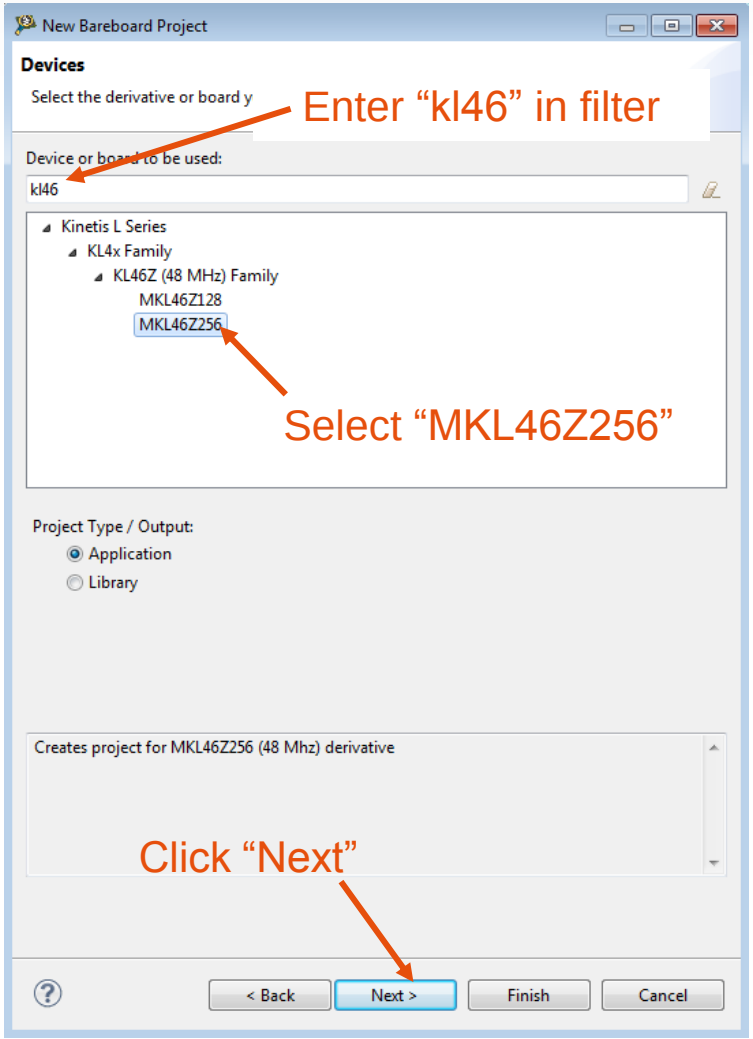
Next up!

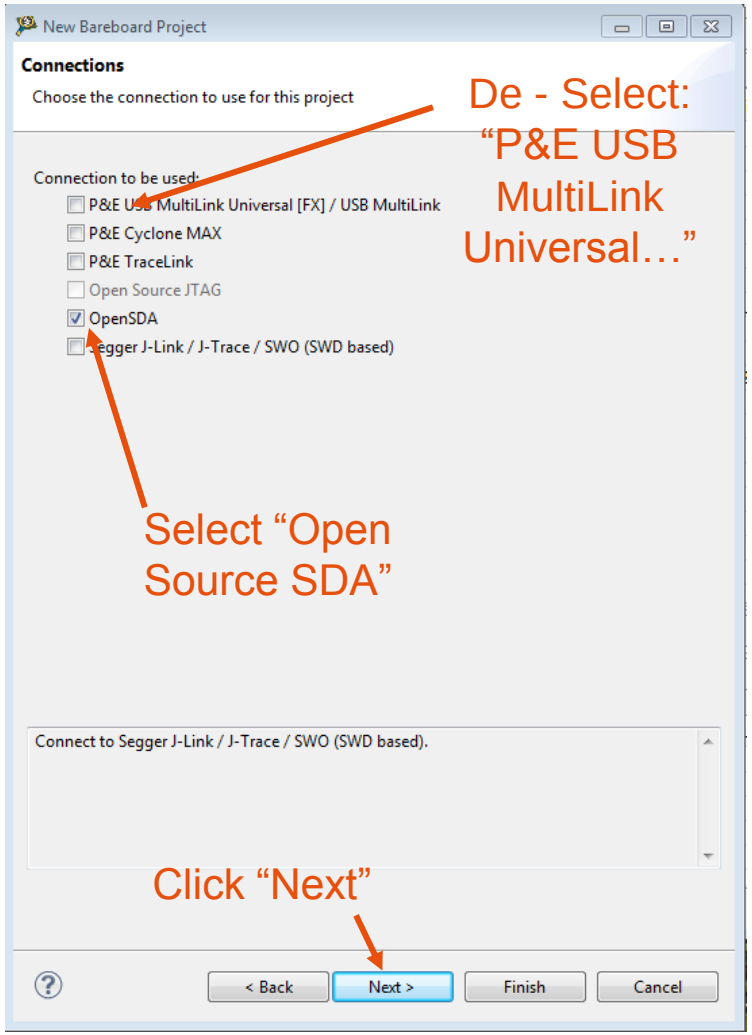
Open New Project Wizard & Select Project Name



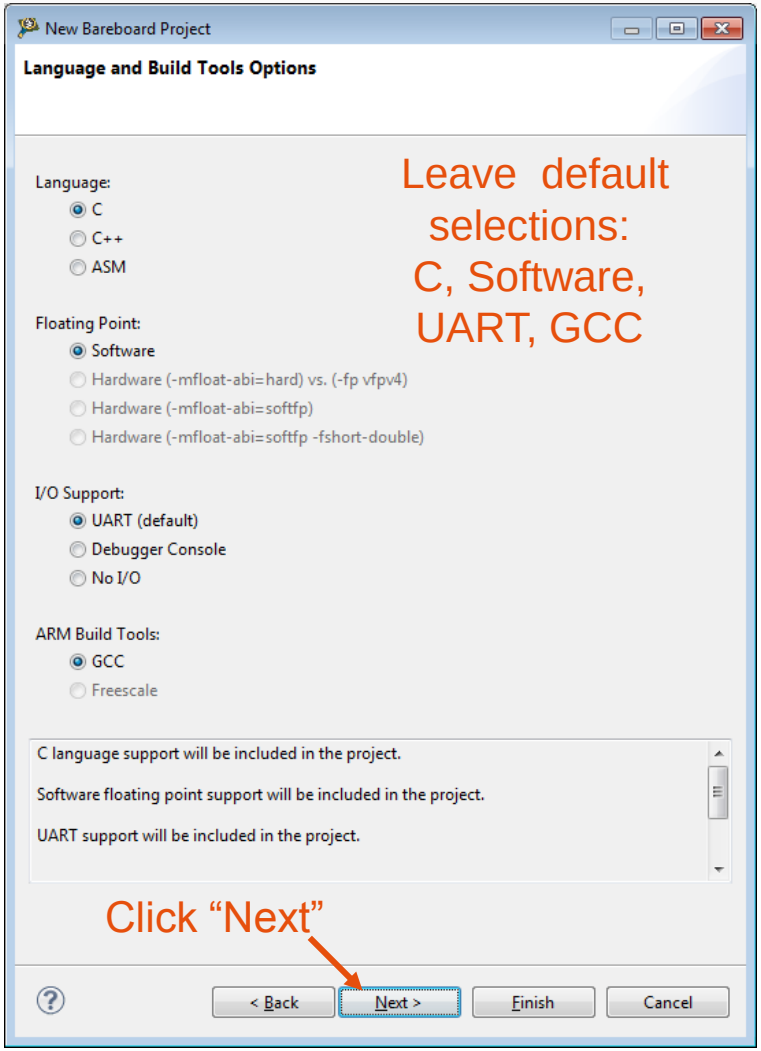


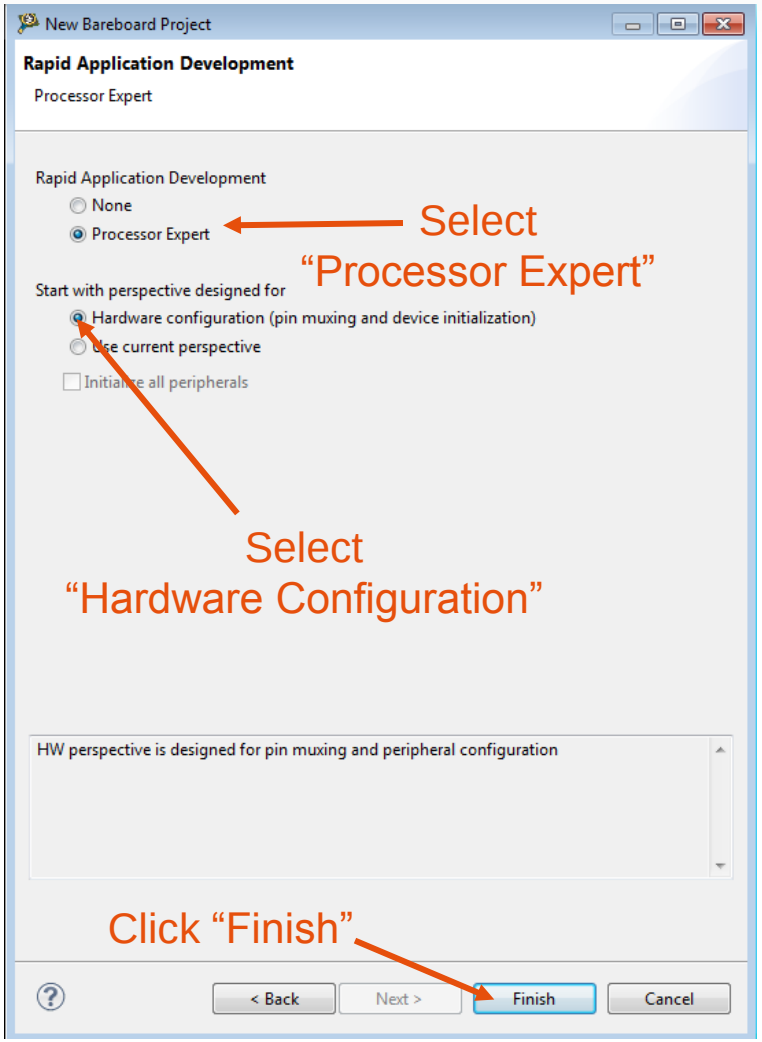
Select Device and Connection



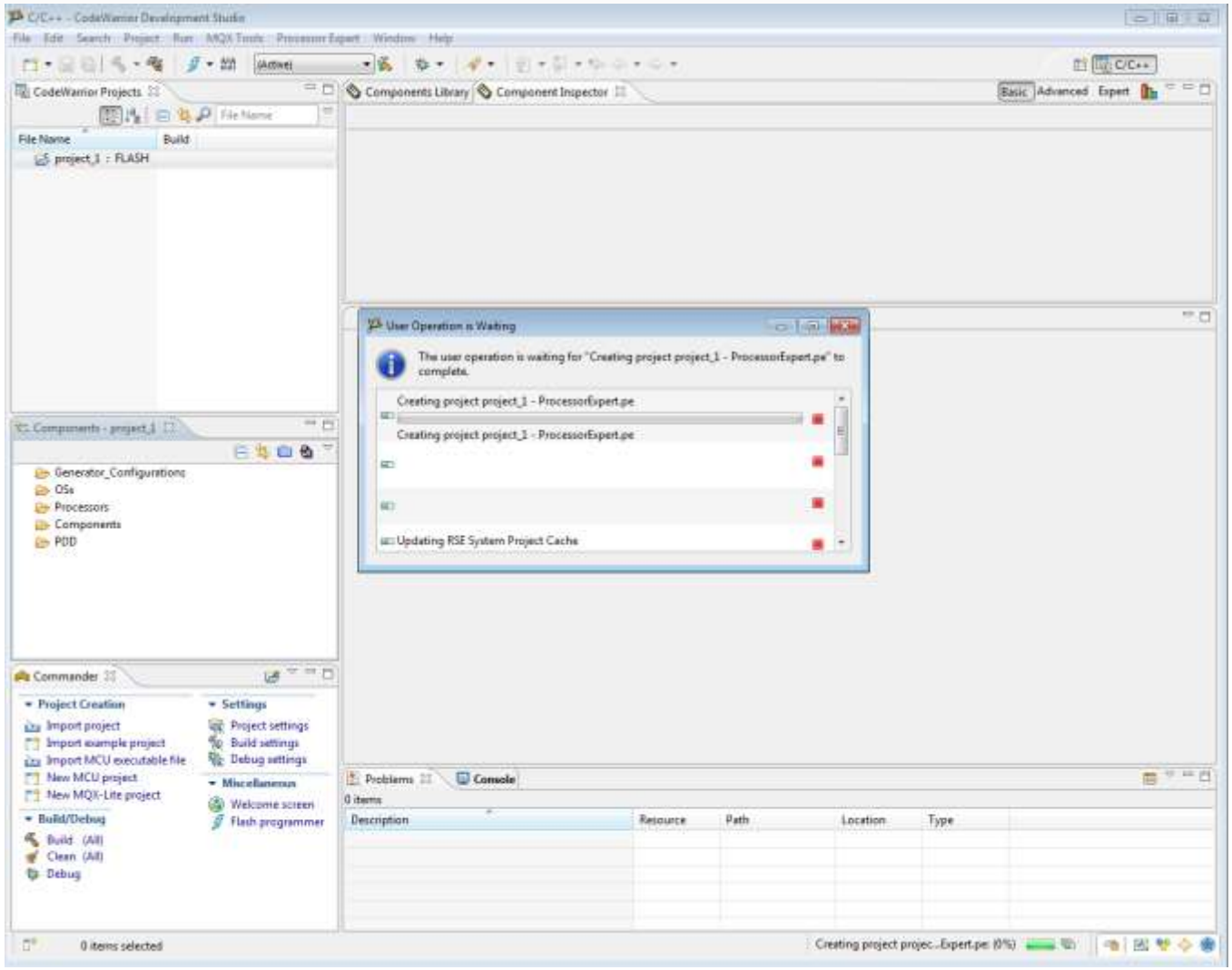


Select Language and Processor Expert

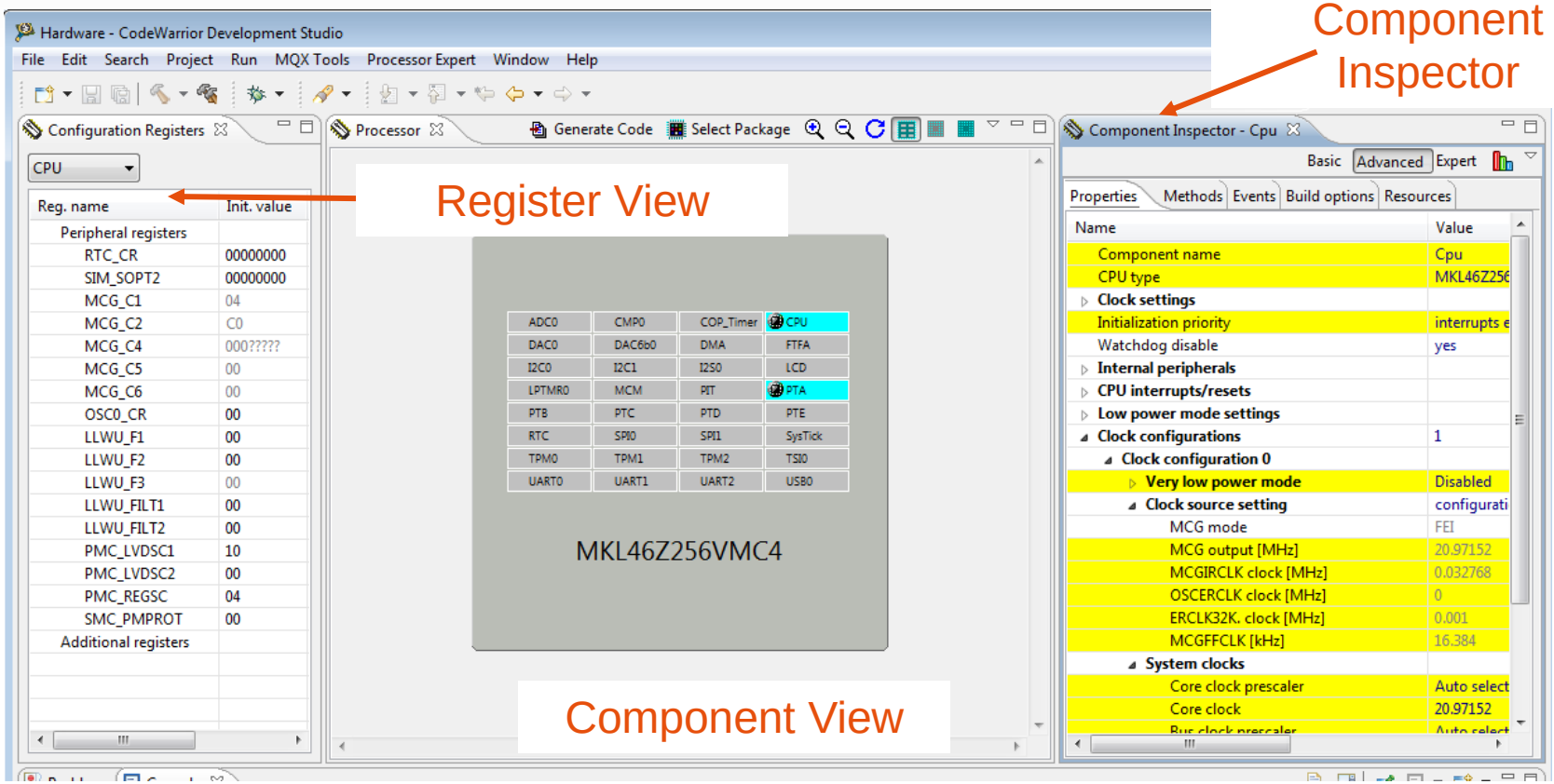




Project Creation in Progress

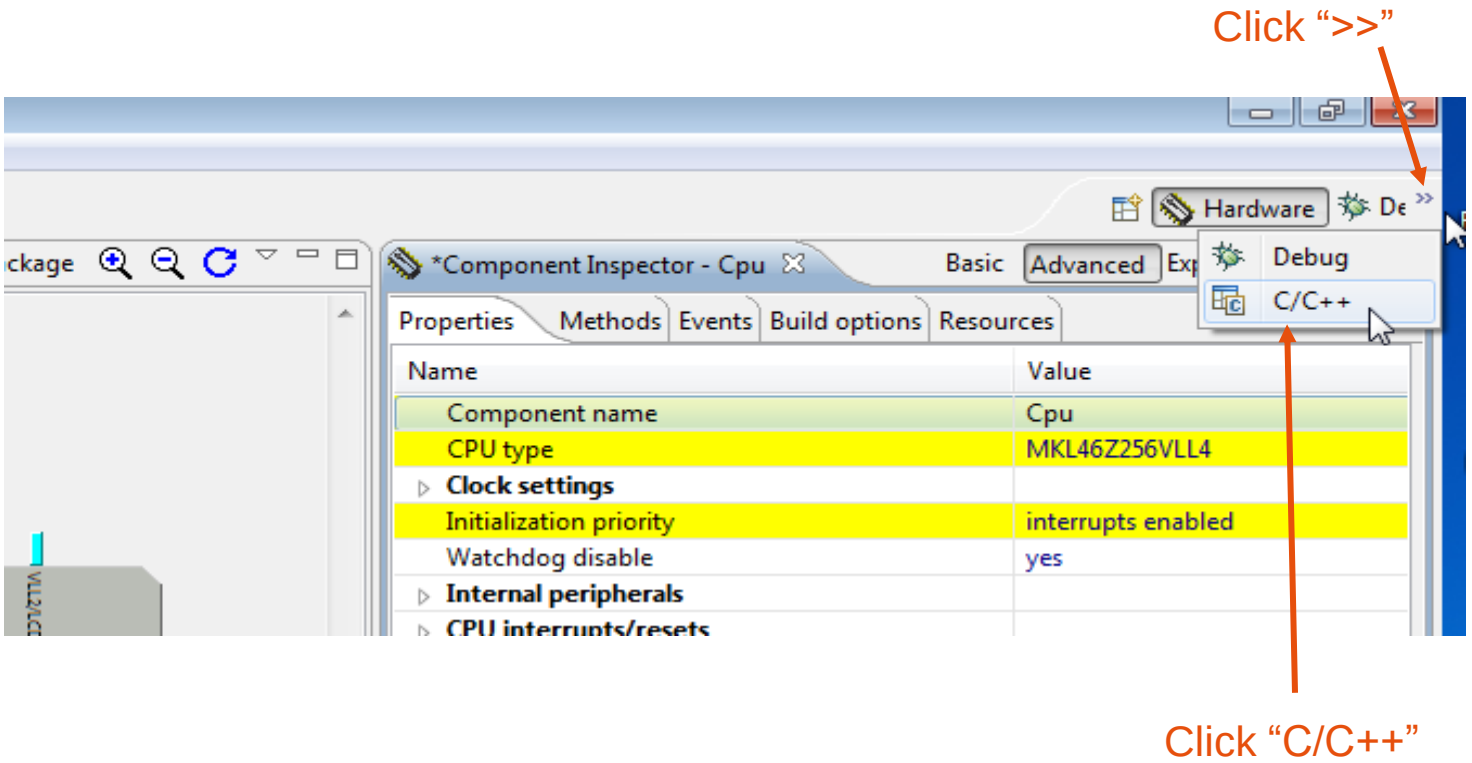


Project displayed in Hardware Perspective

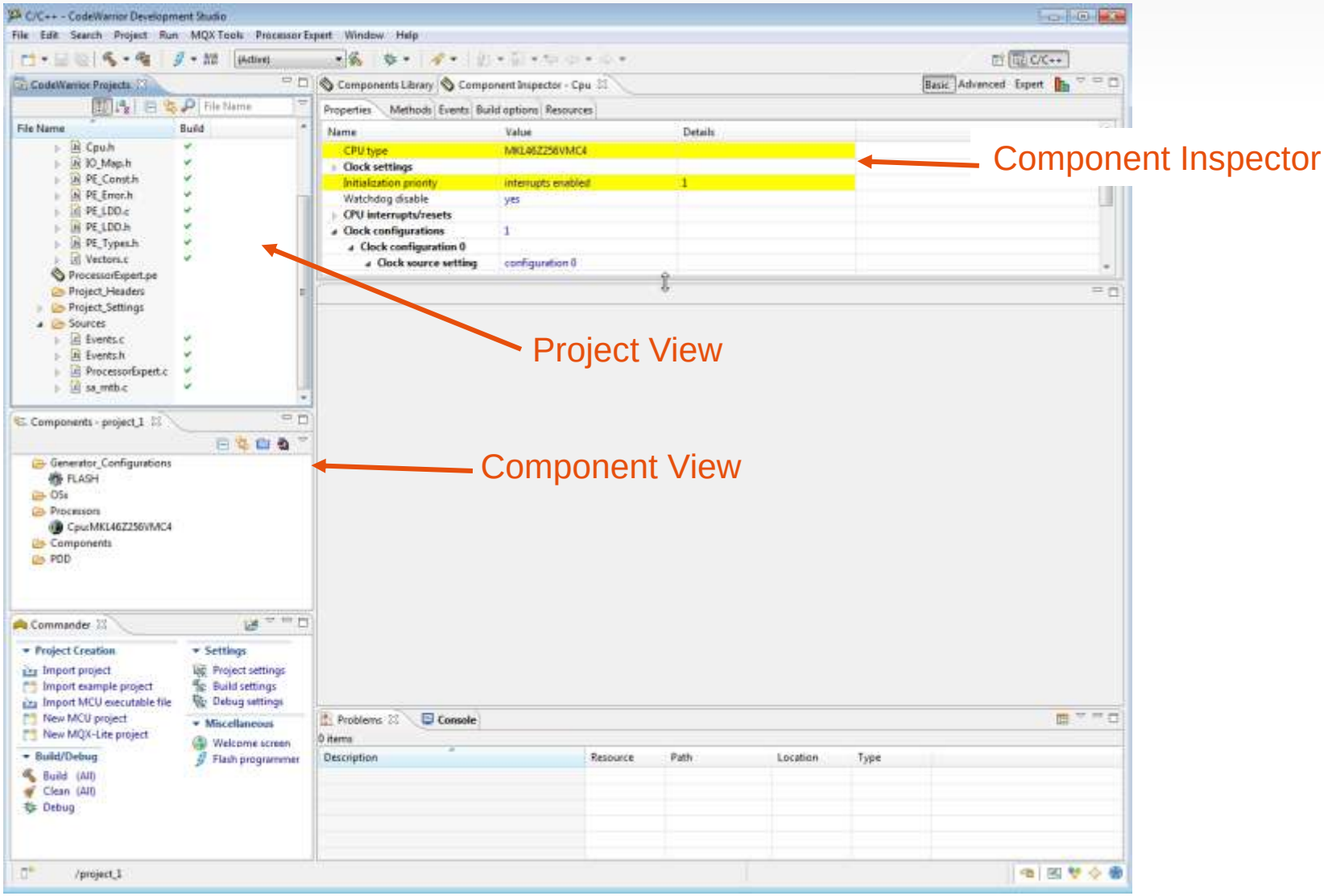


Project displayed in Hardware Perspective

- Select C++ Perspective



Project displayed in C/C++ Perspective



Check Point: Create a New Project to Blink an LED

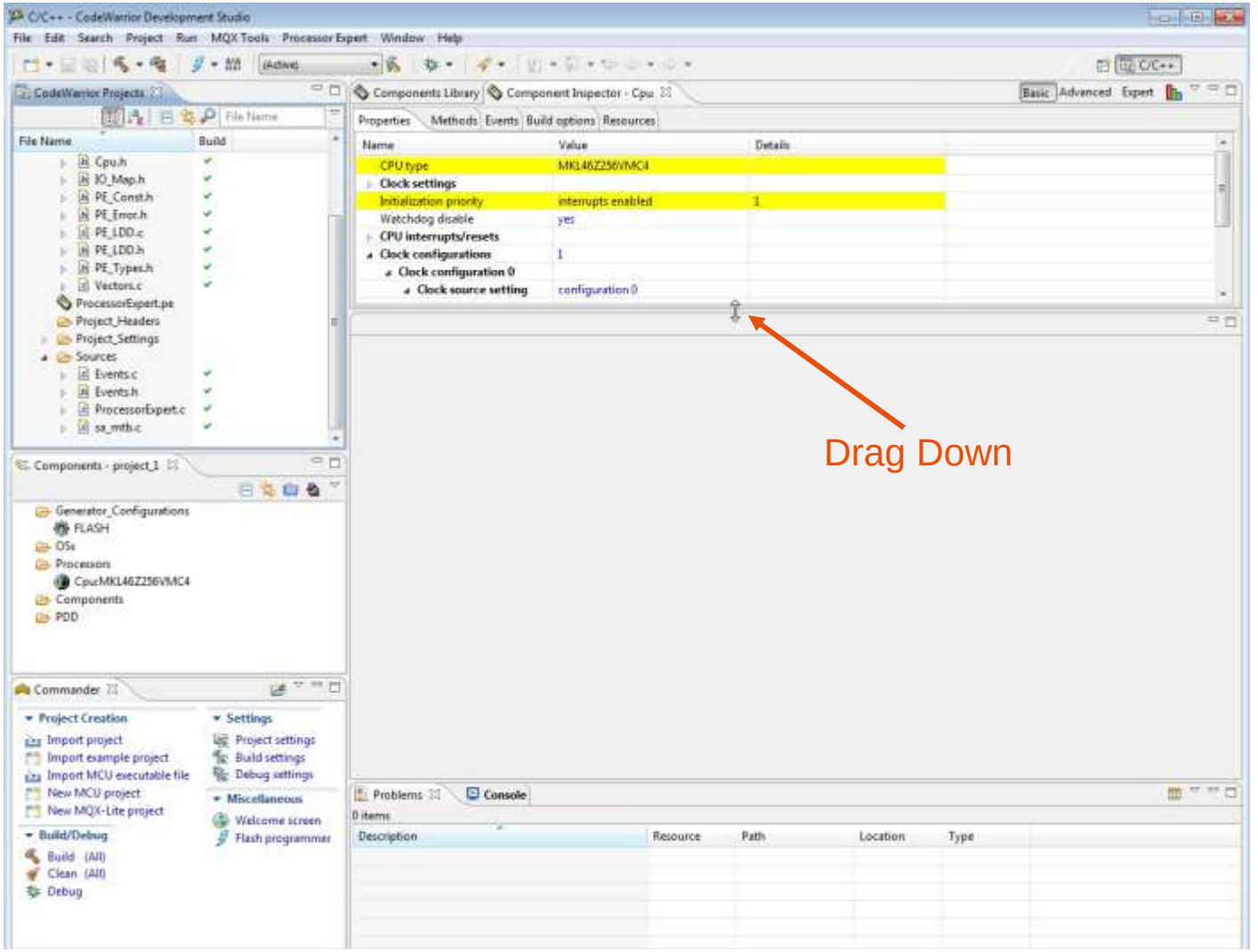
- This hands-on lab shows you how to…
  - Create a new project with the New Project Wizard ✓
  - **Select & Configure Components**
  - Configure Components with the Component Inspector
  - Use High Level Device functions
  - Import existing components
  - Build the project
  - Test the application's functionality
  - Trace the Code

Next up!

Selecting Components

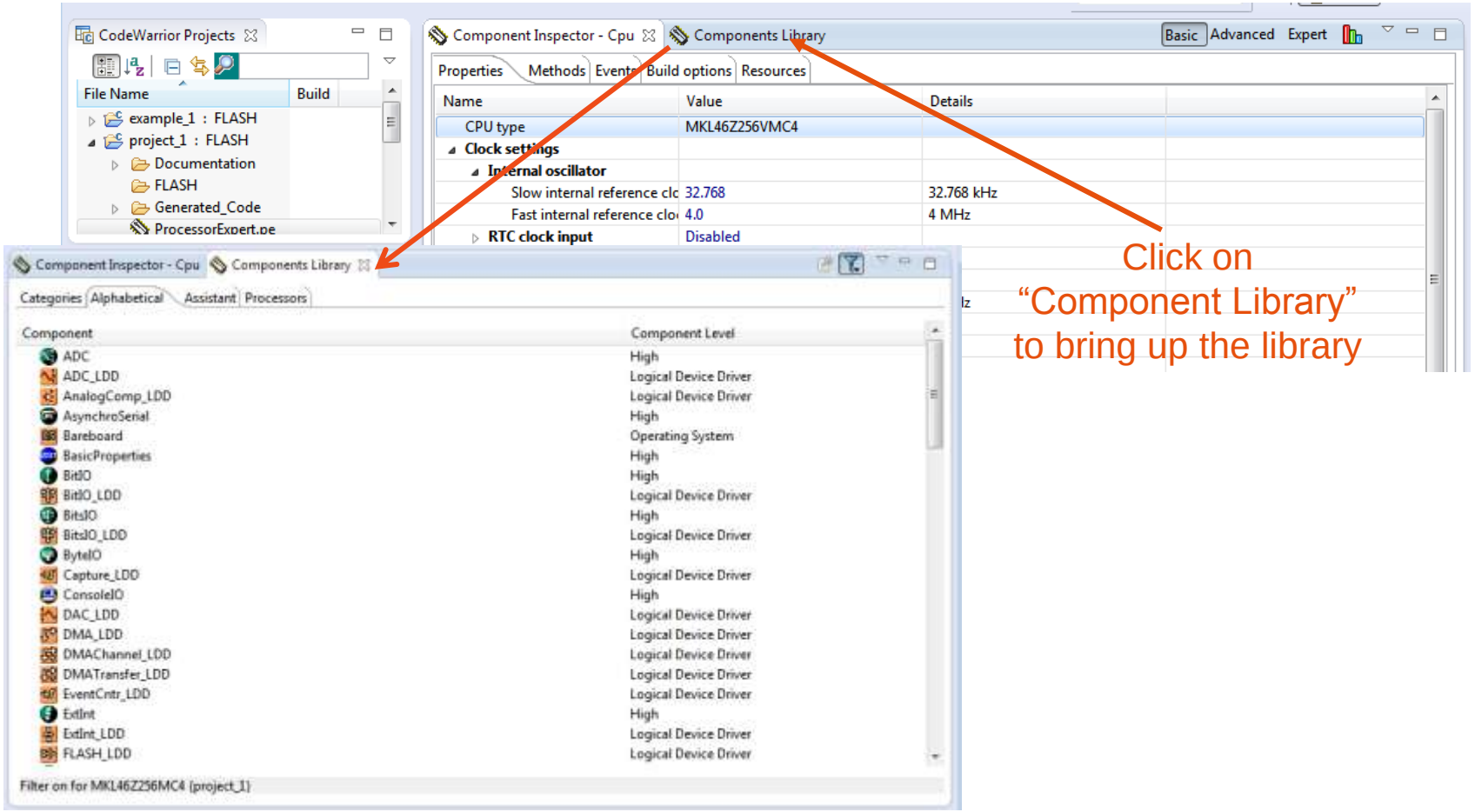
- Components Needed:
  - Processor: CPU: MKL46Z256VMC4 (Preselected based on project wizard information)
- Peripheral Initialization:
  - GPIO: Init_GPIO
- Other components needed for project (High Level Components)
  - GPIO: BitIO: for SW1
  - GPIO: BitIO: for SW2
  - GPIO: BitIO: for Red LED
  - GPIO: BitIO: for Green LED
  - Timer: TimerInt: Flashing the LED
  - AsynchroSerial: Serial UART for debug output

Component Inspector



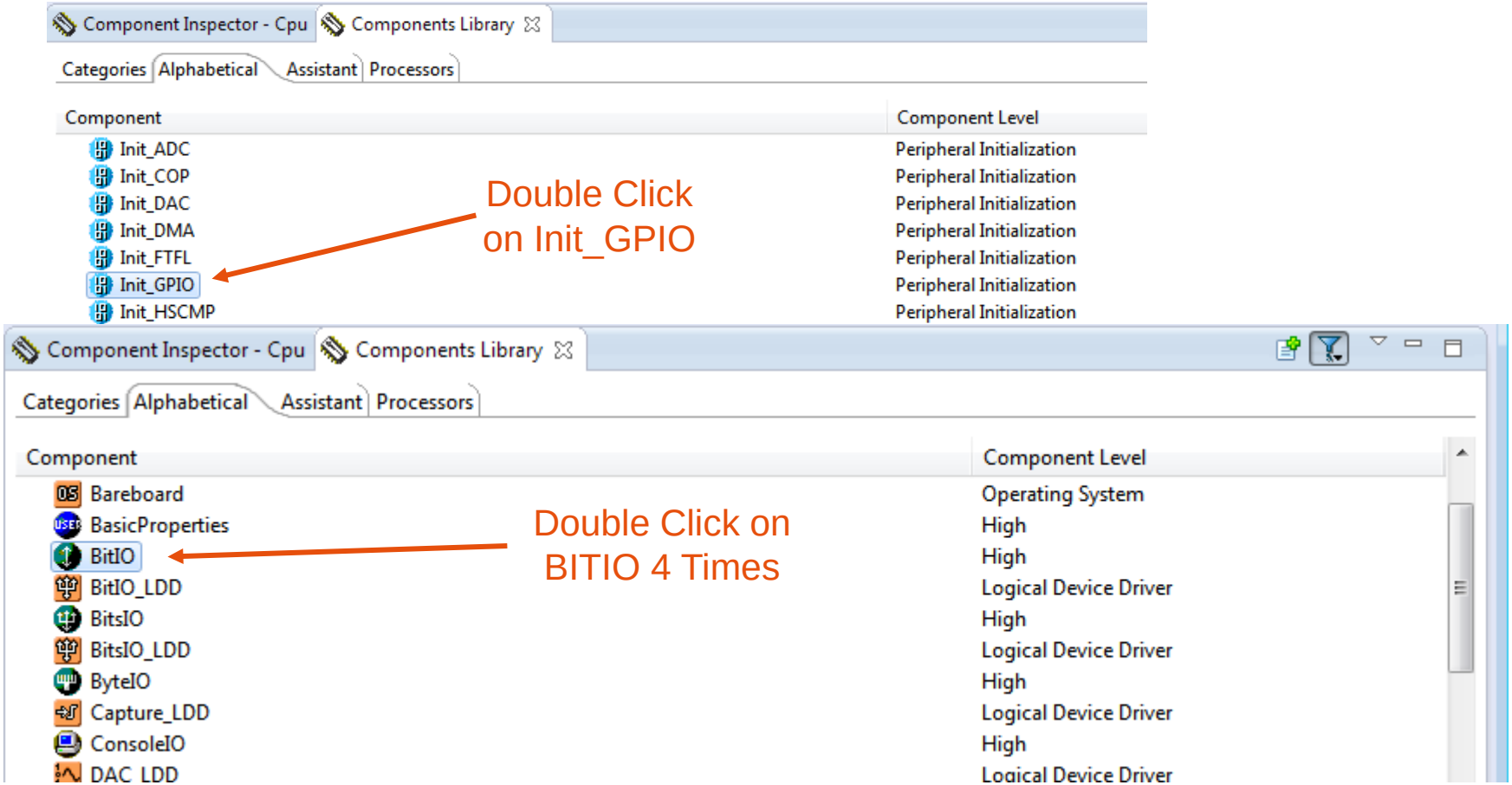
Selecting Components

- Switch to Component Library

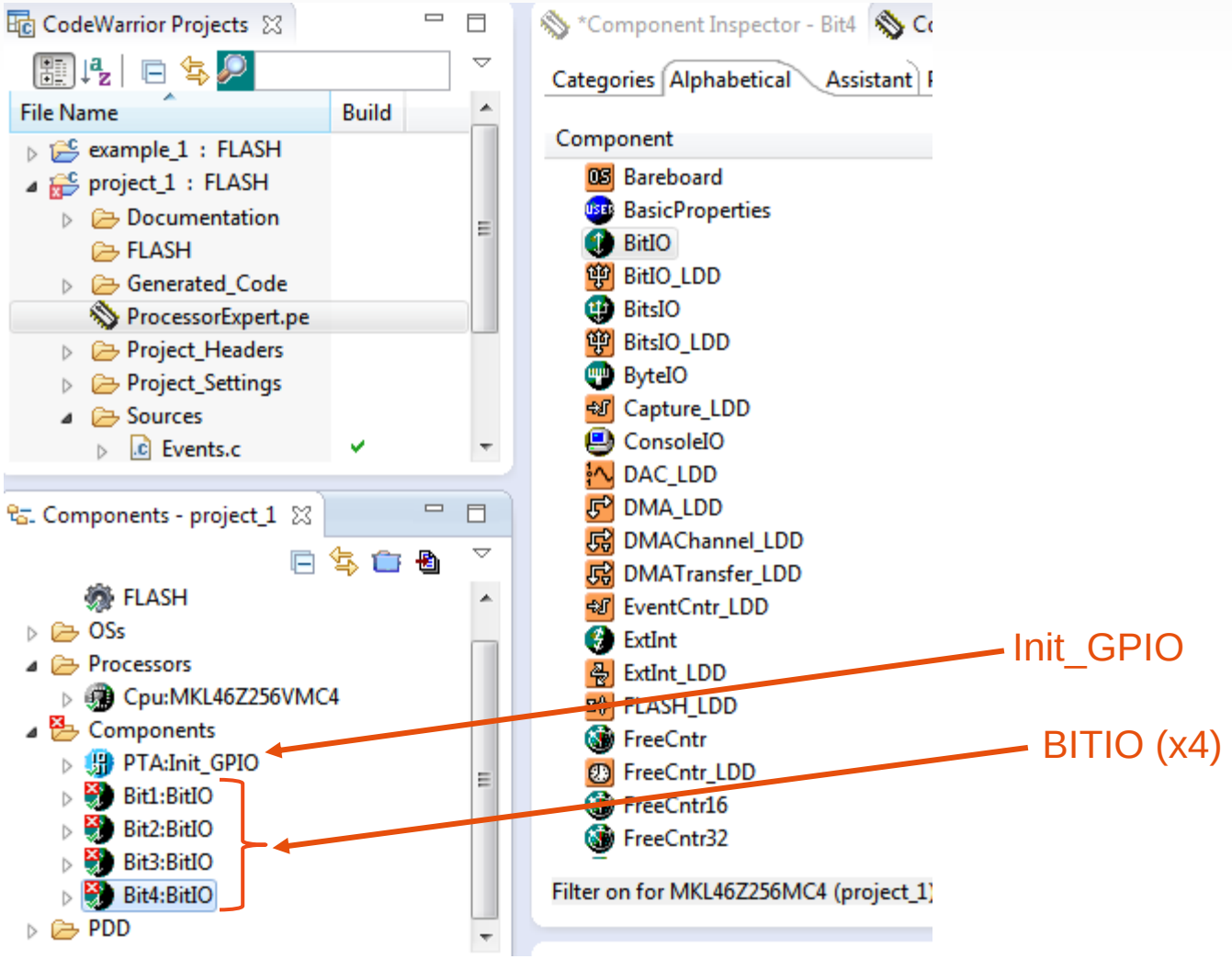


Selecting Components

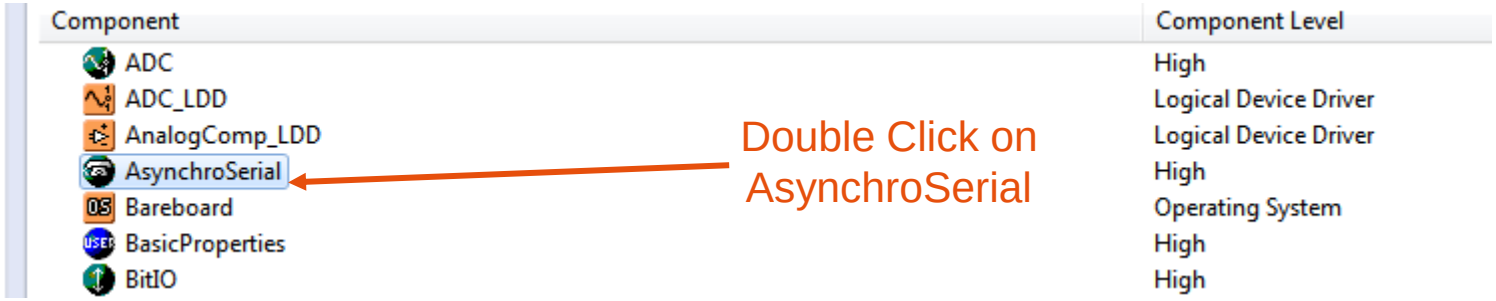
- Double Click on “Init_GPIO”



Selecting Components

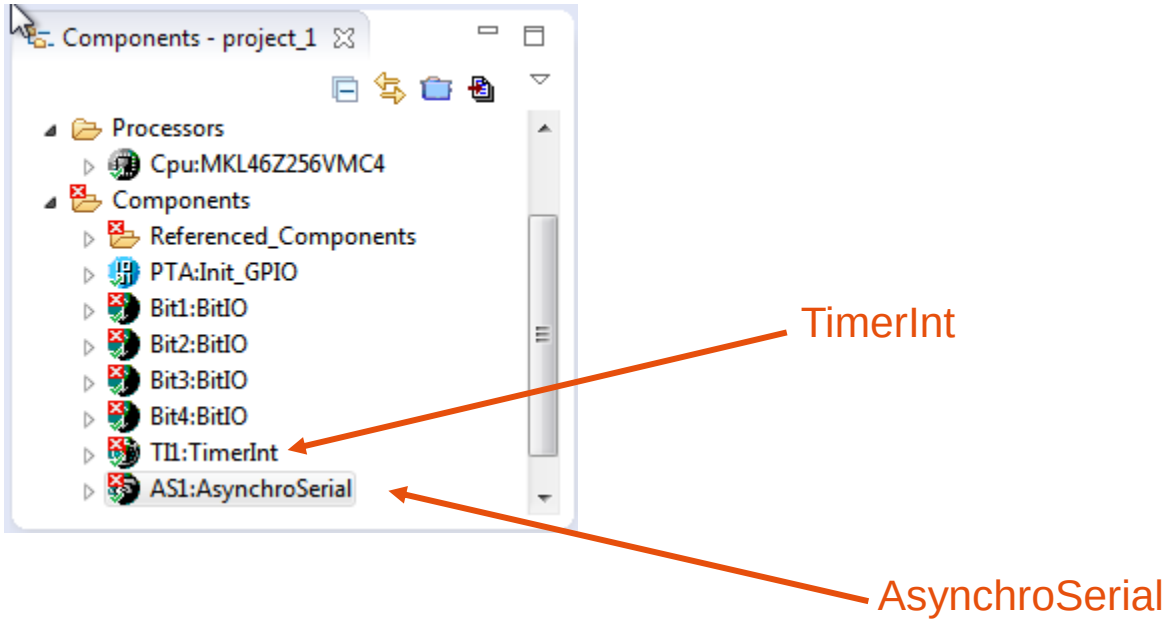


Selecting Components



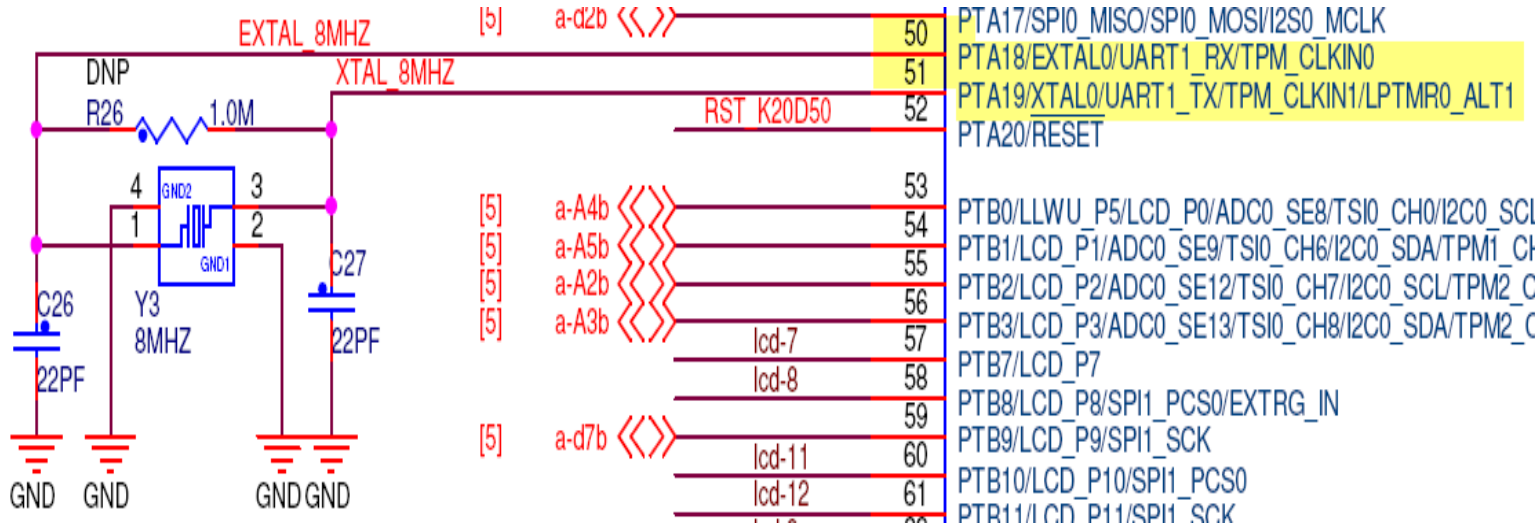
Selecting Components

- The component list should look like this:



FRDM-KL25Z Schematic

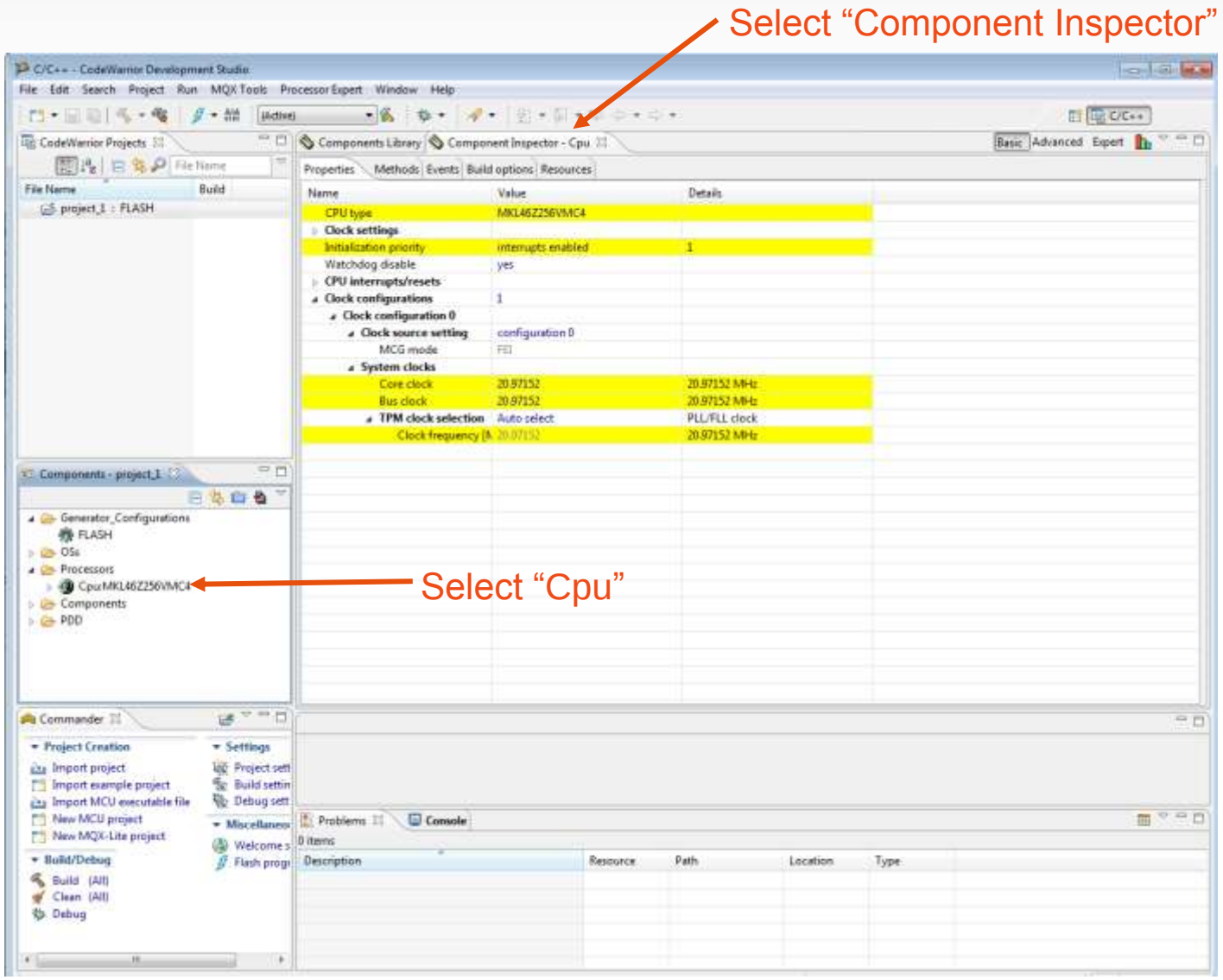
- 8.0 MHz External Clock Input
  - EXTAL0 – pin 50
  - XTAL0 – pin 51



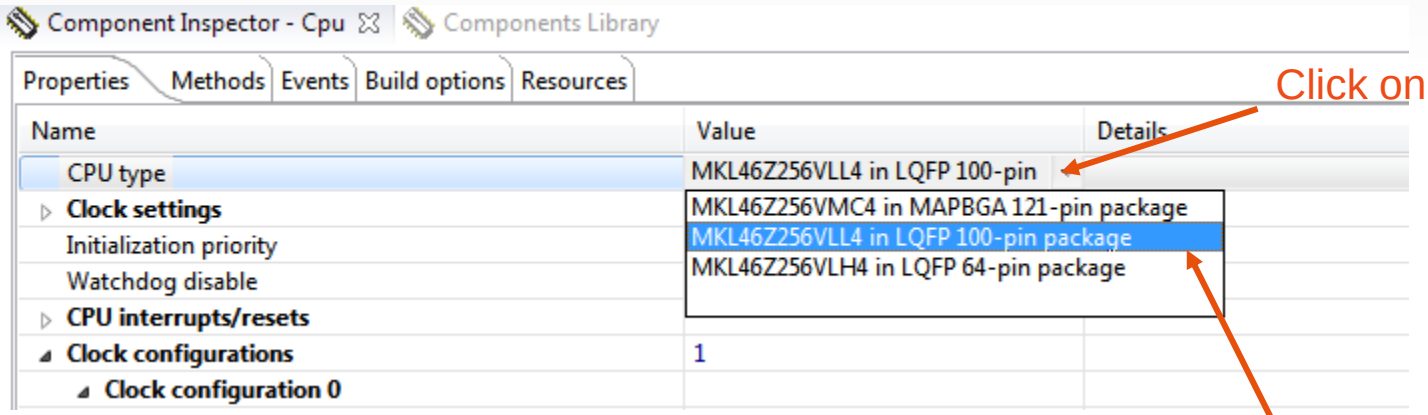
Configure CPU Component

- Configure the CPU component as follows:
  - Package 100 pin LQFP
  - System Oscillator Enabled
  - External Clock 8MHz input
  - MCG Mode PEE
  - PLL Output 96MHz
  - Core Clock 48MHz
  - Bus Clock 24MHz

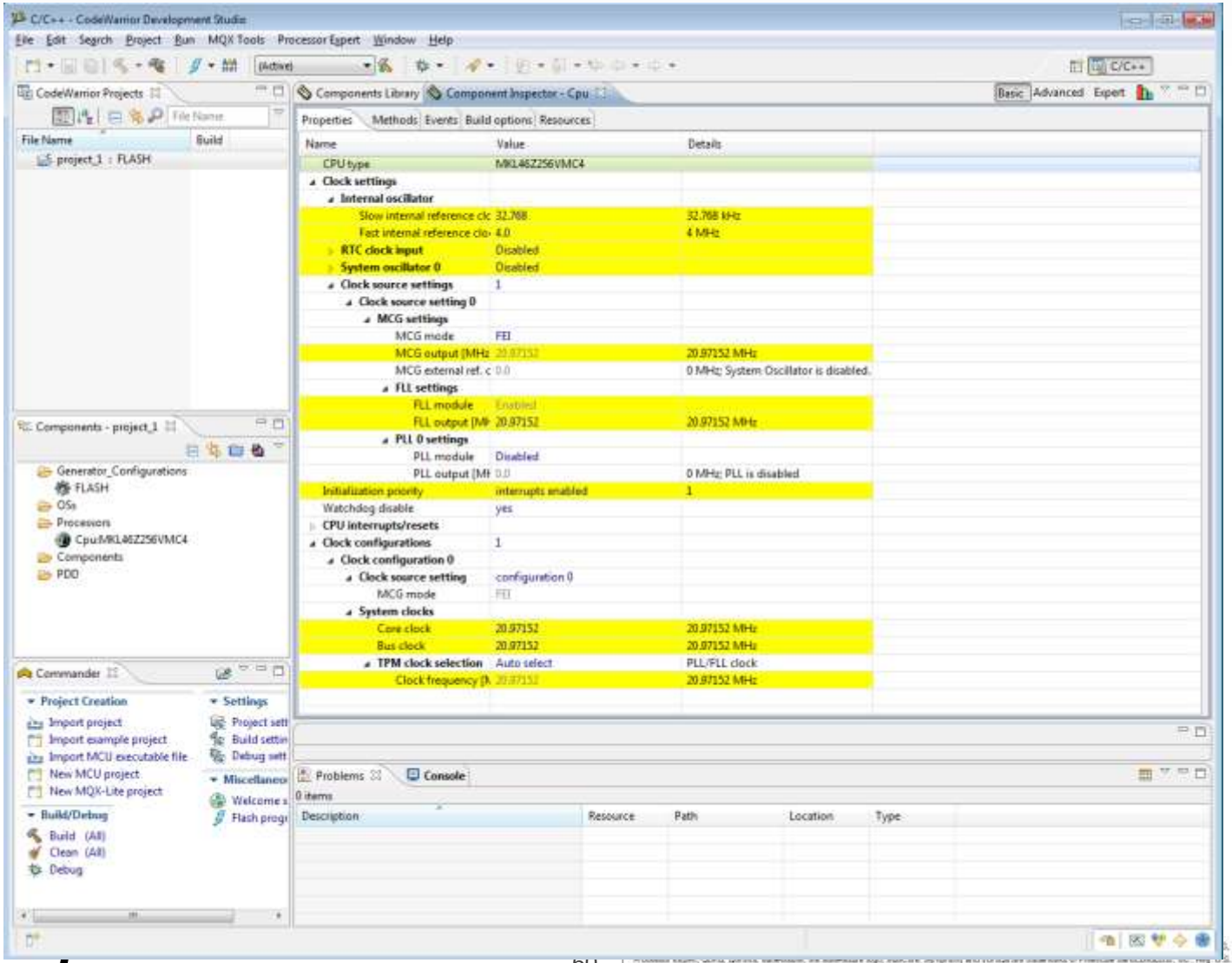
Component Inspector



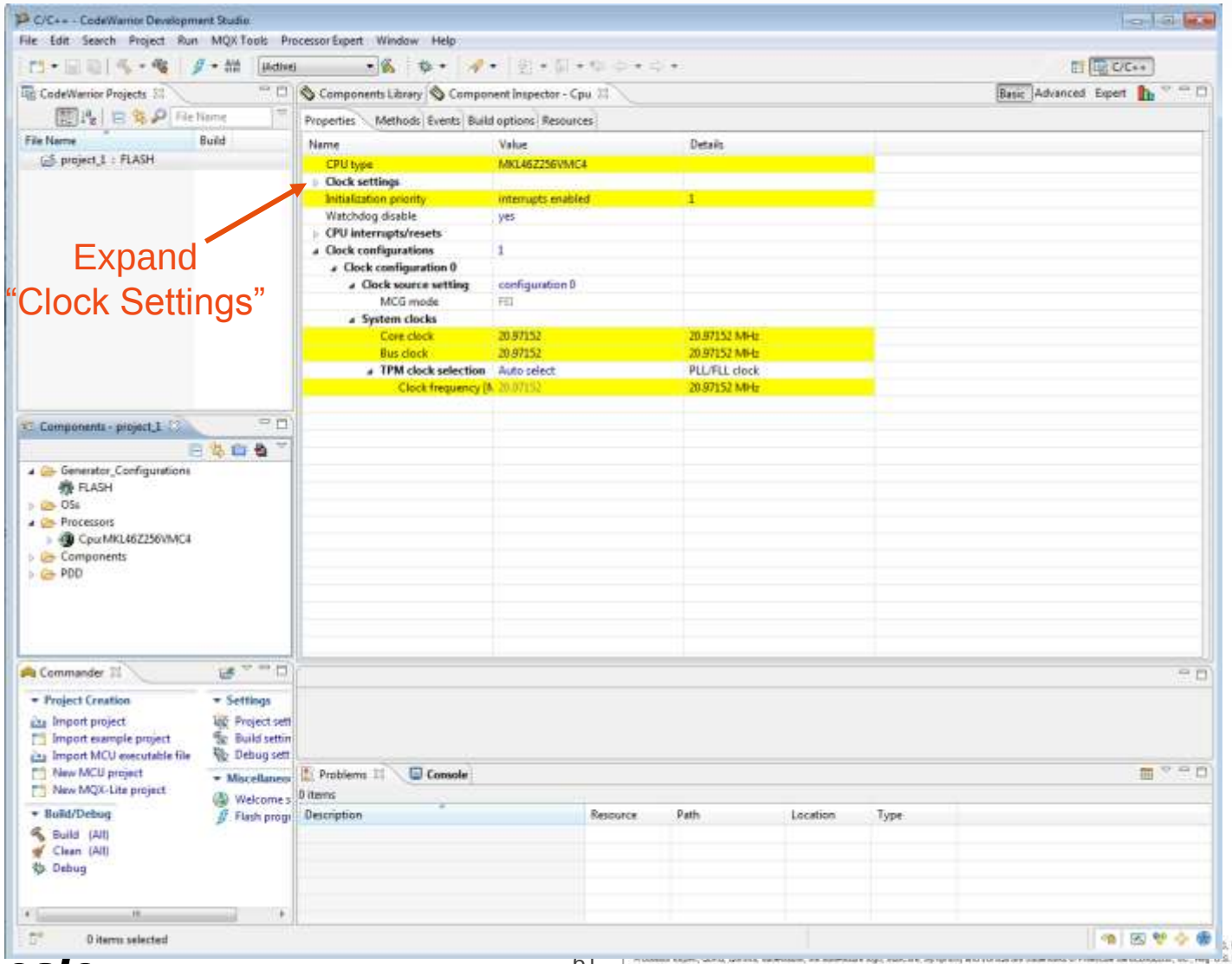
Component Inspector – Package



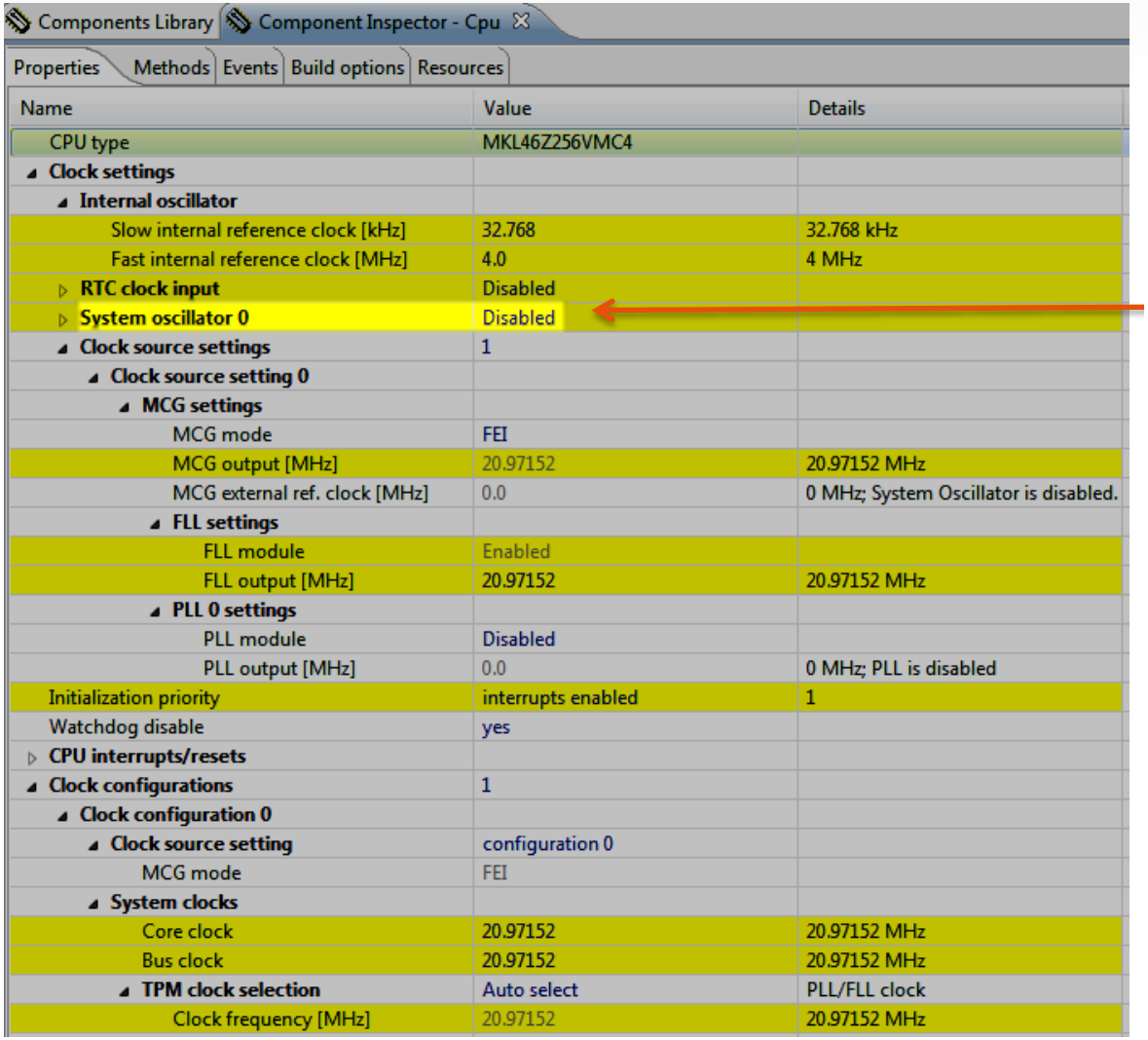
Component Inspector – Clock Settings



Component Inspector – Clock Settings

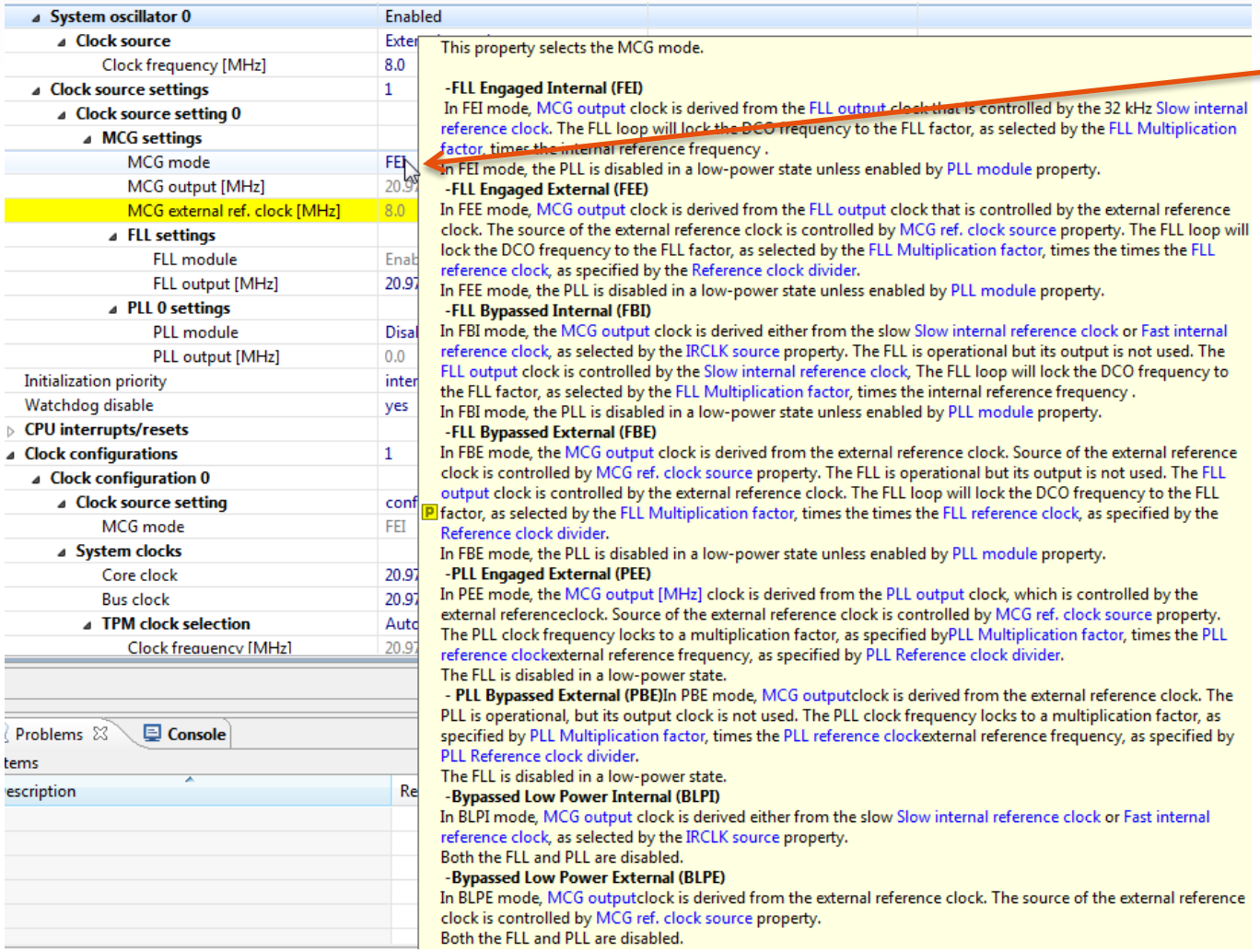


Component Inspector – Clock Settings

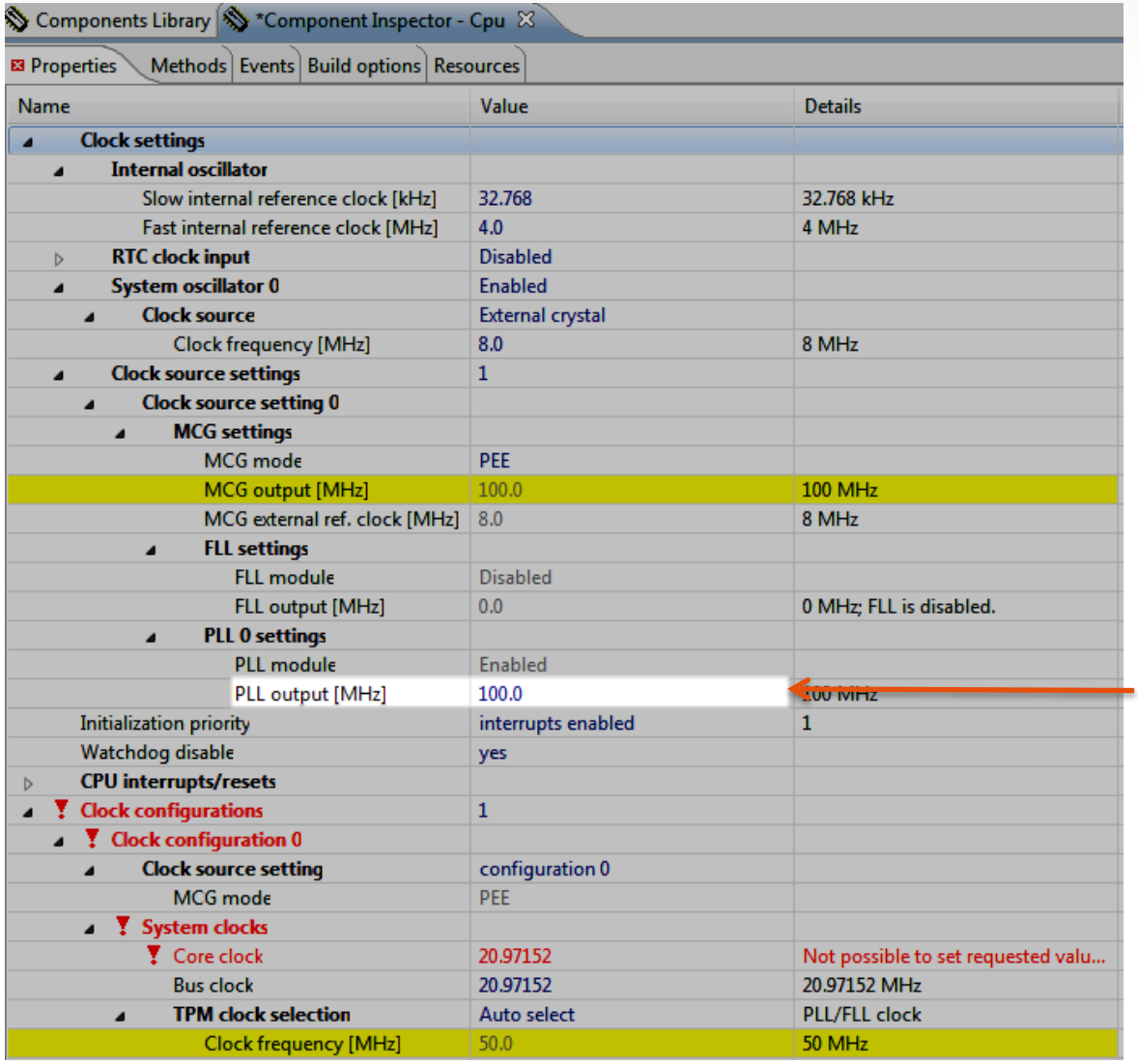


Select "Enable"

Component Inspector – Clock Settings

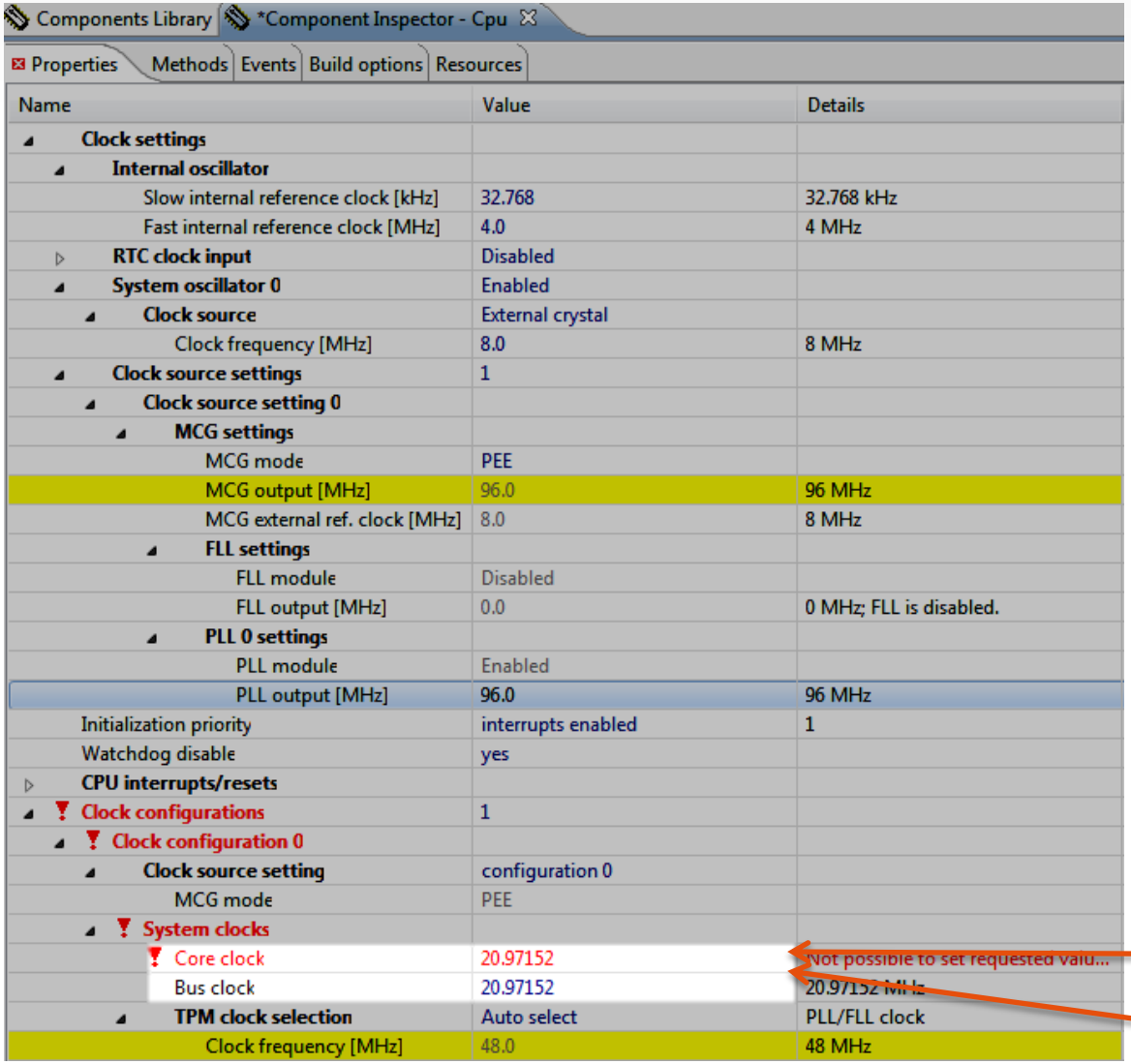


Component Inspector – Clock Settings

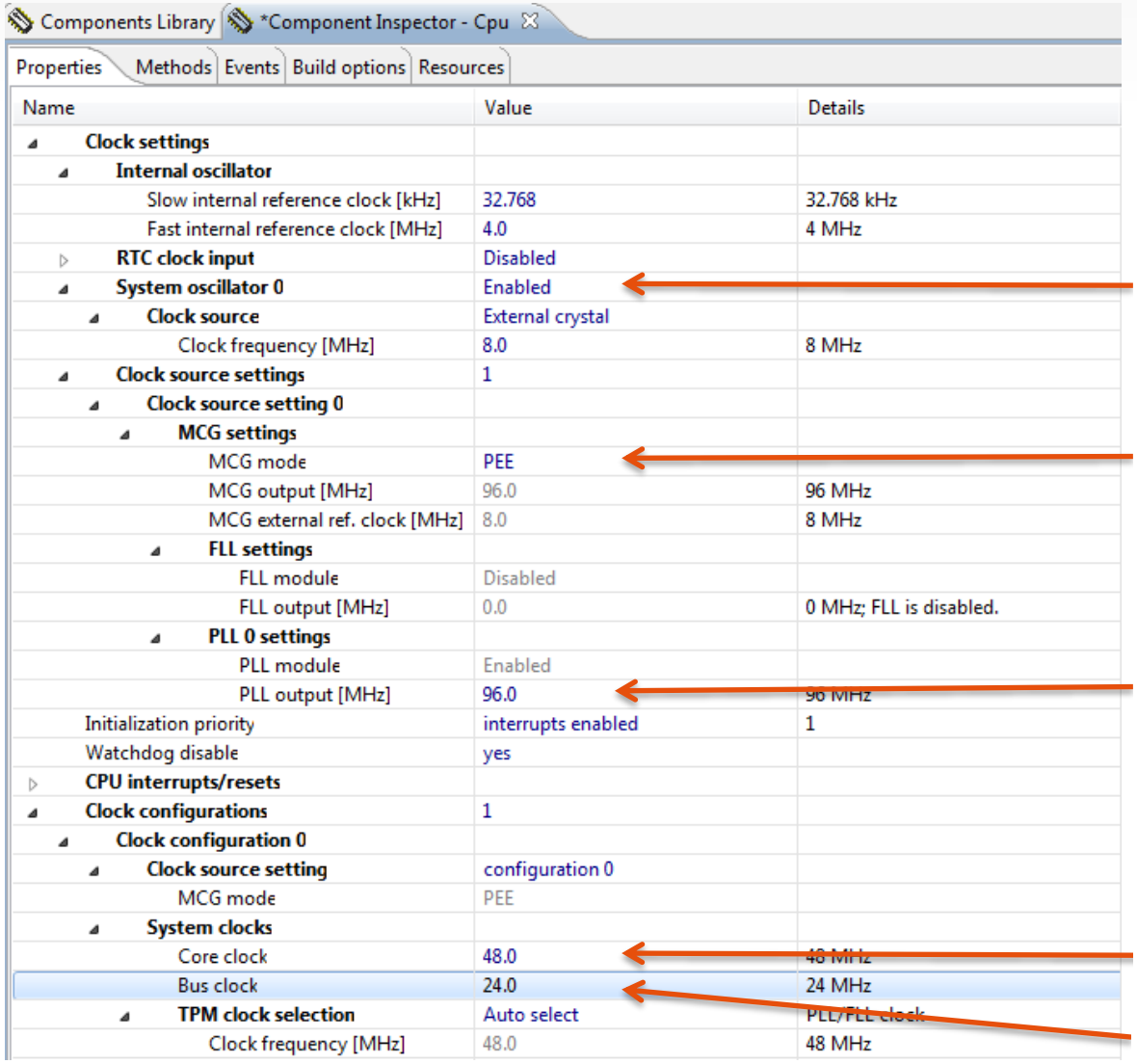


Enter “96”

Component Inspector – Clock Configurations

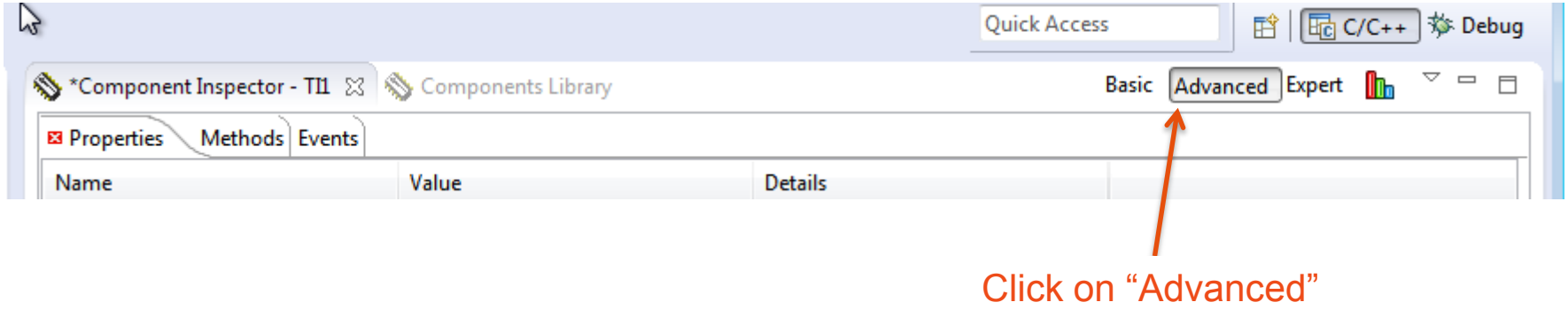


Component Inspector: CPU Summary



Component Inspector

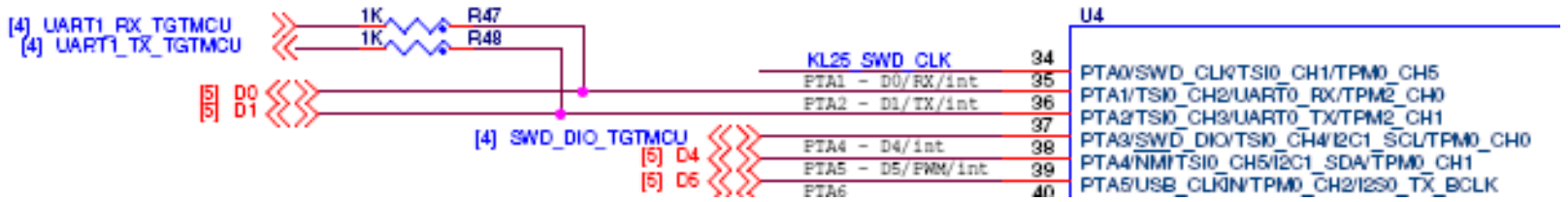
- Change to “Advanced” Mode



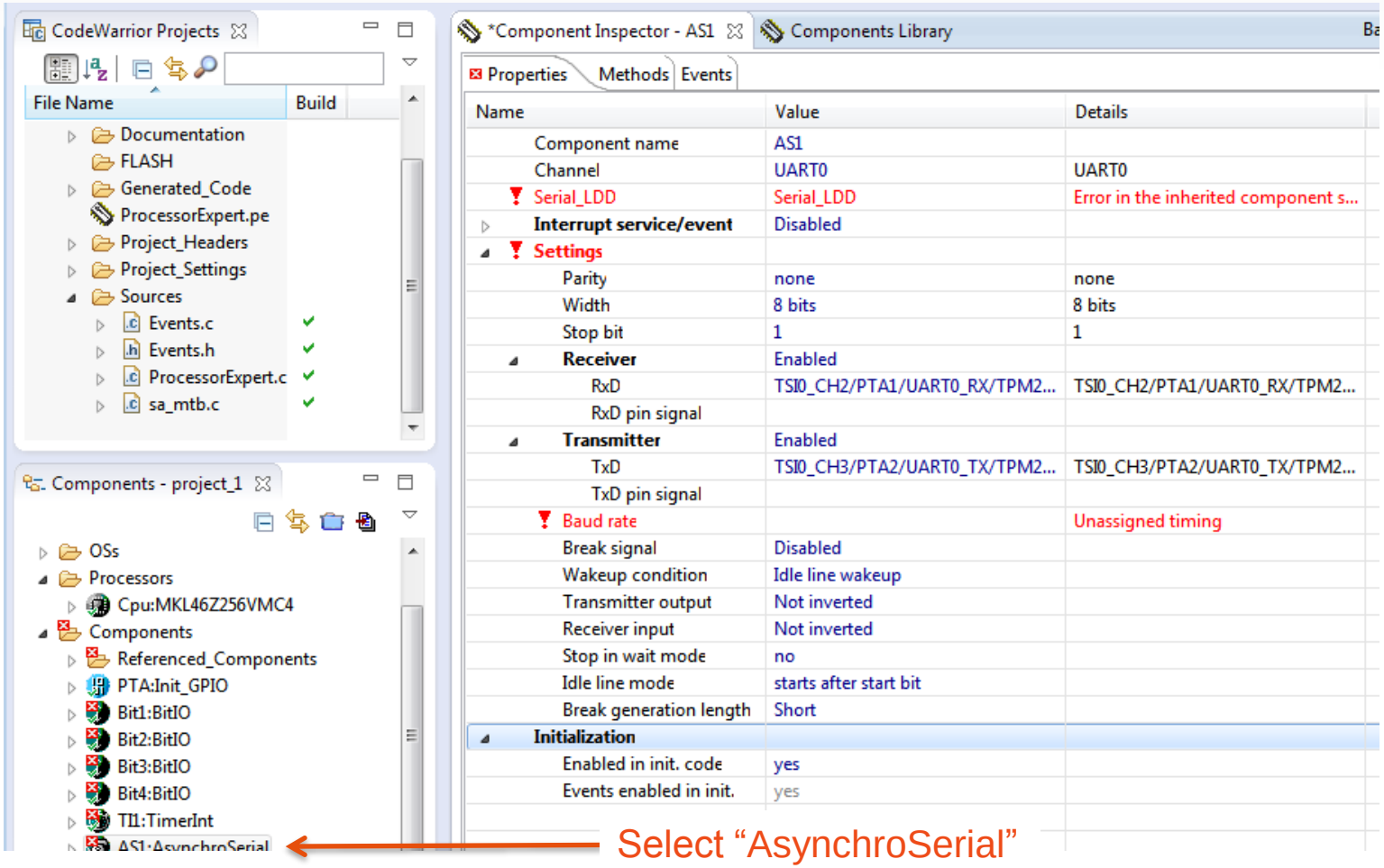
Configure AsynchroSerial Component

- Configure the AsynchroSerial component as follows:
  - Baud rate 115200
  - Receiver RxD UART0
  - Transmitter TxD UART0
  - Leave all other settings at their default settings
    - Channel: UART0
    - Parity: none
    - Width: 8 bits
    - Stop bit: 1
    - Receiver: Enabled
    - RxD: TSIO_CH2/…
    - Transmitter: Enabled
    - TxD: TSIO_CH3/…
    - Stop in wait mode: no
    - Idle line mode: starts after start bit
    - Enable in init code: yes

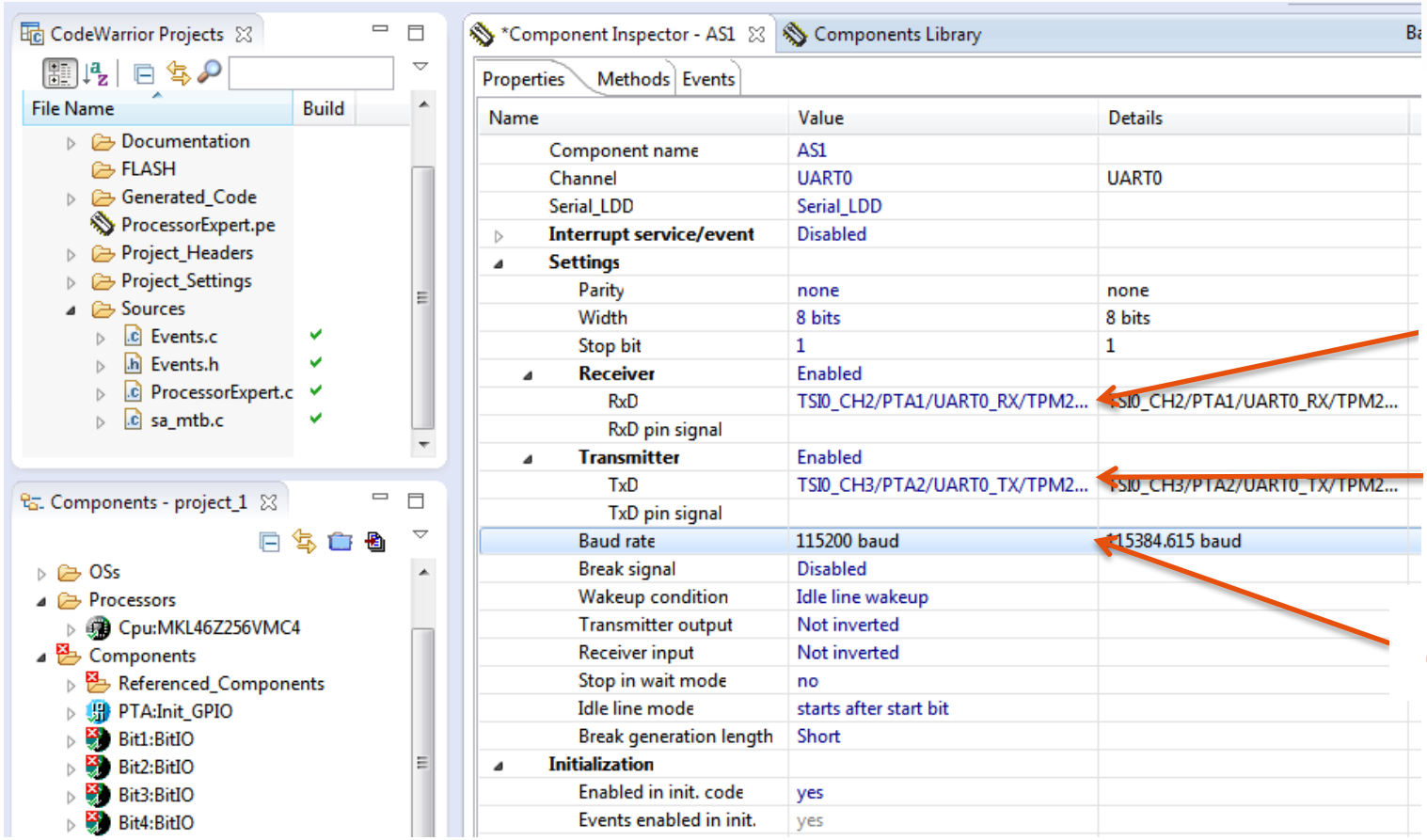
Configuration UART Component



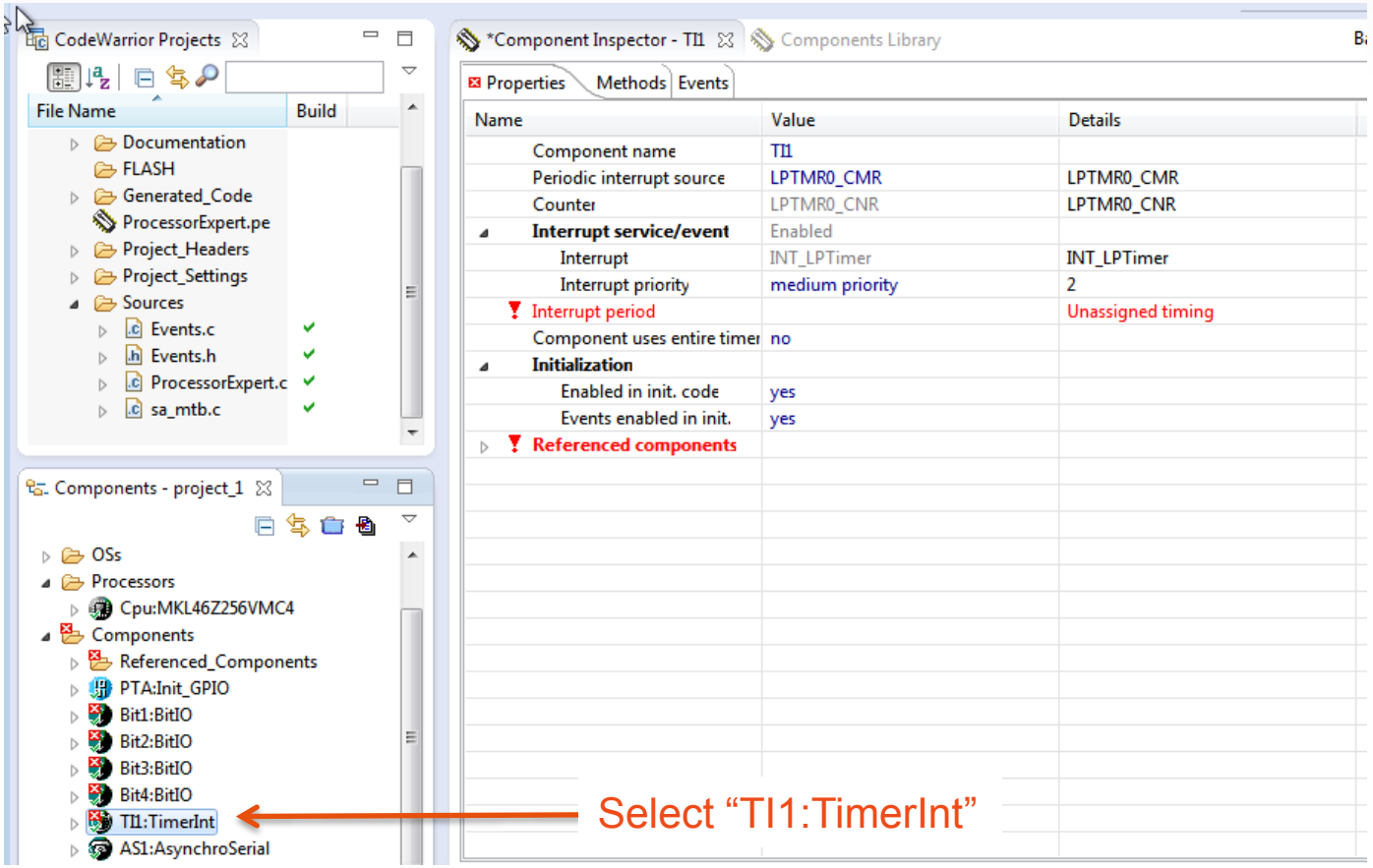
Peripheral Initialization – AsynchroSerial



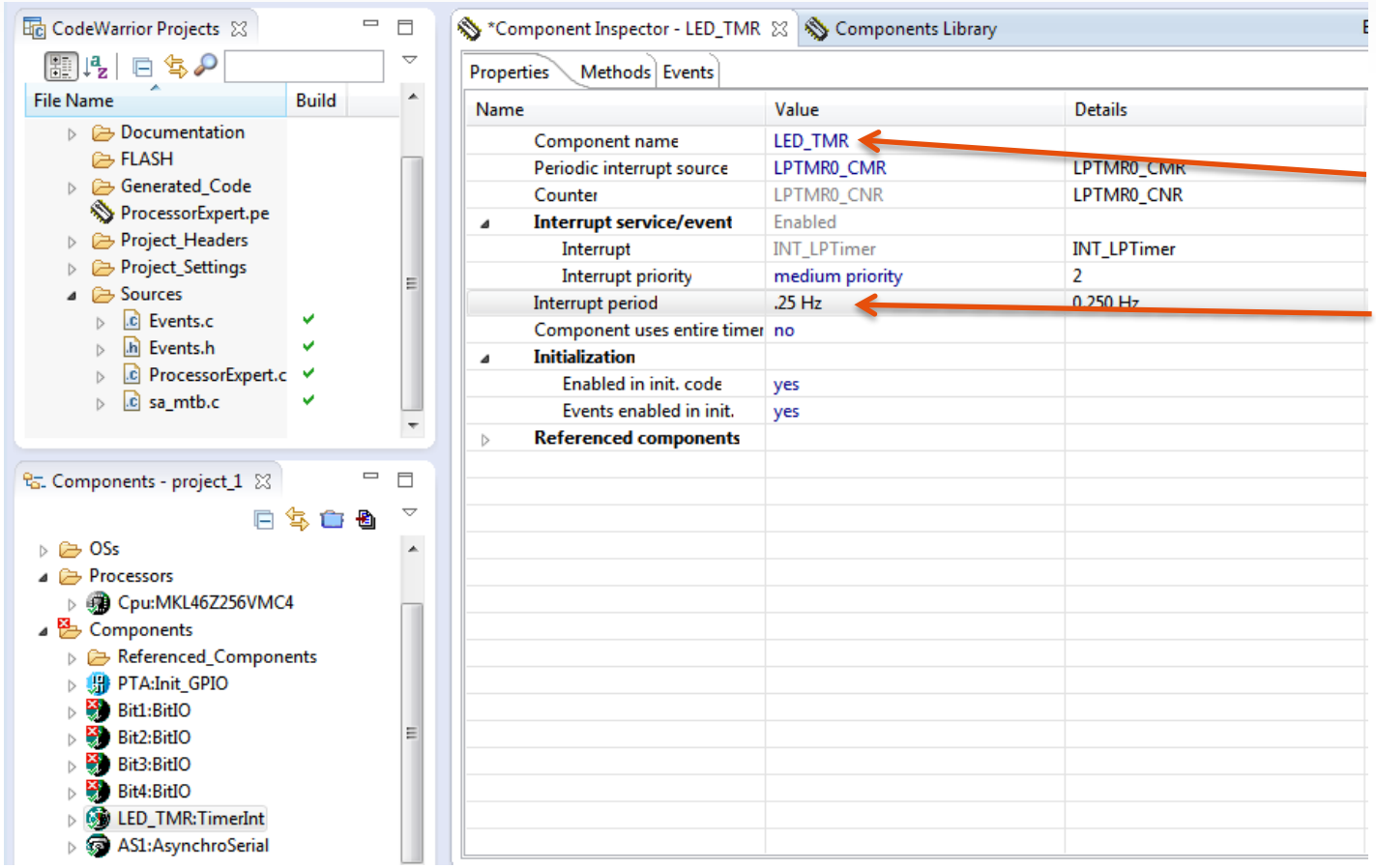
Peripheral Initialization – AsynchroSerial



Peripheral Initialization – TimerInt



Peripheral Initialization – TimerInt

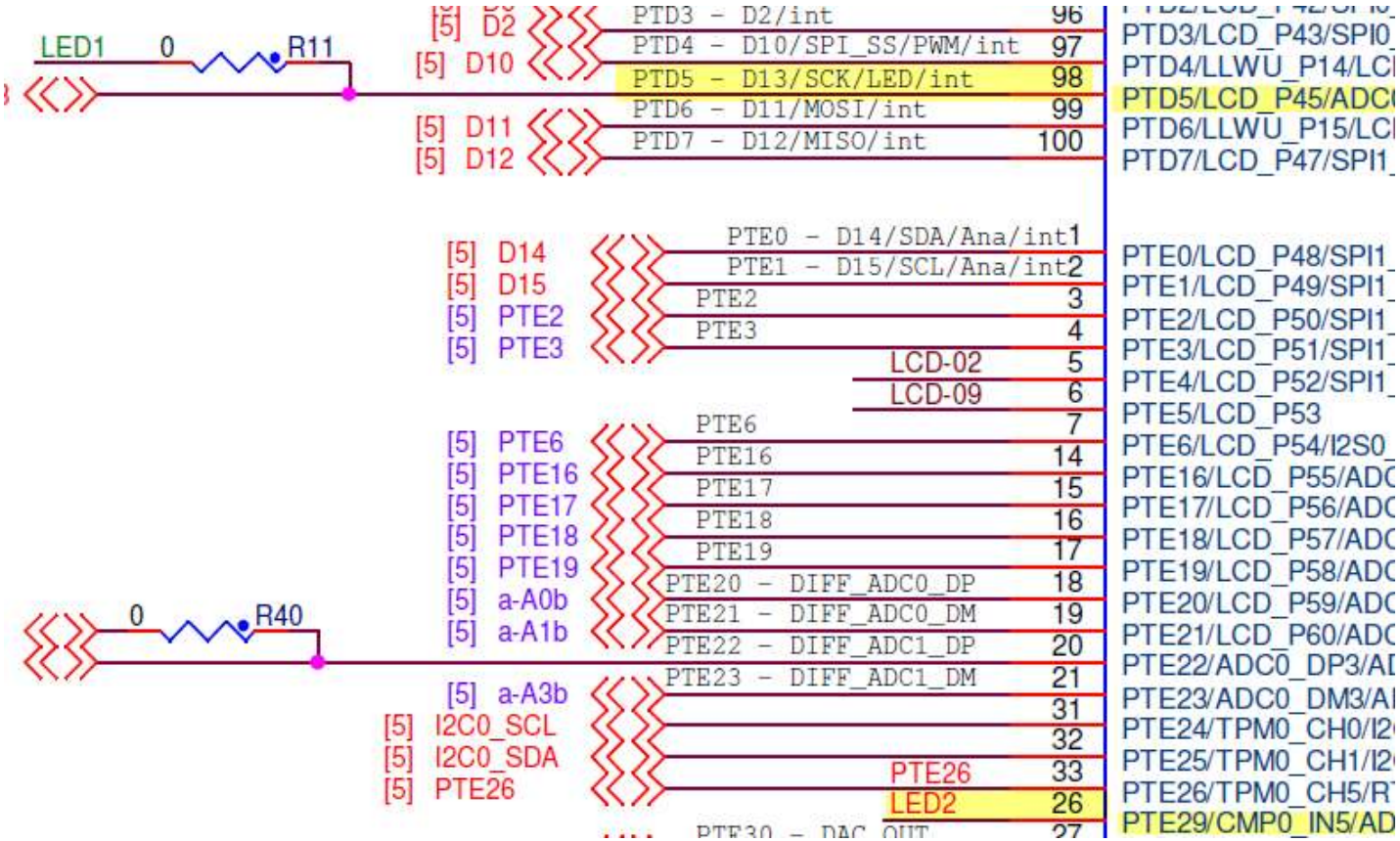


Select "LED_TMR"

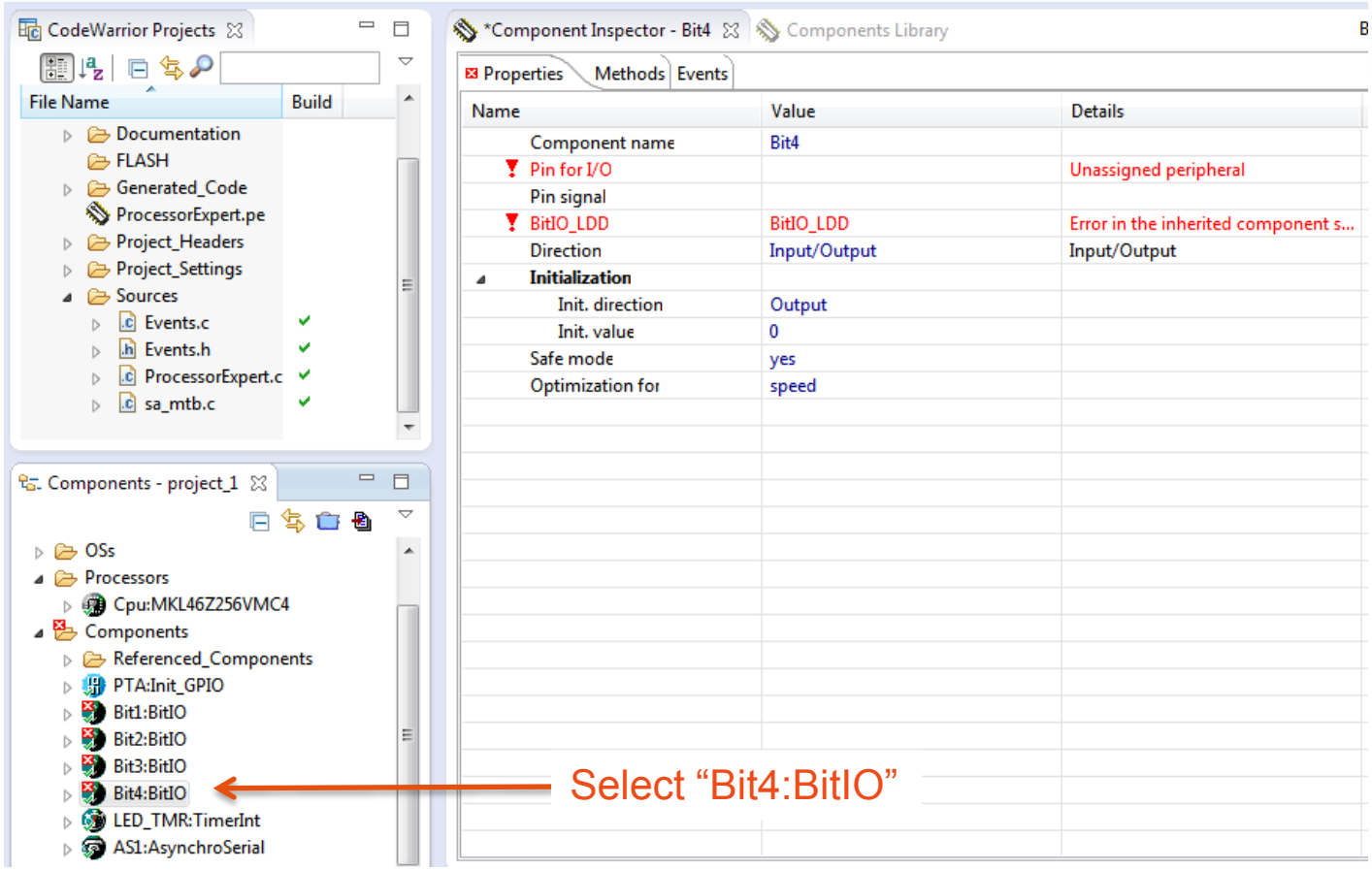
Enter "25Hz"

Hardware Pin Details

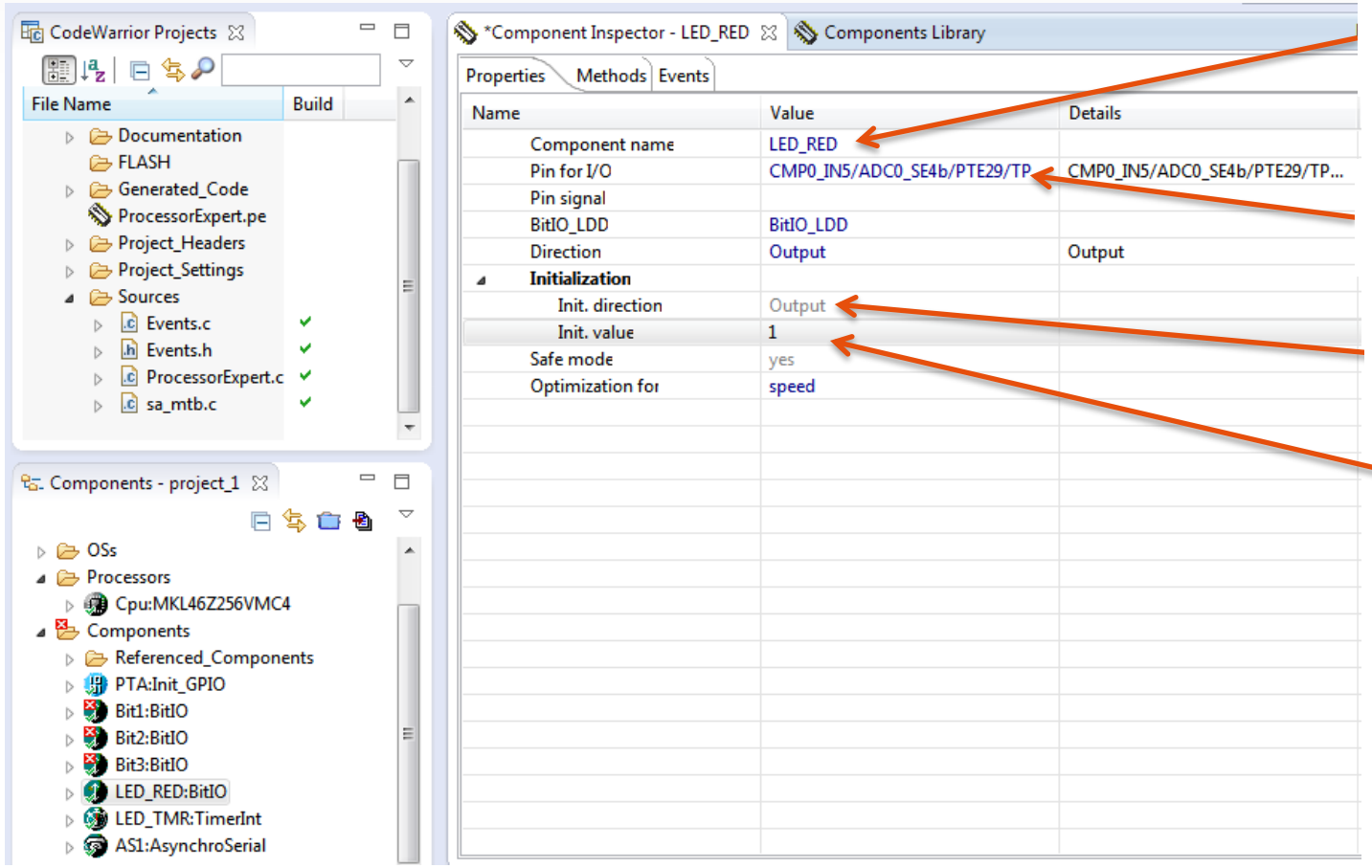
- LED1(green) on PTD5
- LED2 (red) on PTE29



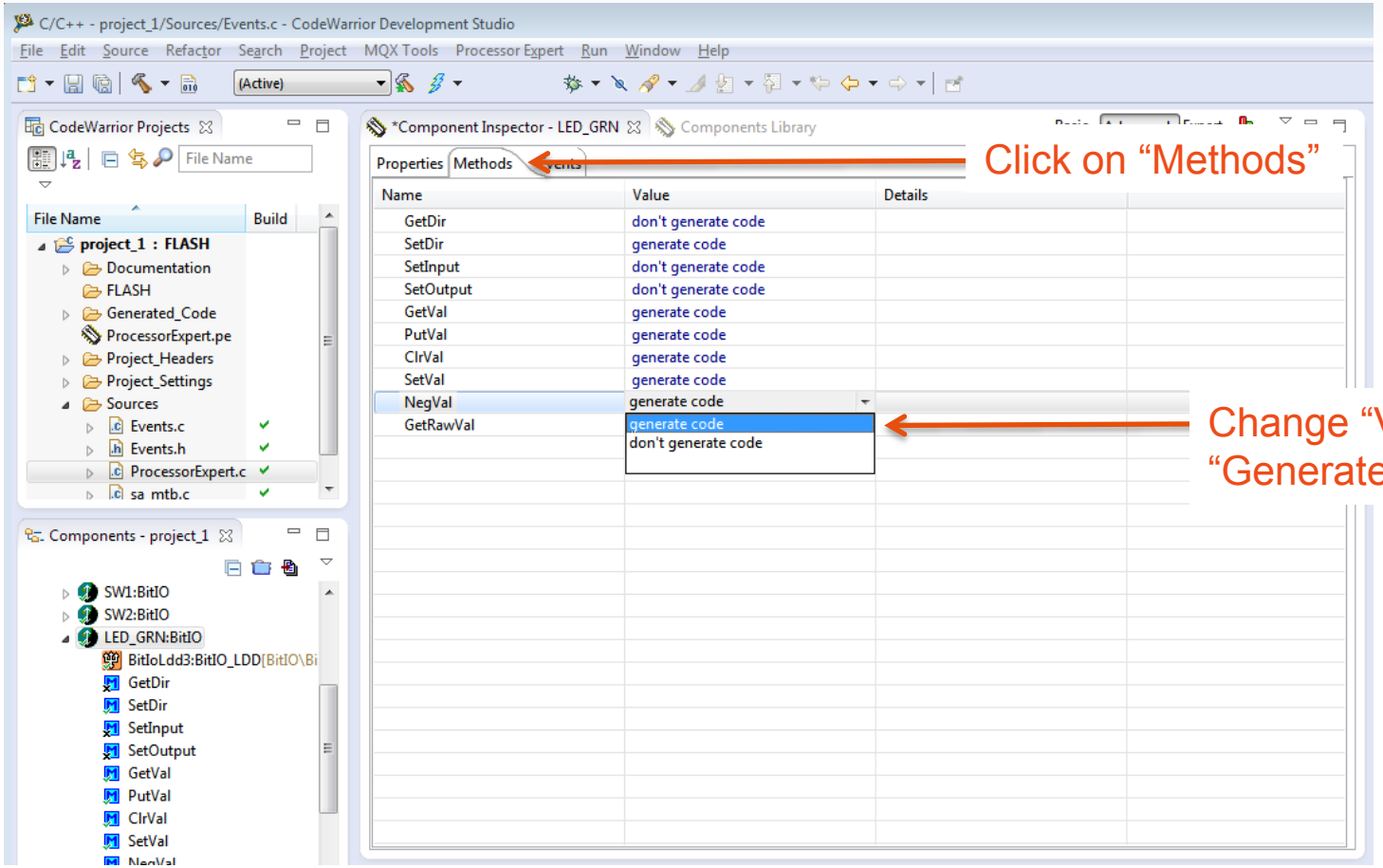
Peripheral Initialization – Bit4:BitIO



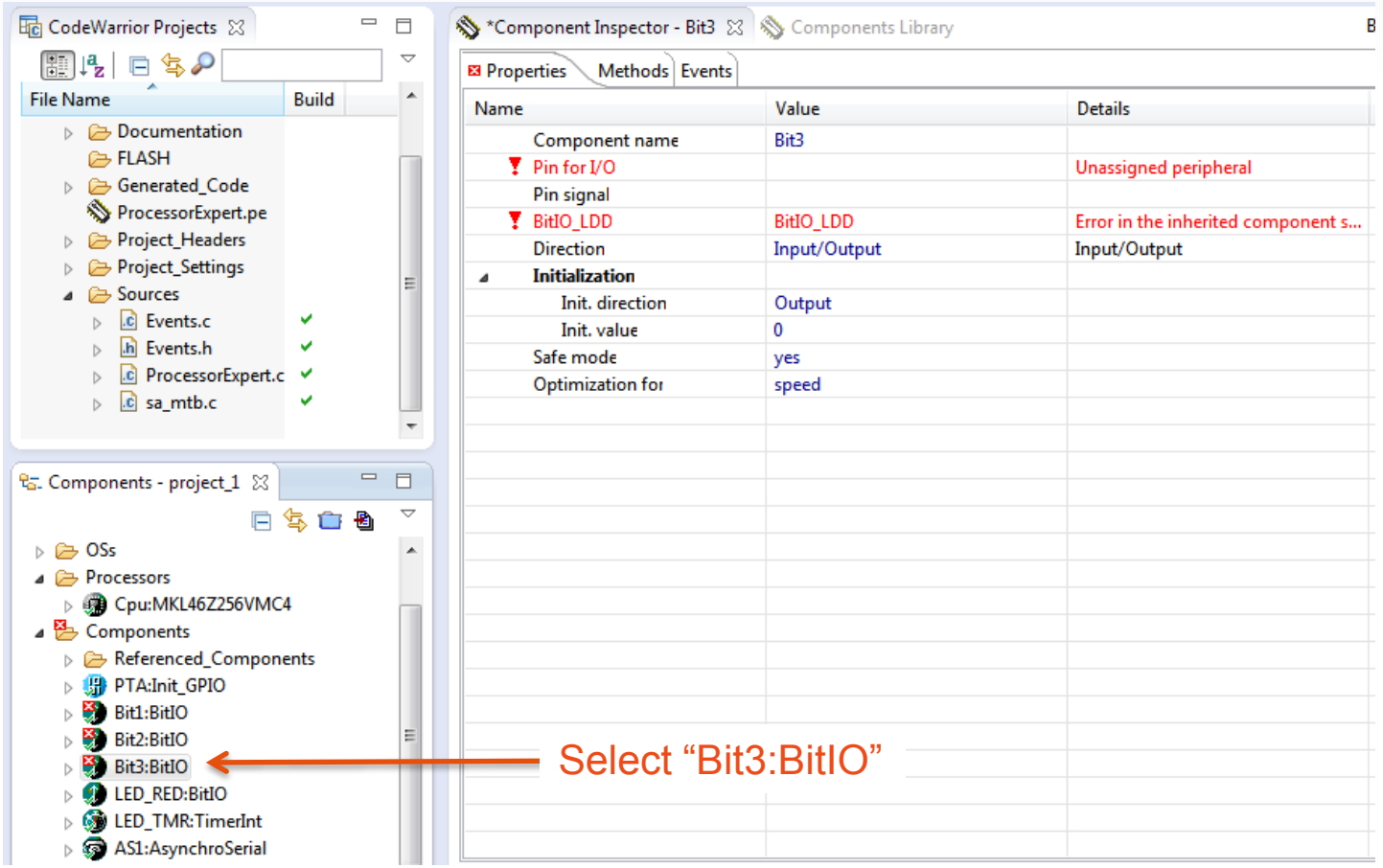
Peripheral Initialization – Bit4:BitIO



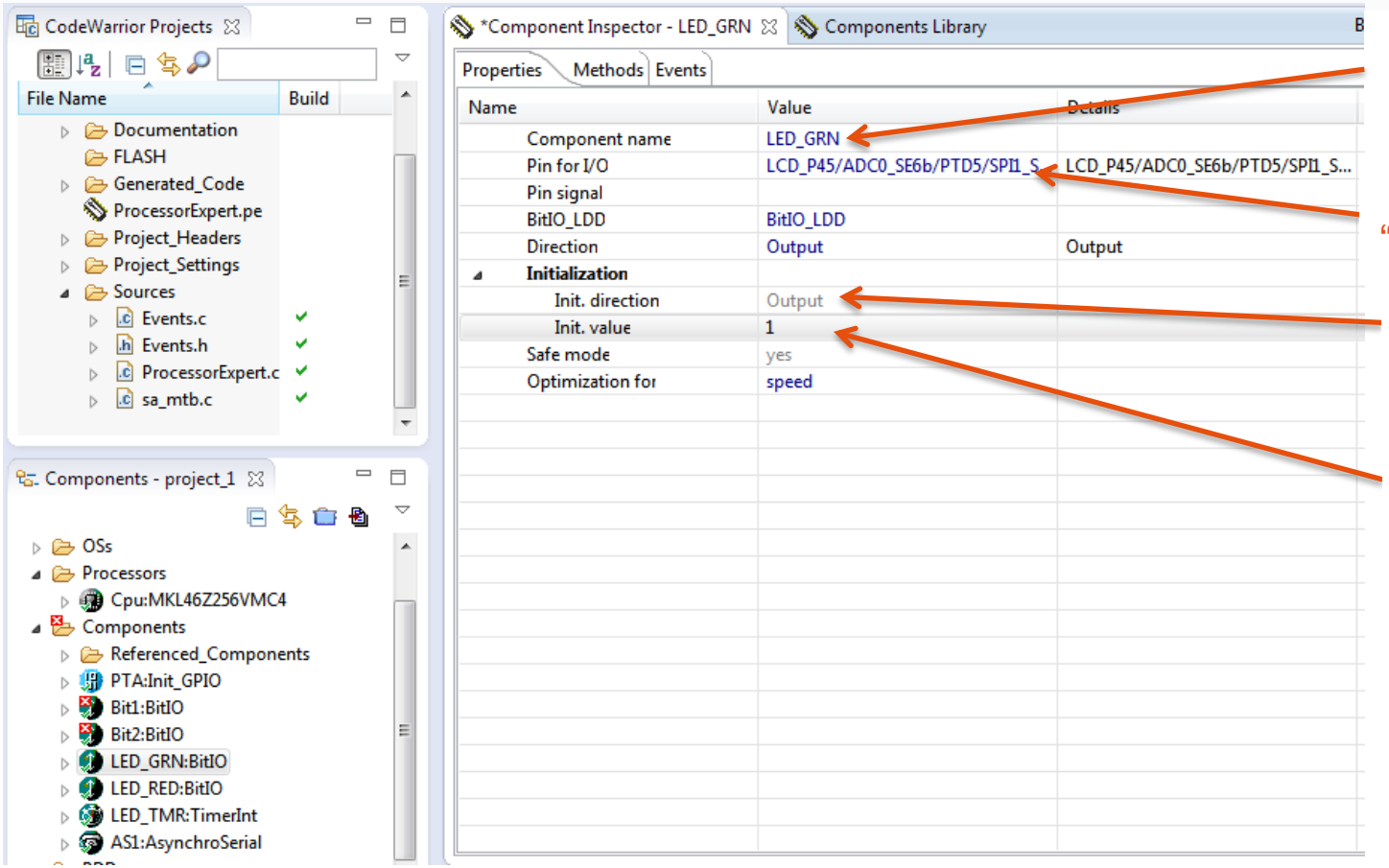
Peripheral Initialization – Bit4:BitIO



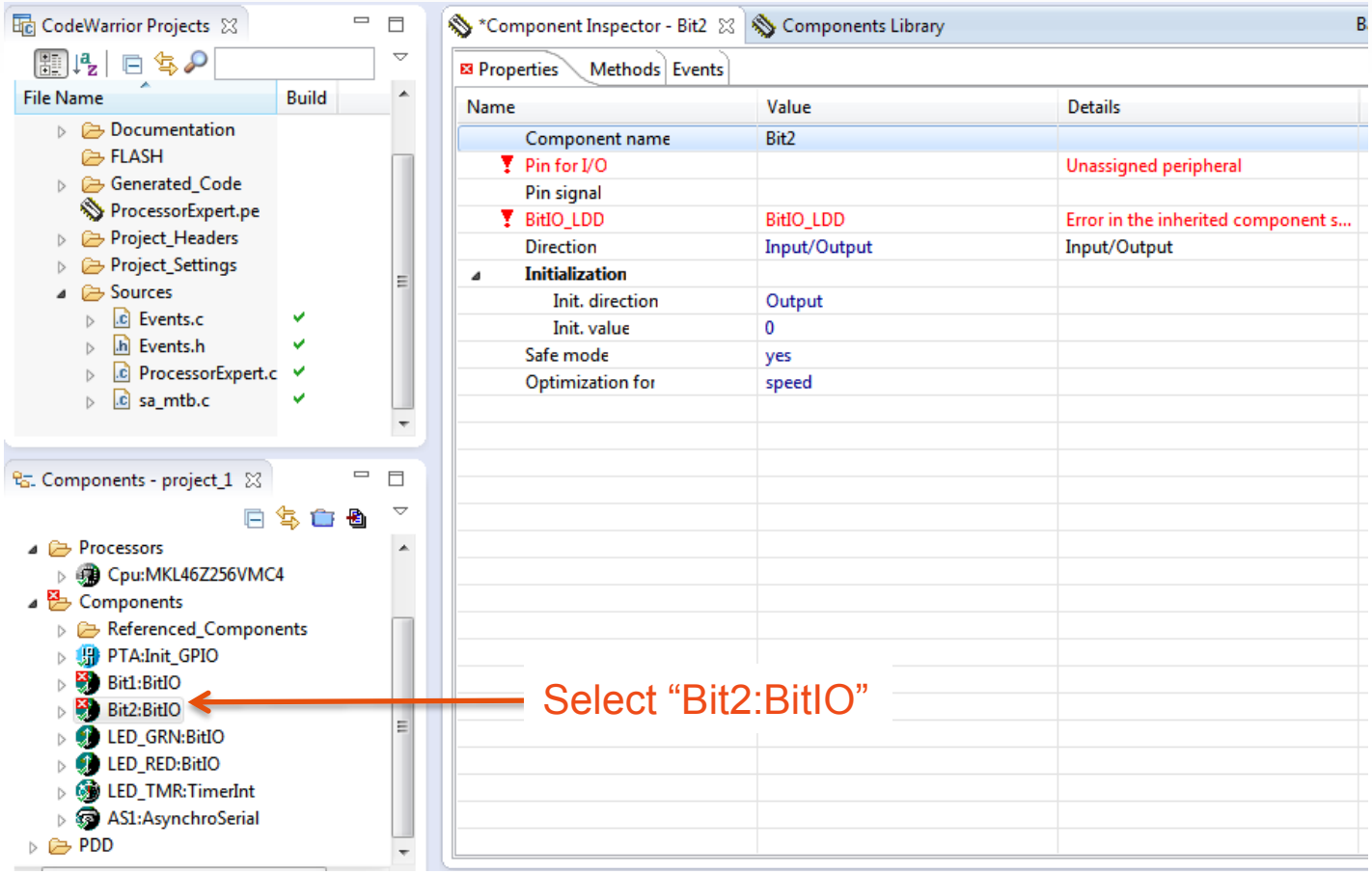
Peripheral Initialization – Bit3:BitIO



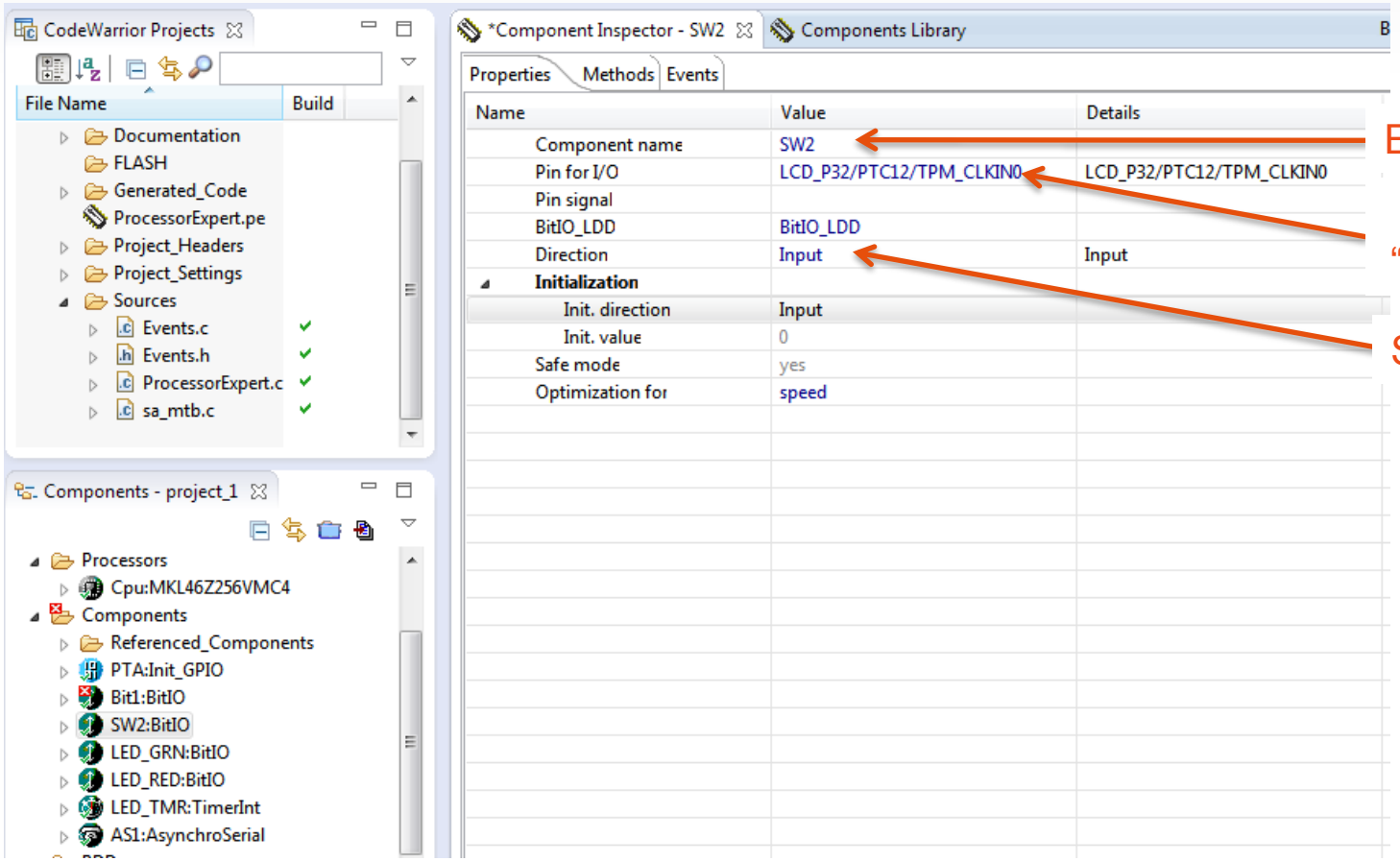
Peripheral Initialization – Bit3:BitIO



Peripheral Initialization – Bit2:BitIO



Peripheral Initialization – Bit2:BitIO

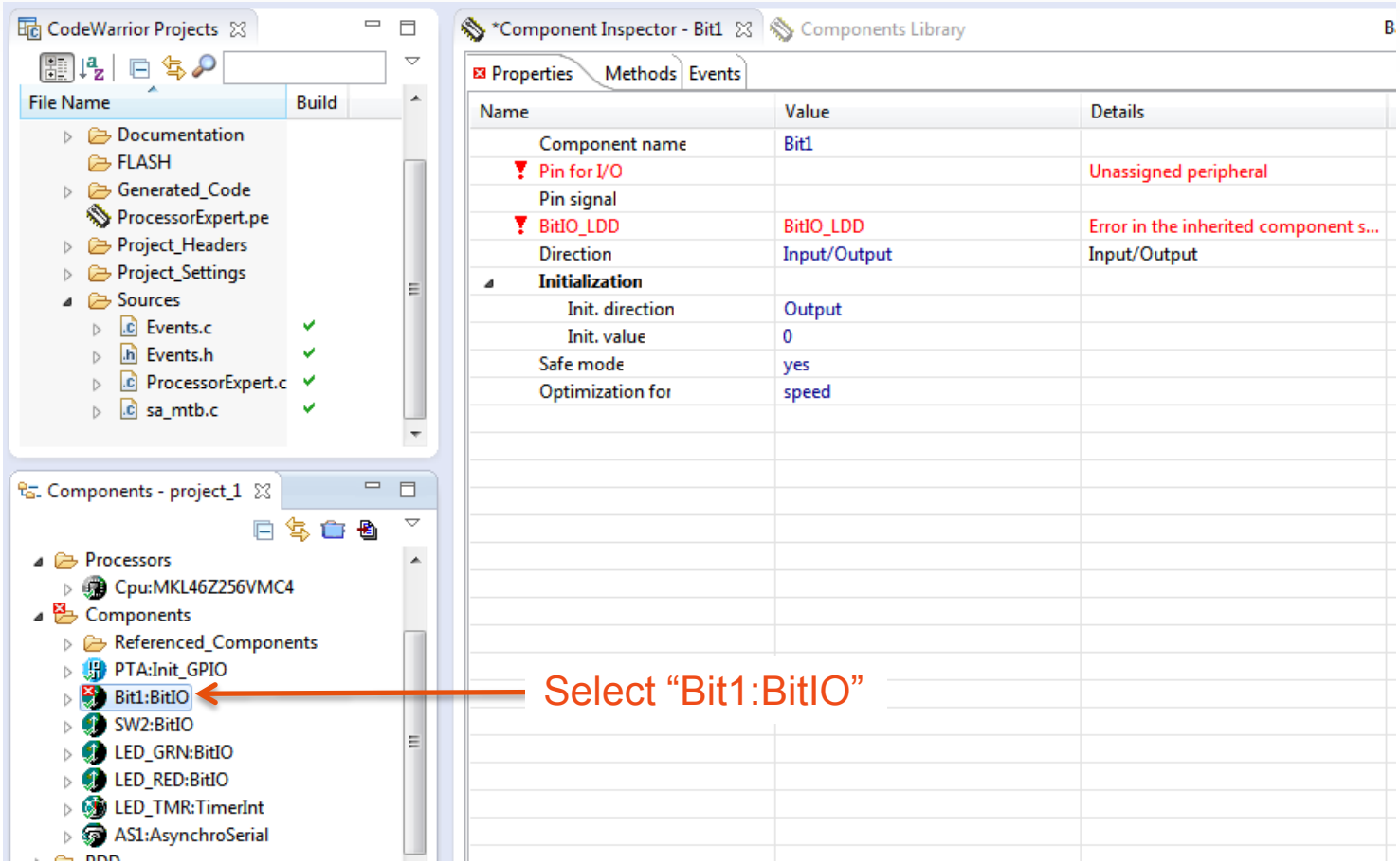


Enter “SW2”

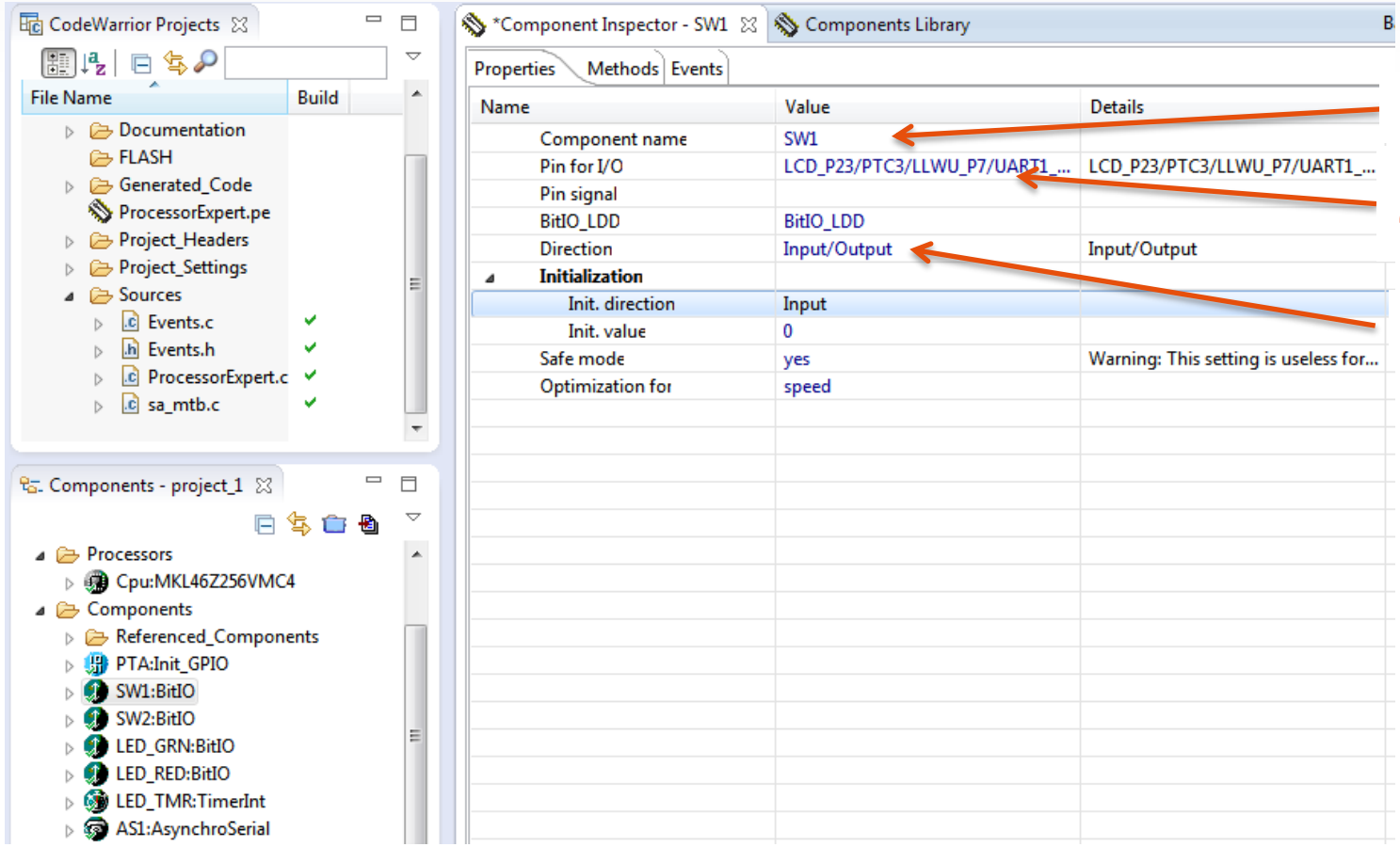
Select “LCD_P32/…”

Select “Input”

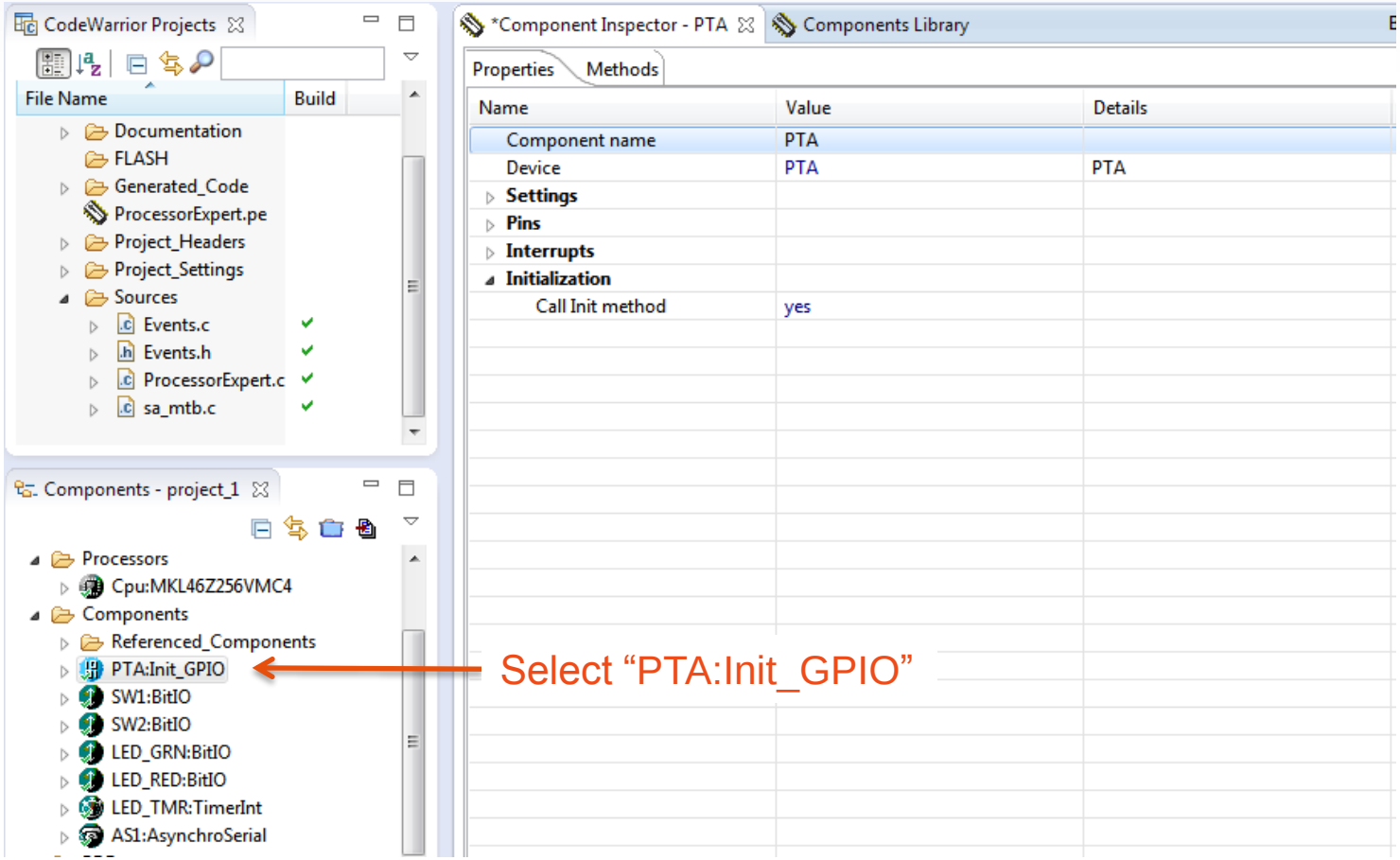
Peripheral Initialization – Bit1:BitIO



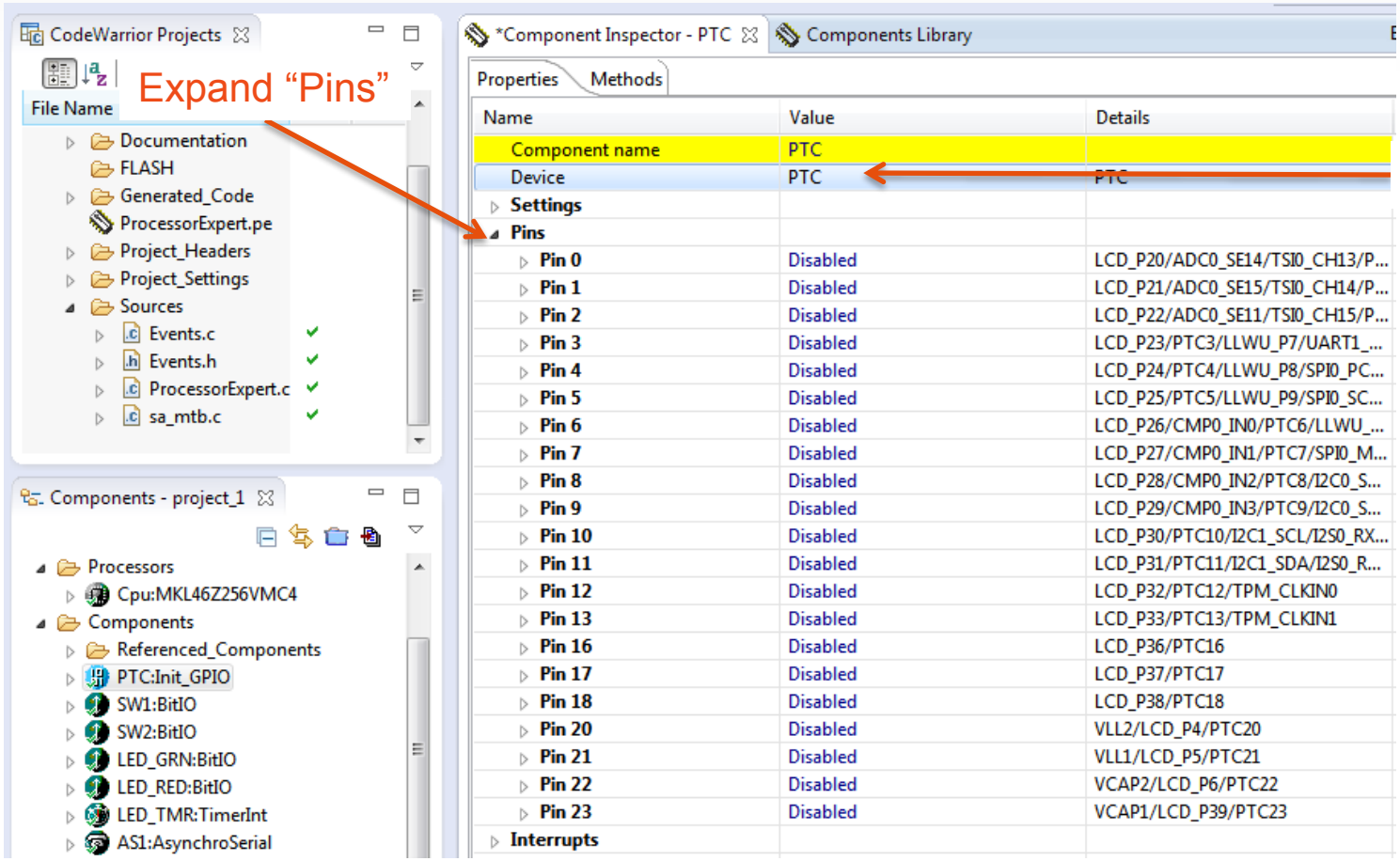
Peripheral Initialization – Bit1:BitIO



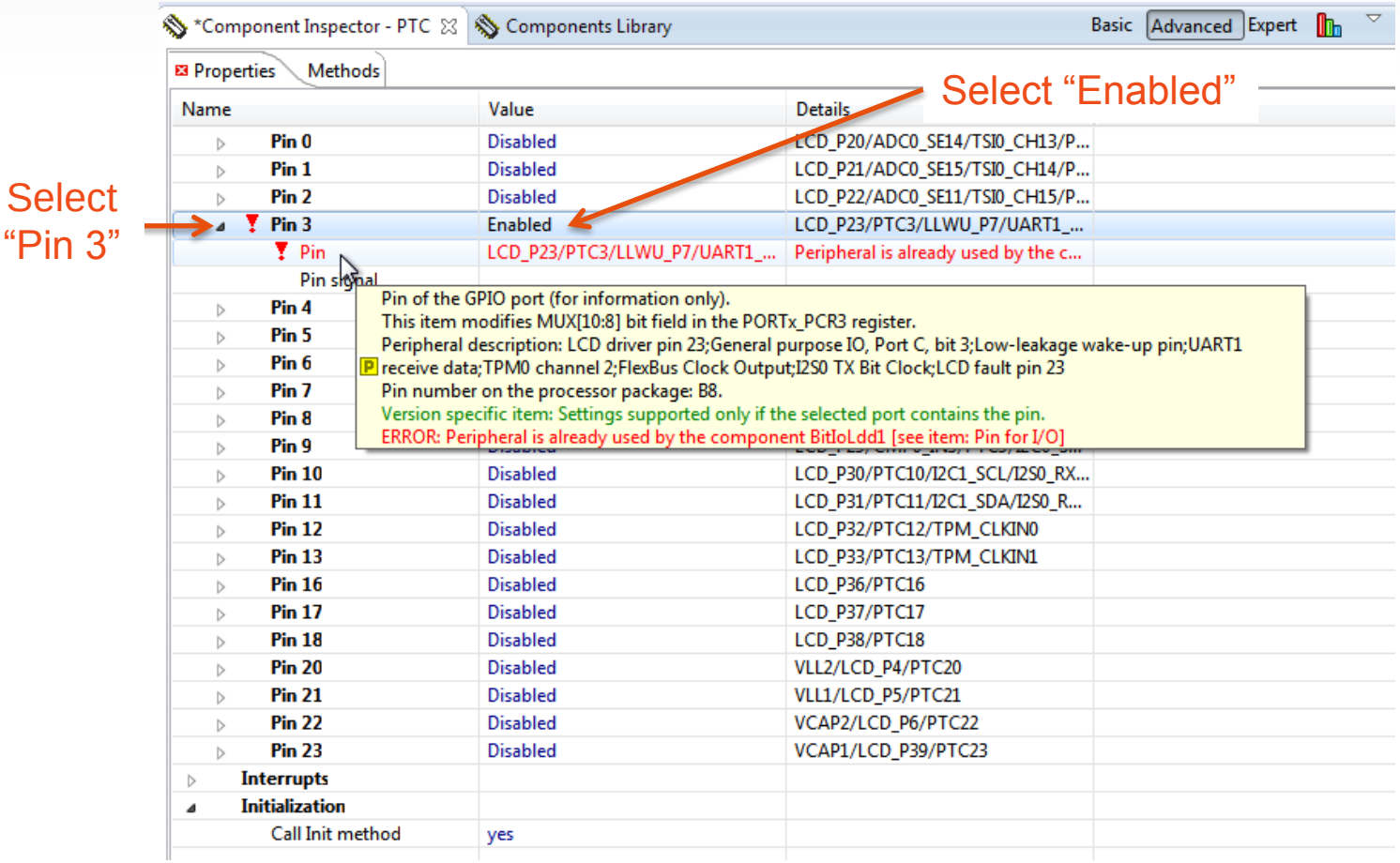
Peripheral Initialization – Init_GPIO



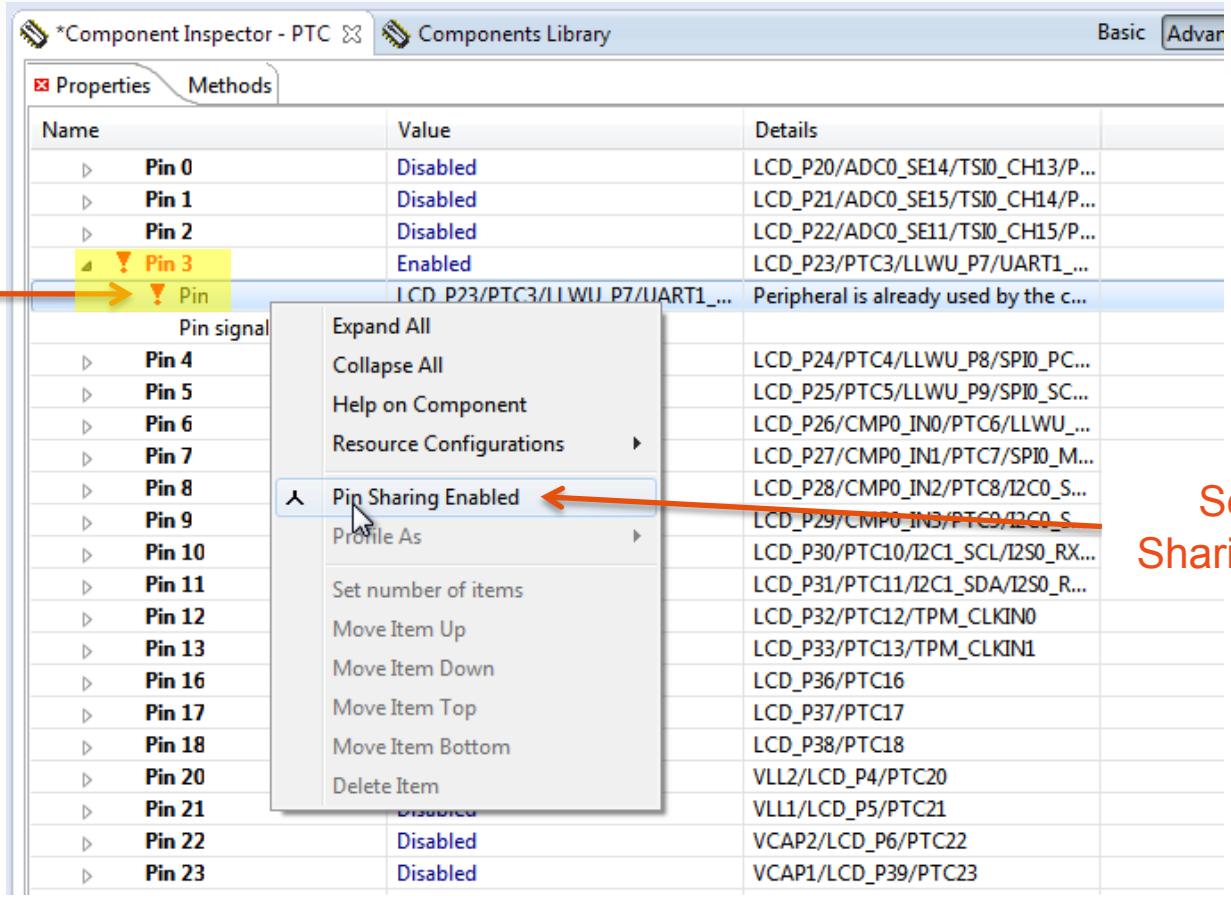
Peripheral Initialization – Init_GPIO



Peripheral Initialization – PTC:Init_GPIO

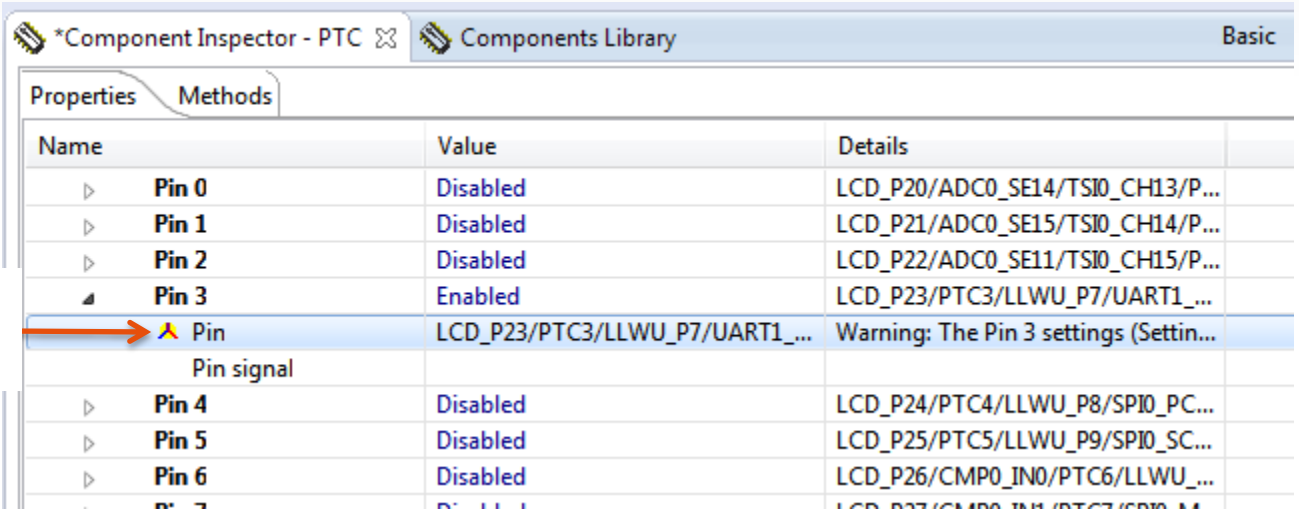


Peripheral Initialization – PTC:Init_GPIO

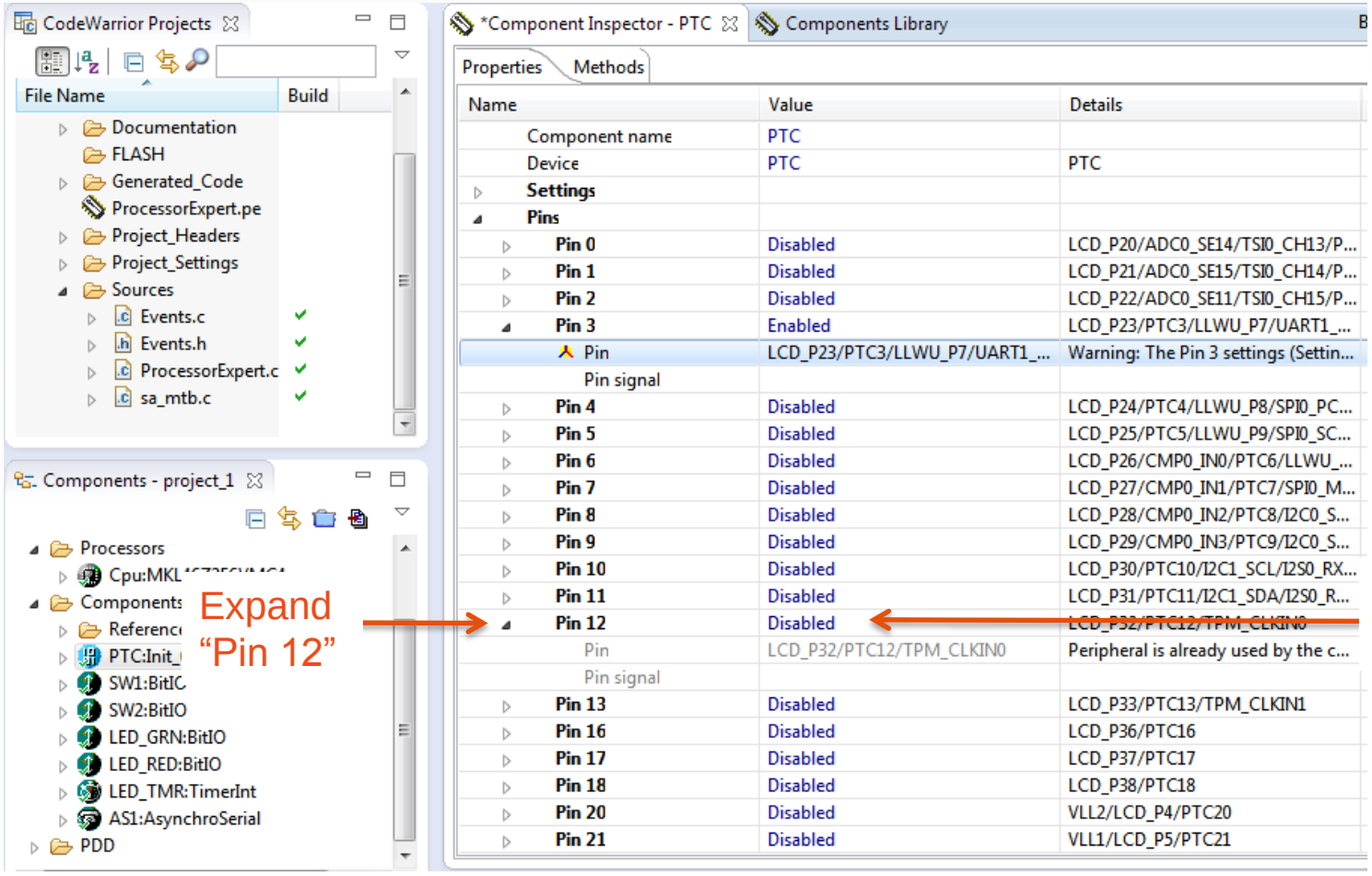


Peripheral Initialization – PTC:Init_GPIO

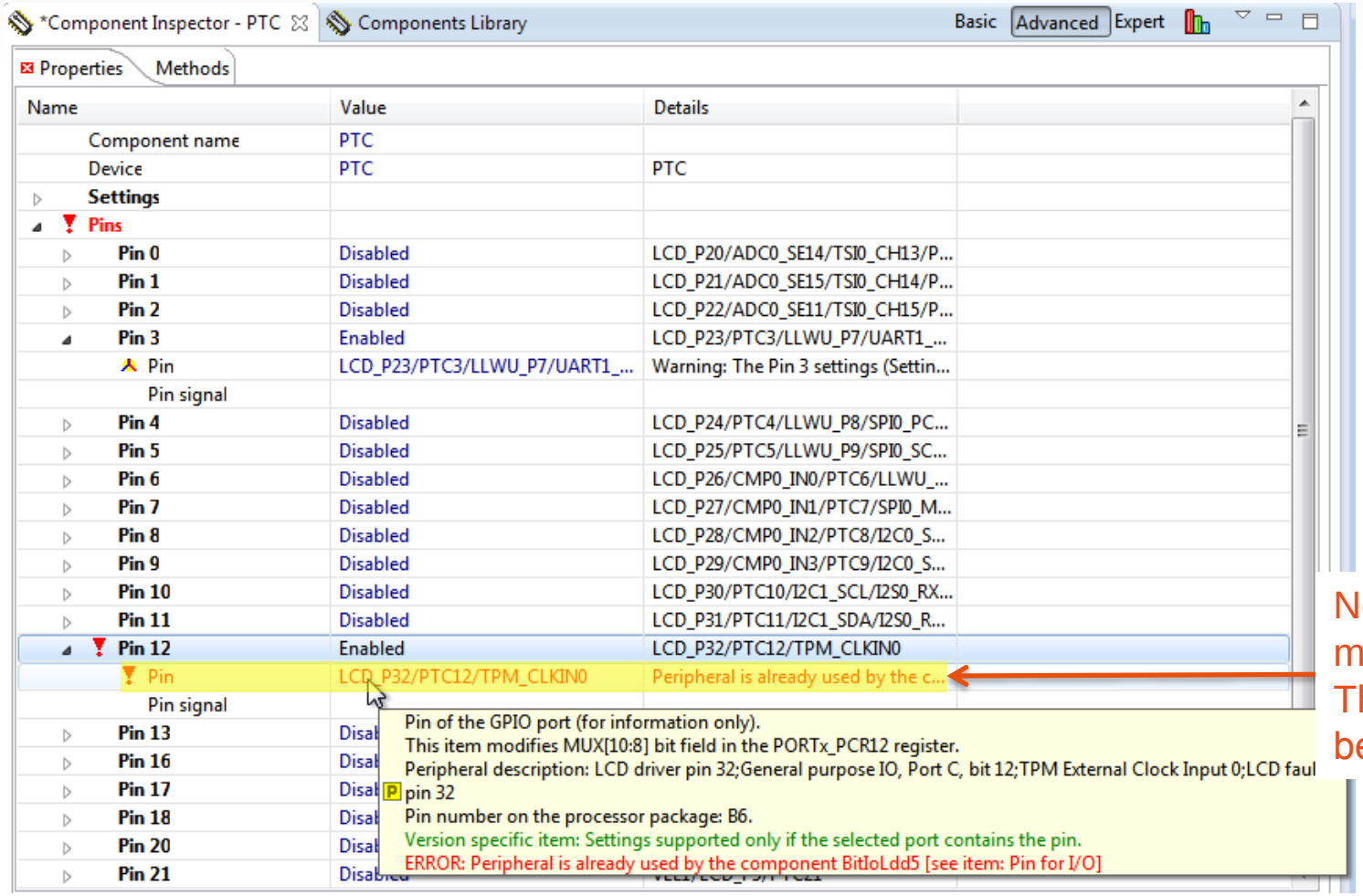
Pin sharing is enabled



Peripheral Initialization – PTC:Init_GPIO

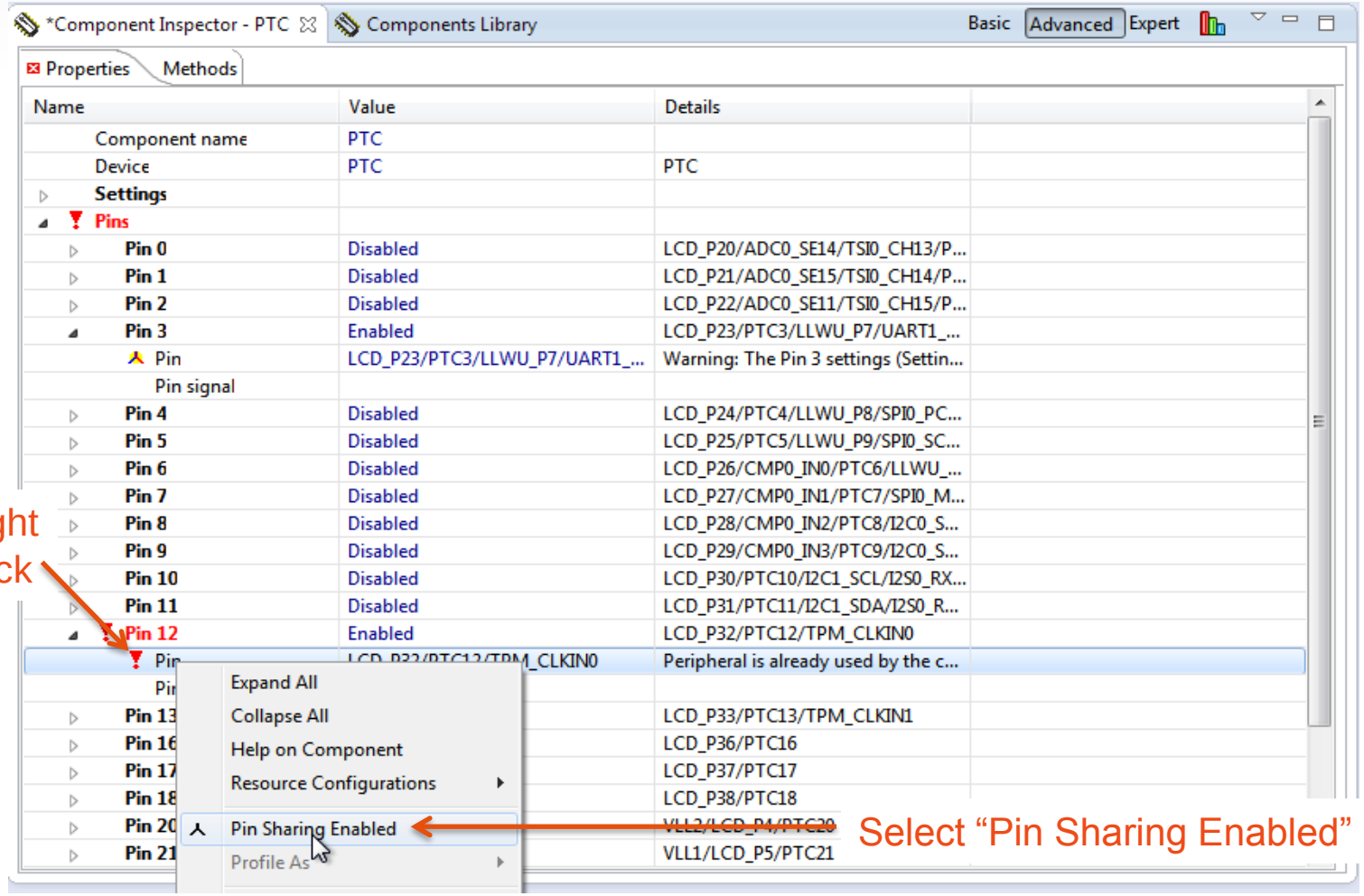


Peripheral Initialization – PTC:Init_GPIO



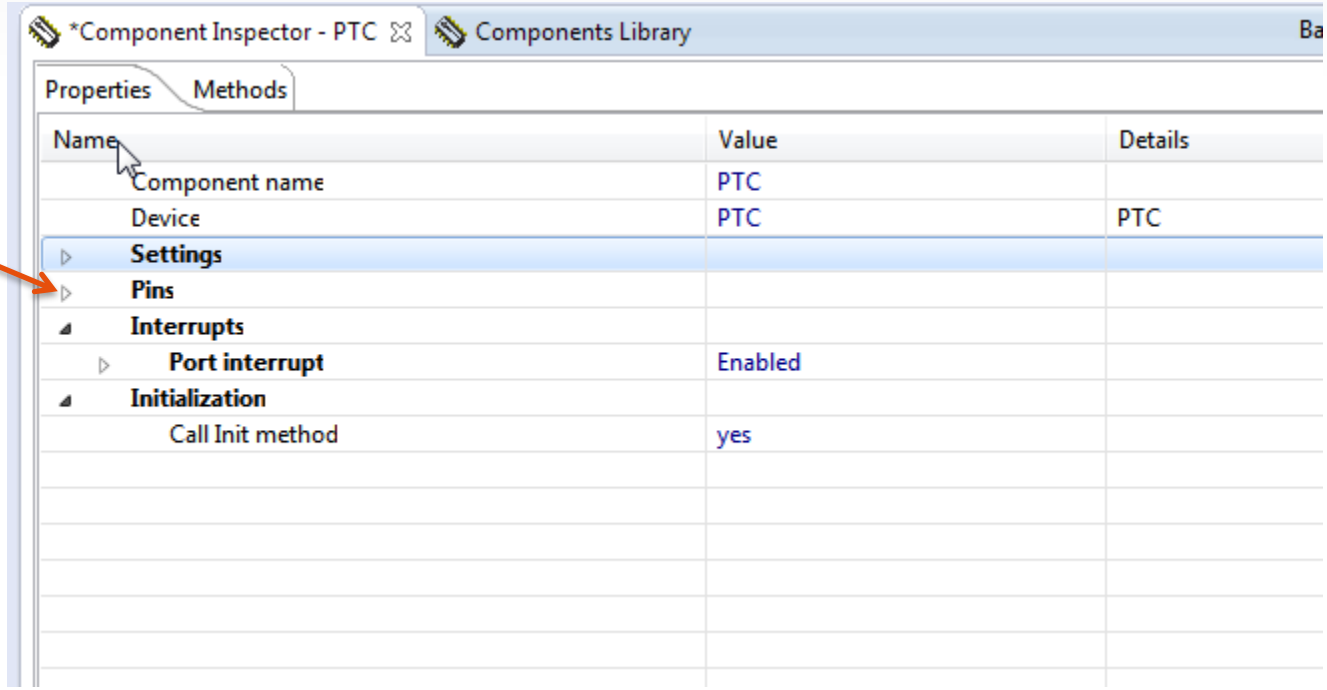
Notice error message. This pin must be “shared”

Peripheral Initialization – PTC:Init_GPIO



Peripheral Initialization – PTC:Init_GPIO

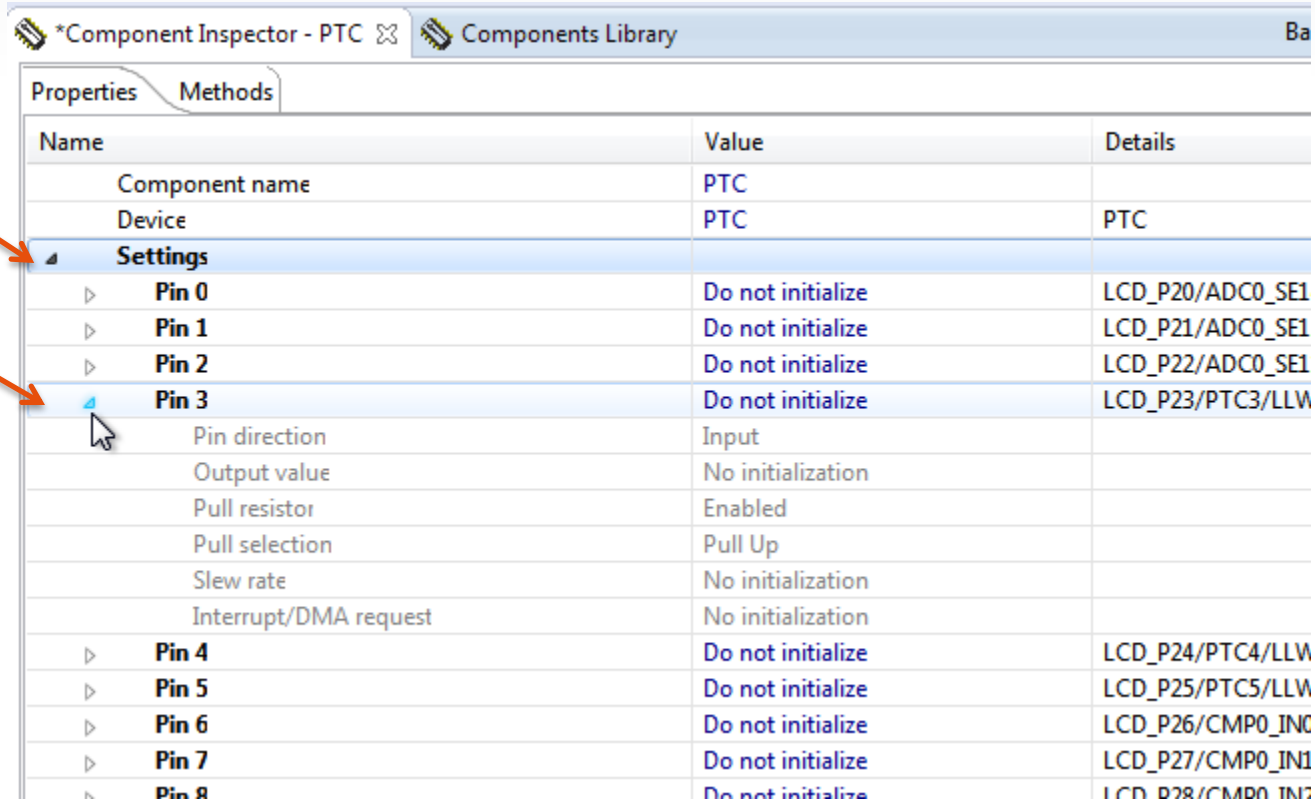
Close "Pins"



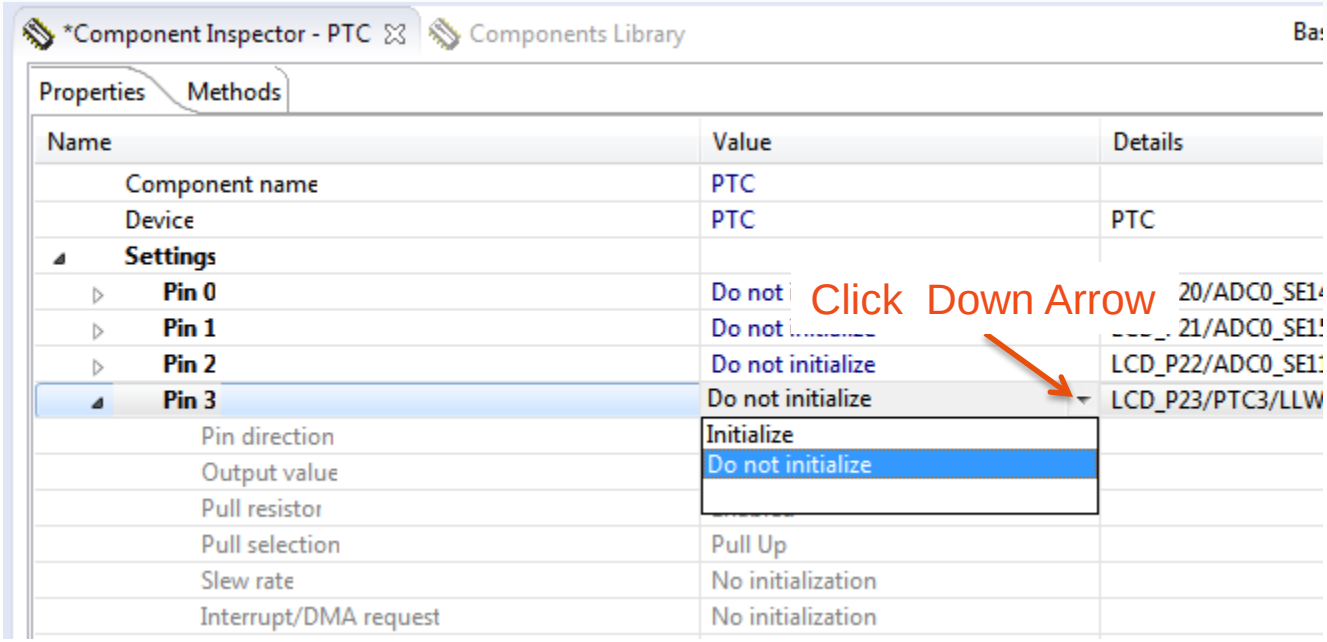
Peripheral Initialization – PTC:Init_GPIO

Expand "Setting"

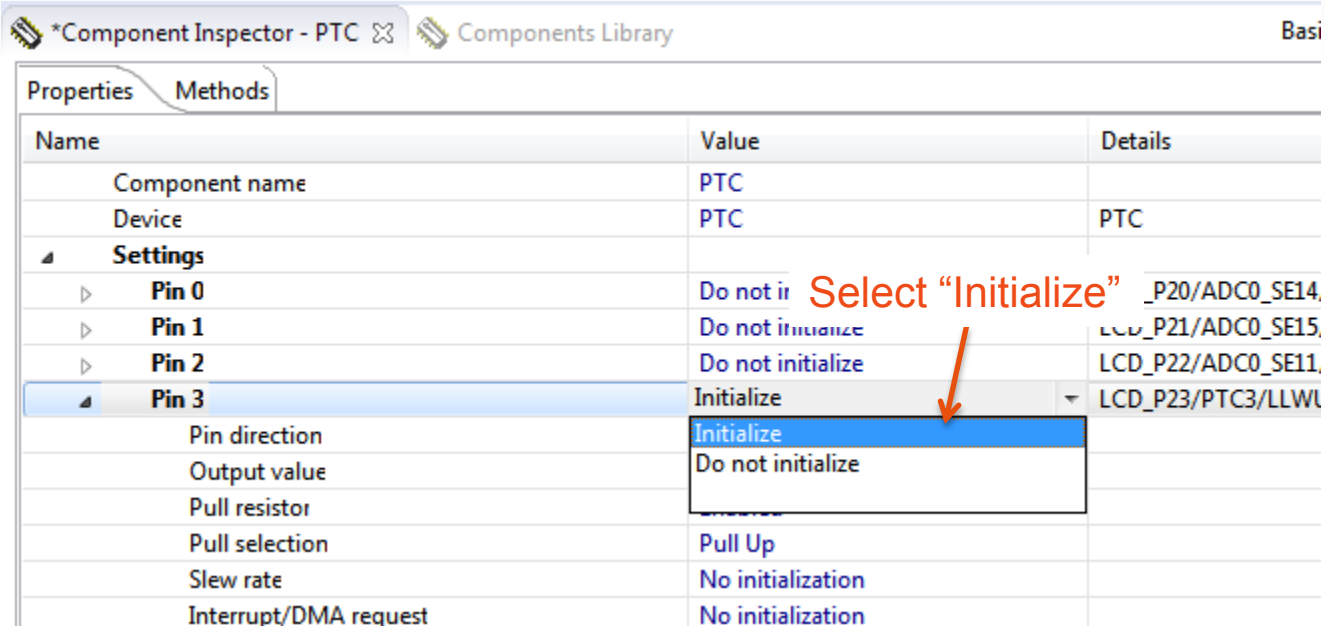
Expand "Pin 3"



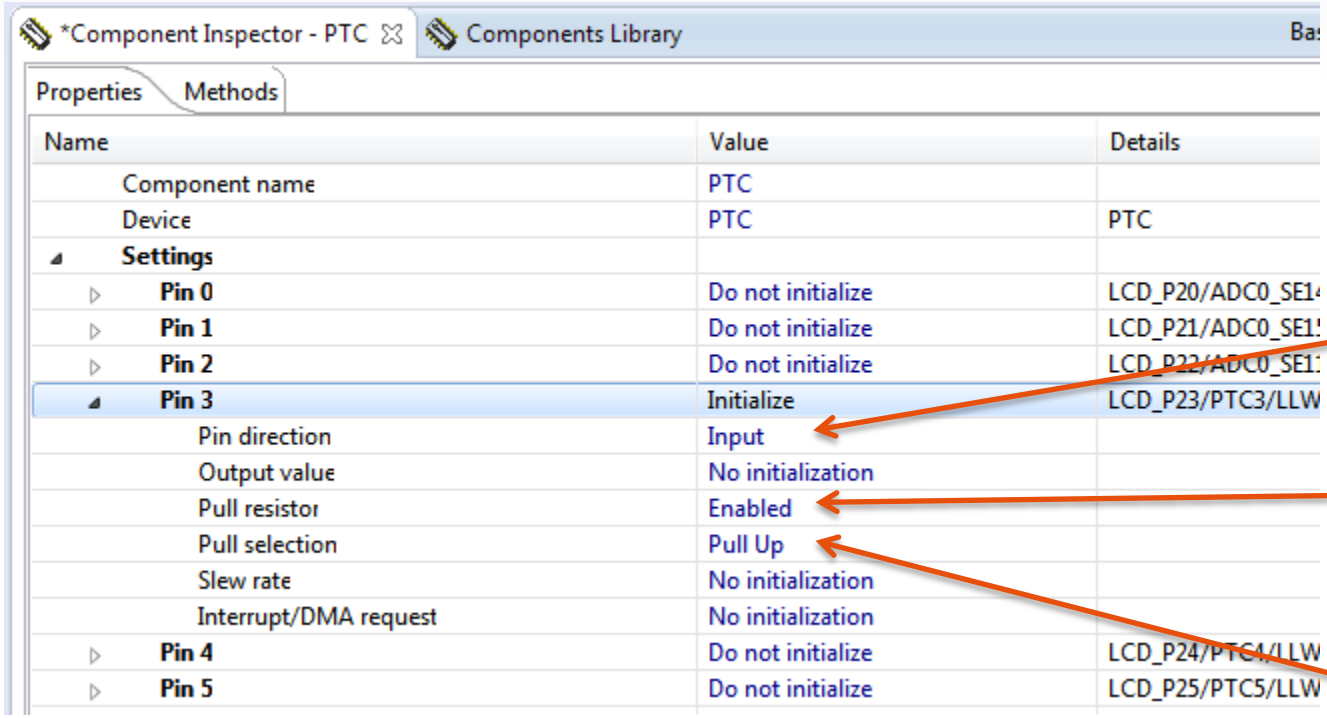
Peripheral Initialization – PTC:Init_GPIO



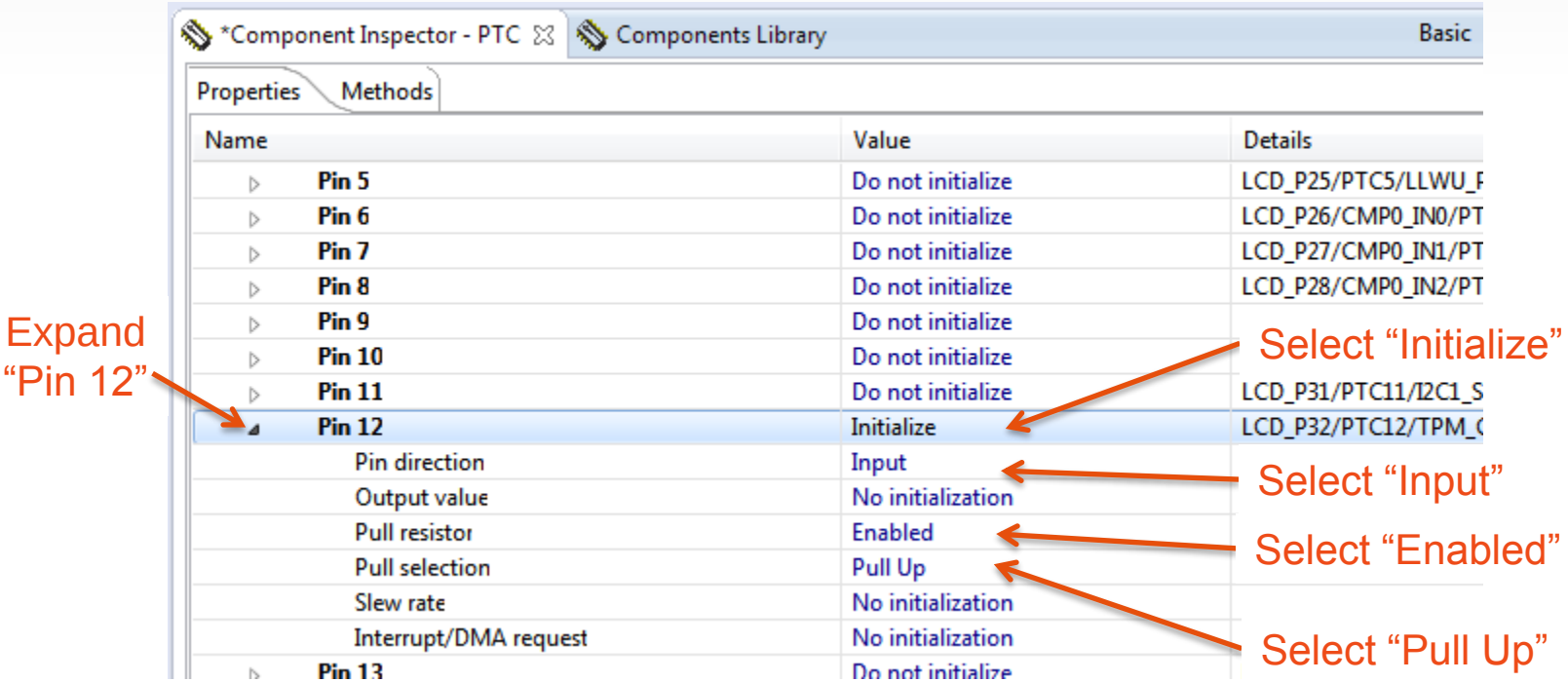
Peripheral Initialization – PTC:Init_GPIO



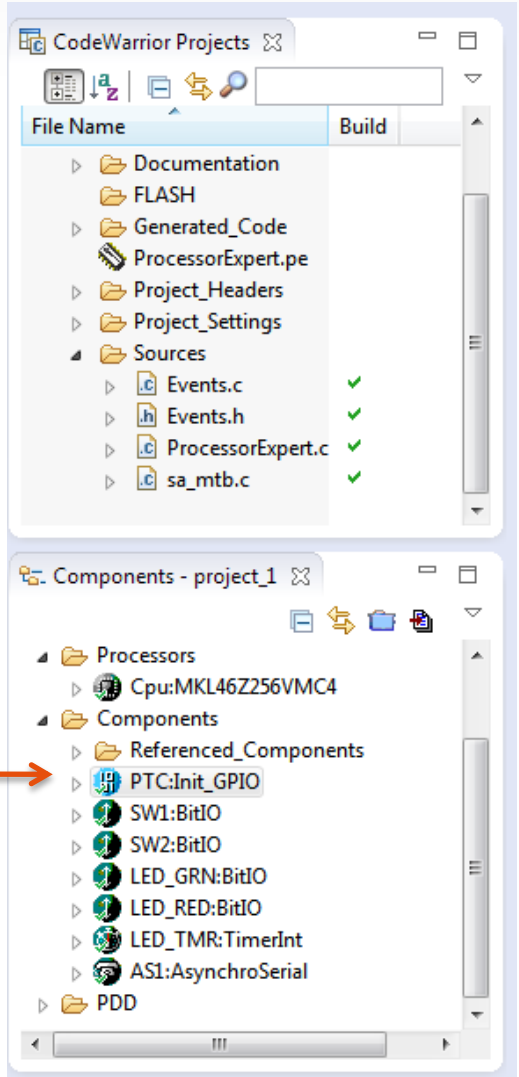
Peripheral Initialization – PTC:Init_GPIO



Peripheral Initialization – PTC:Init_GPIO



Peripheral Initialization



No Errors

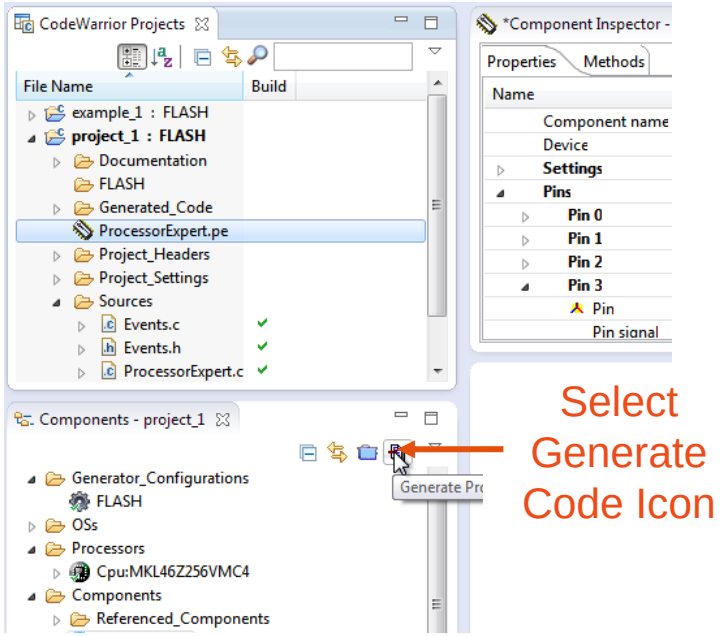
Check Point: Create a New Project

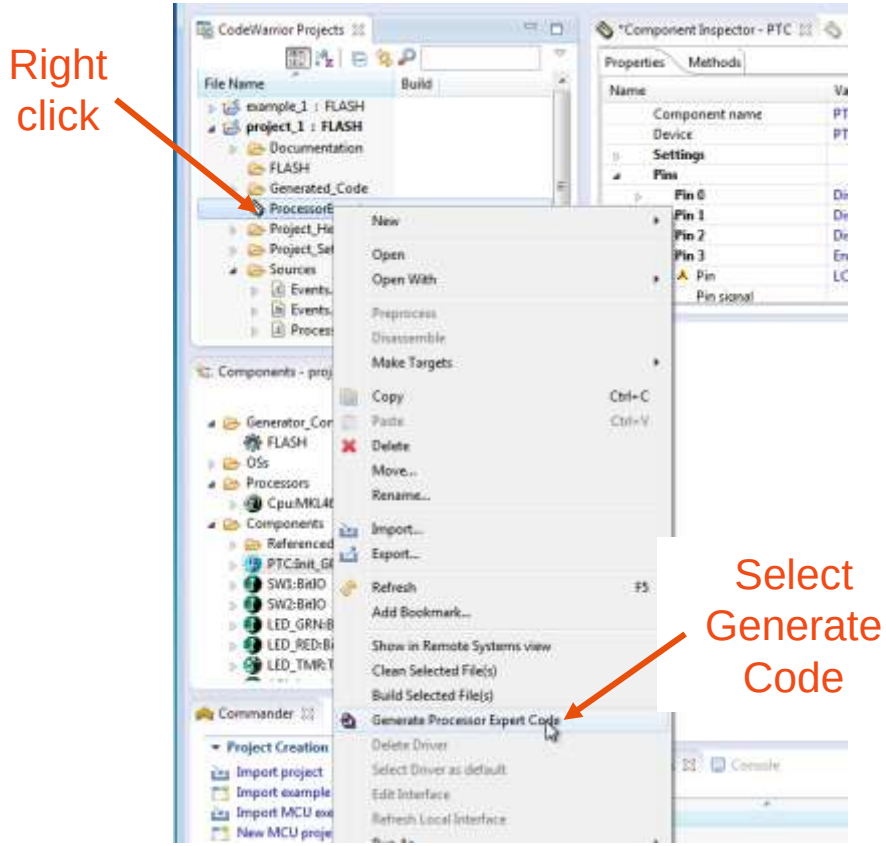
- This hands-on lab shows you how to…
  - Create a new project with the New Project Wizard ✓
  - Select and setup High Level Components ✓
  - **Generate Processor Expert Code**
  - Import existing files
  - Build the project
  - Test the application's functionality

Next up!

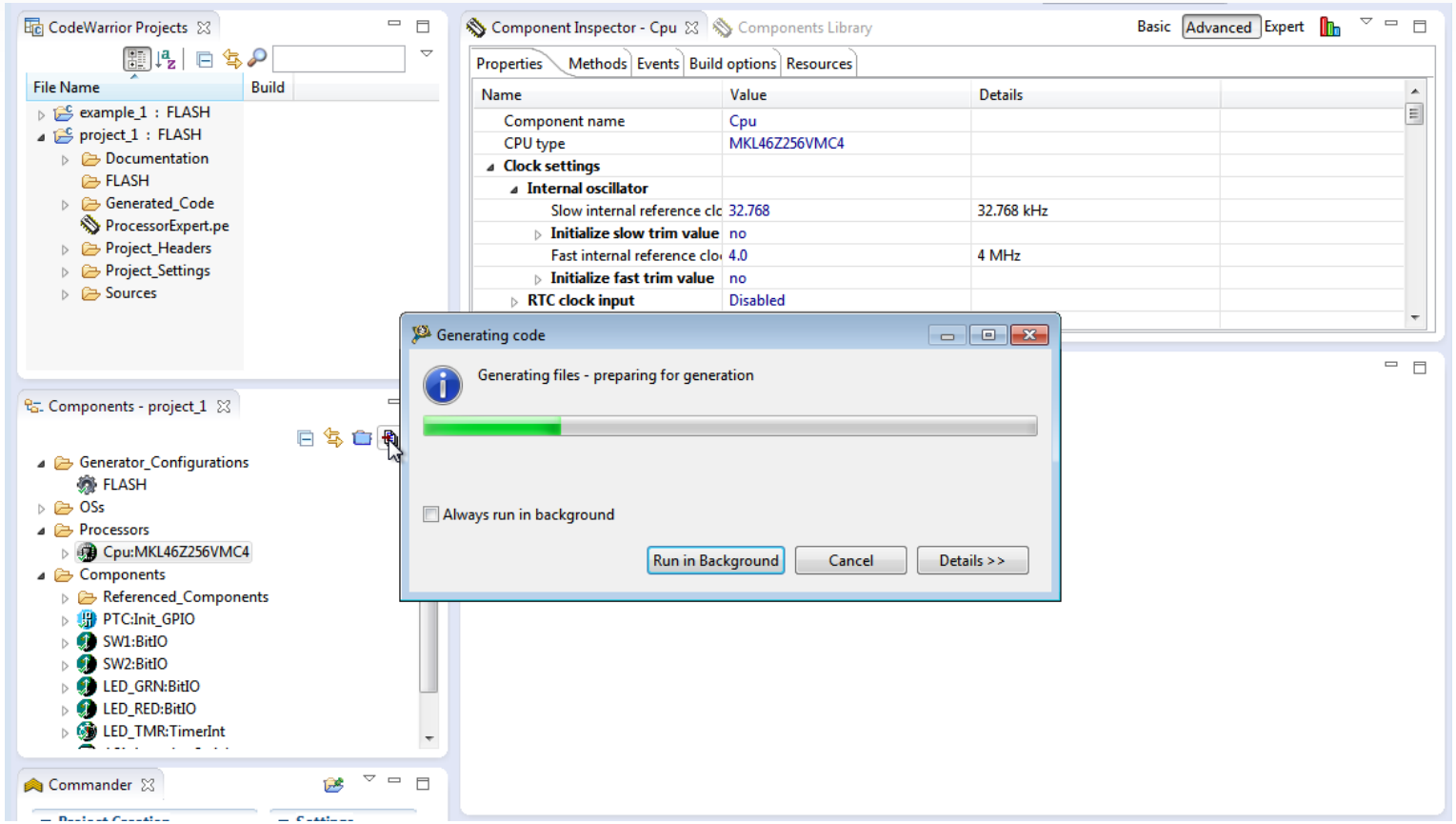
Generate Code

- Now that the components have been setup, its time to generate code. This can be done in one of two ways…





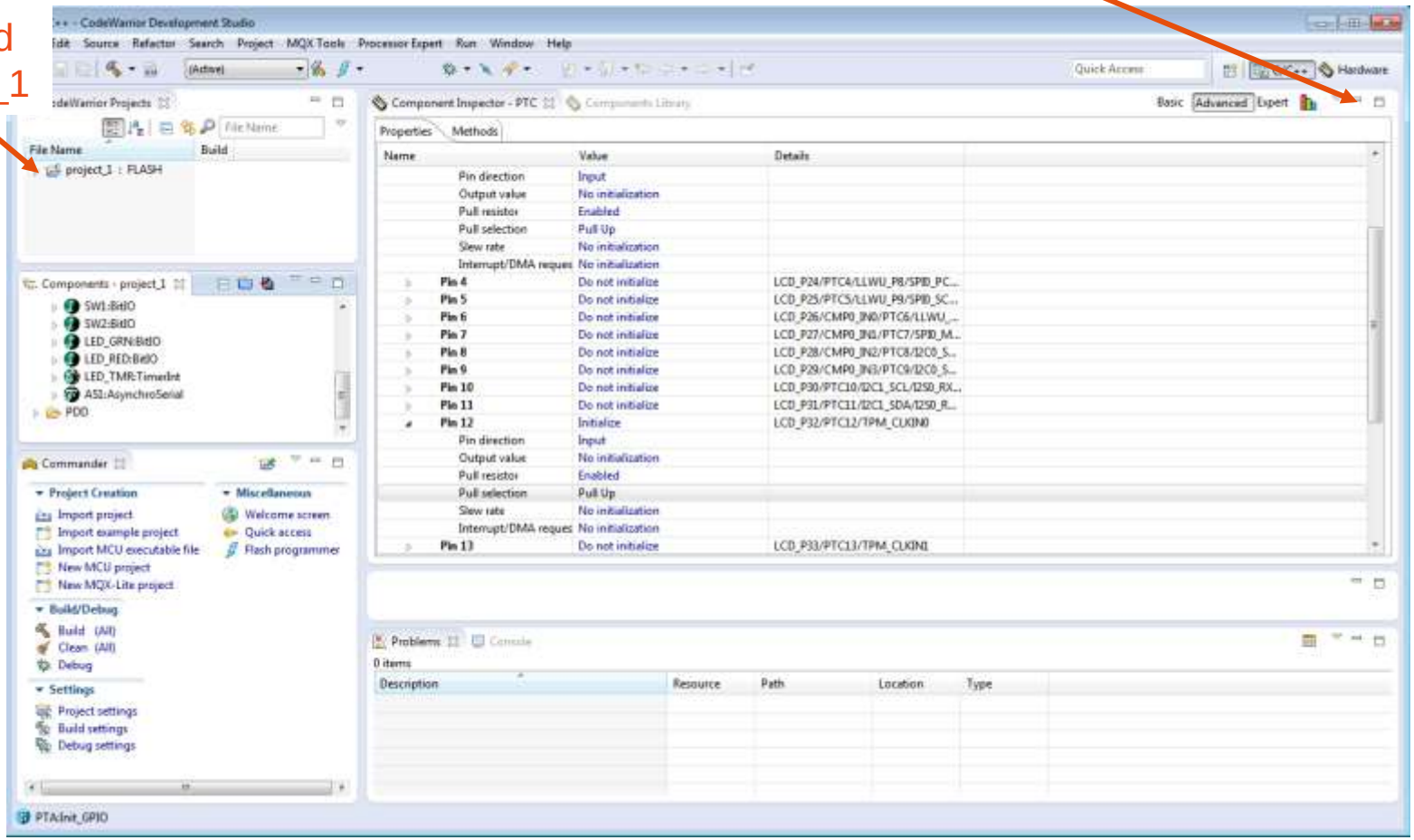
Code is being generated



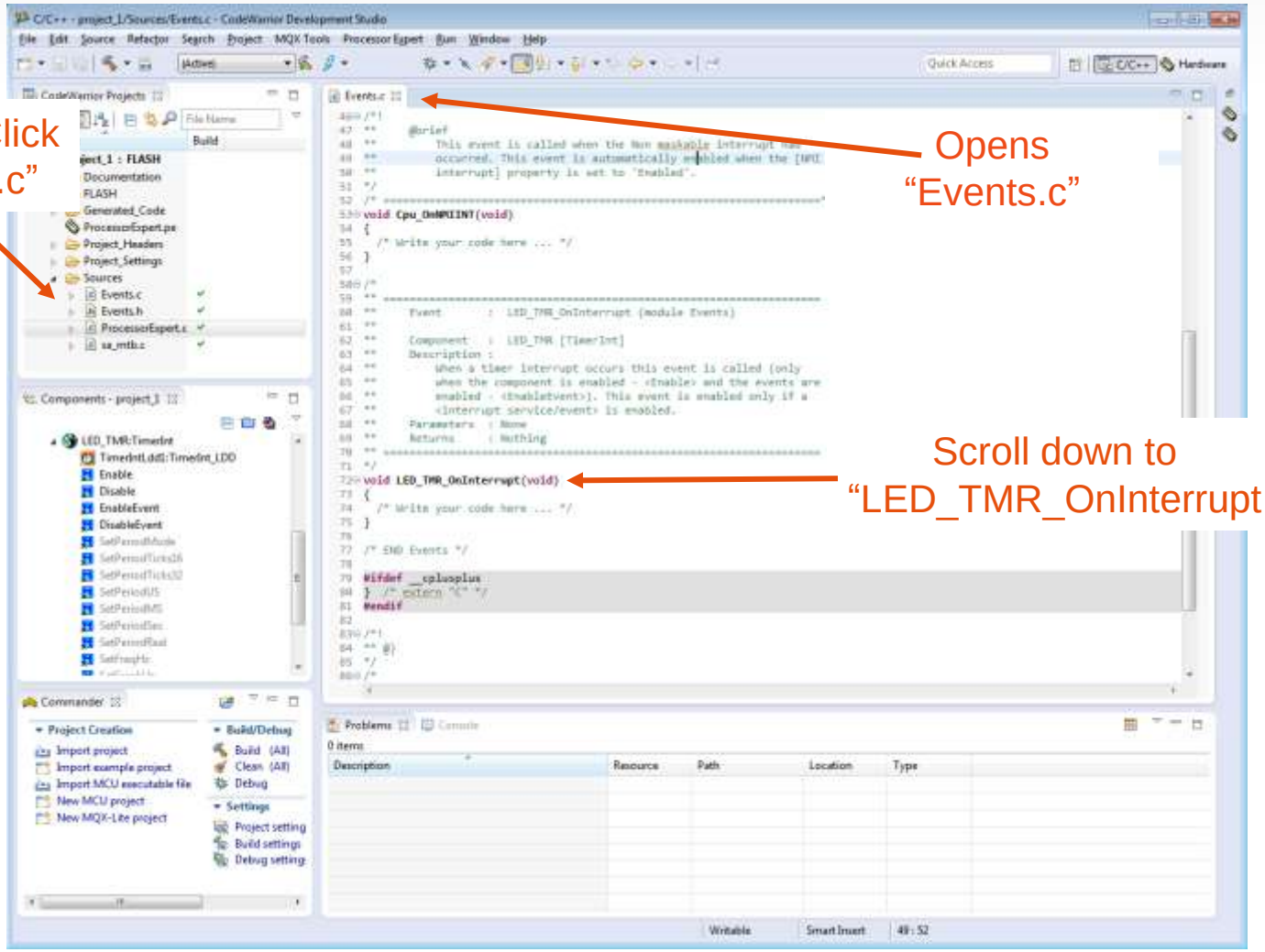
Using PEx Capability

Minimize
Processor Expert View

Expand
Project_1



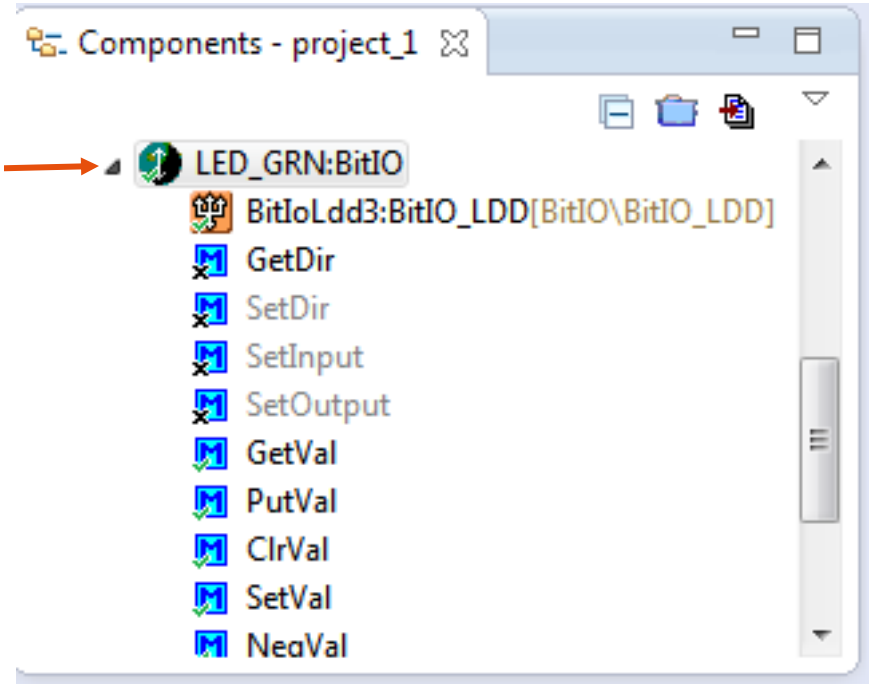
Using PEx Capability



Using PEx Capability

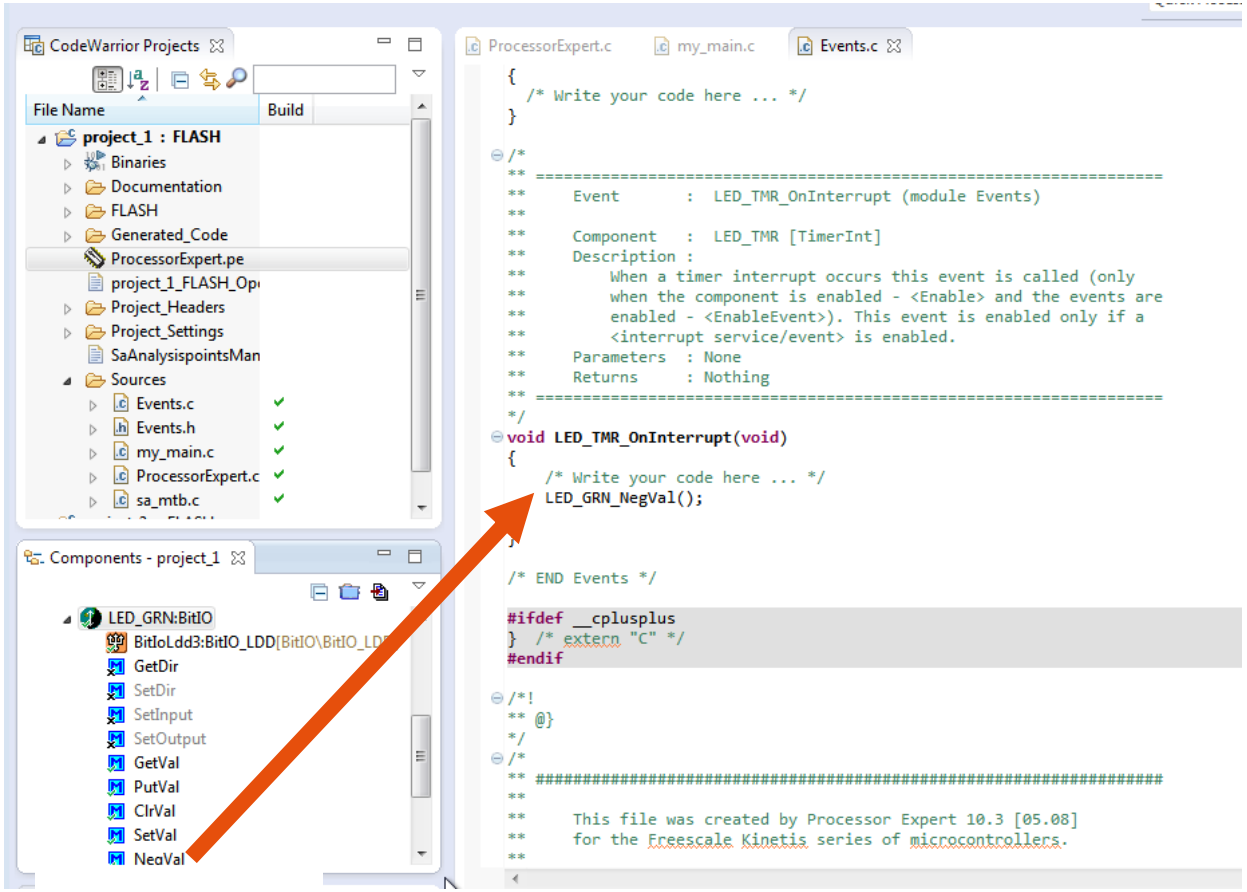
- Add code to the Timer Interrupt - LED_TMR_OnInterrupt()

Expand
“LED_GRN:BitIO



Using PEx Capability

- Add code to the Timer Interrupt - LED_TMR_OnInterrupt()



Drag & Drop
"NegVal"

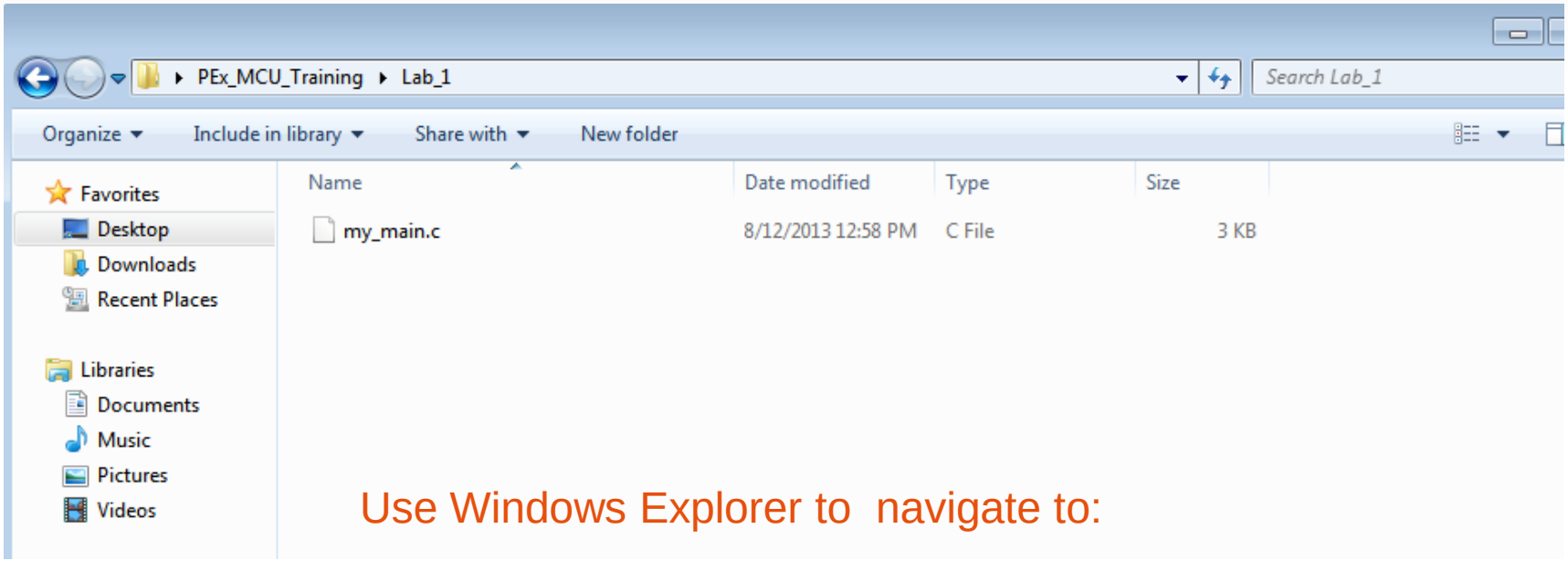
Check Point: Create a New Project to Blink an LED

- This hands-on lab shows you how to…
  - Create a new project with the New Project Wizard ✓
  - Select and setup High Level Components ✓
  - Generate Processor Expert Code ✓
  - **Import existing files**
  - Build the project
  - Test the application's functionality

Next up!

Importing Files

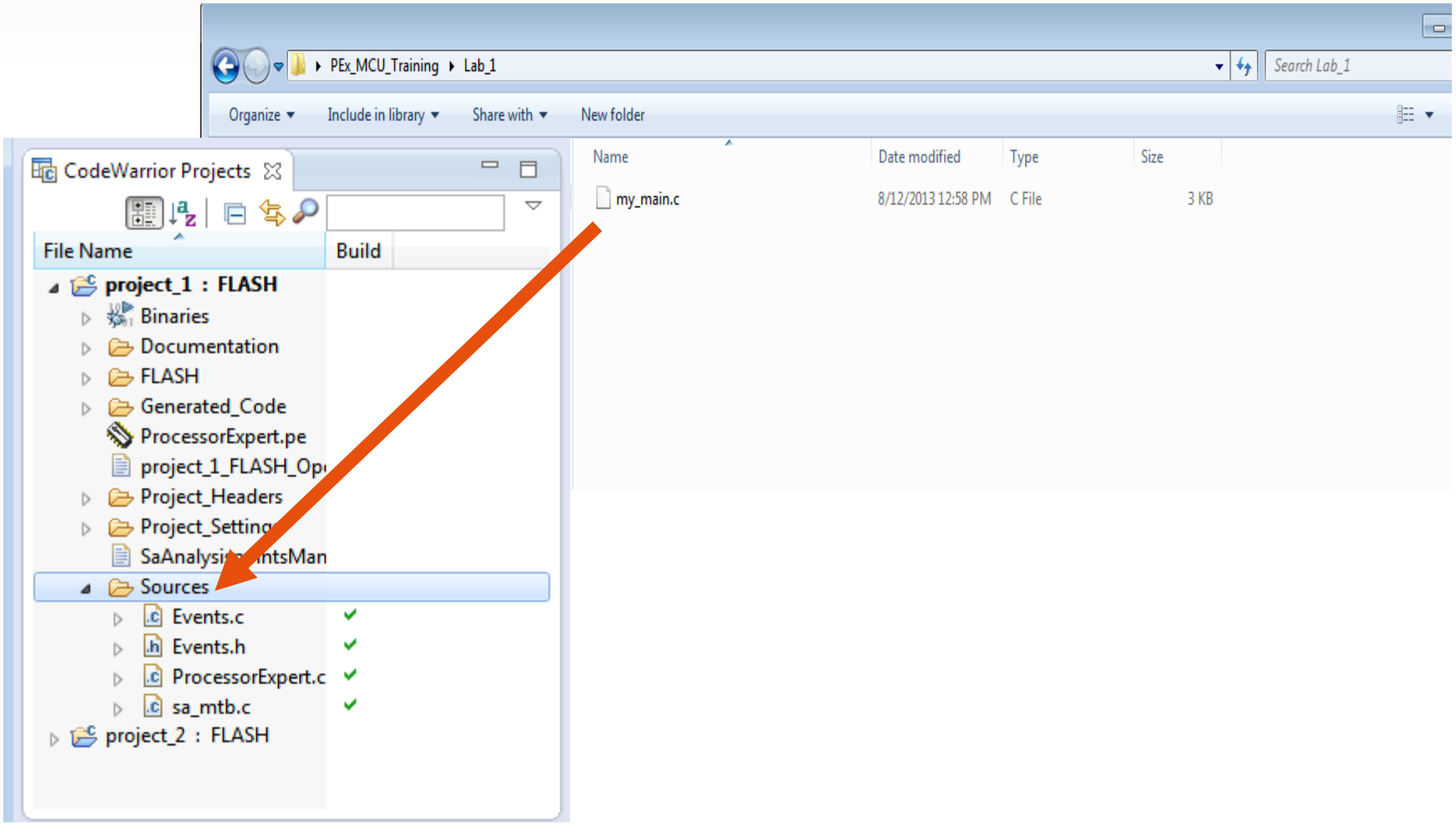
Drag and Drop



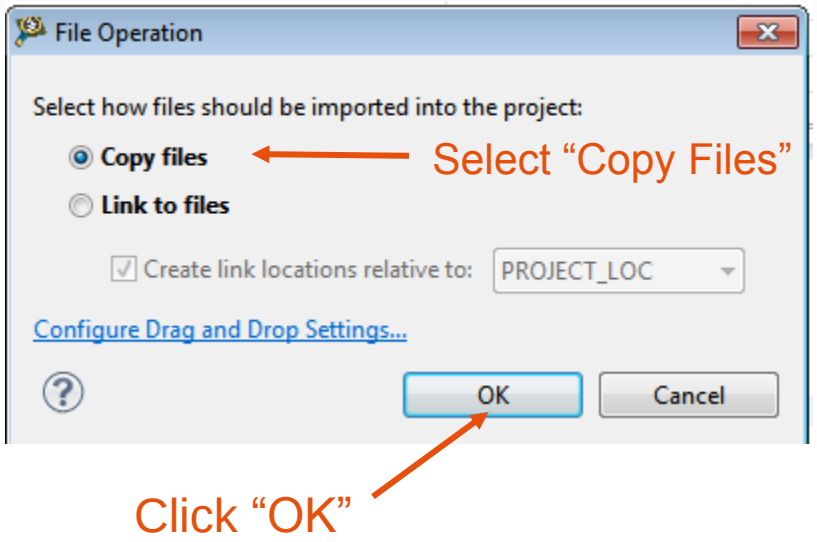
Use Windows Explorer to navigate to:

..\Desktop\PEx_MCU Training\Lab_1

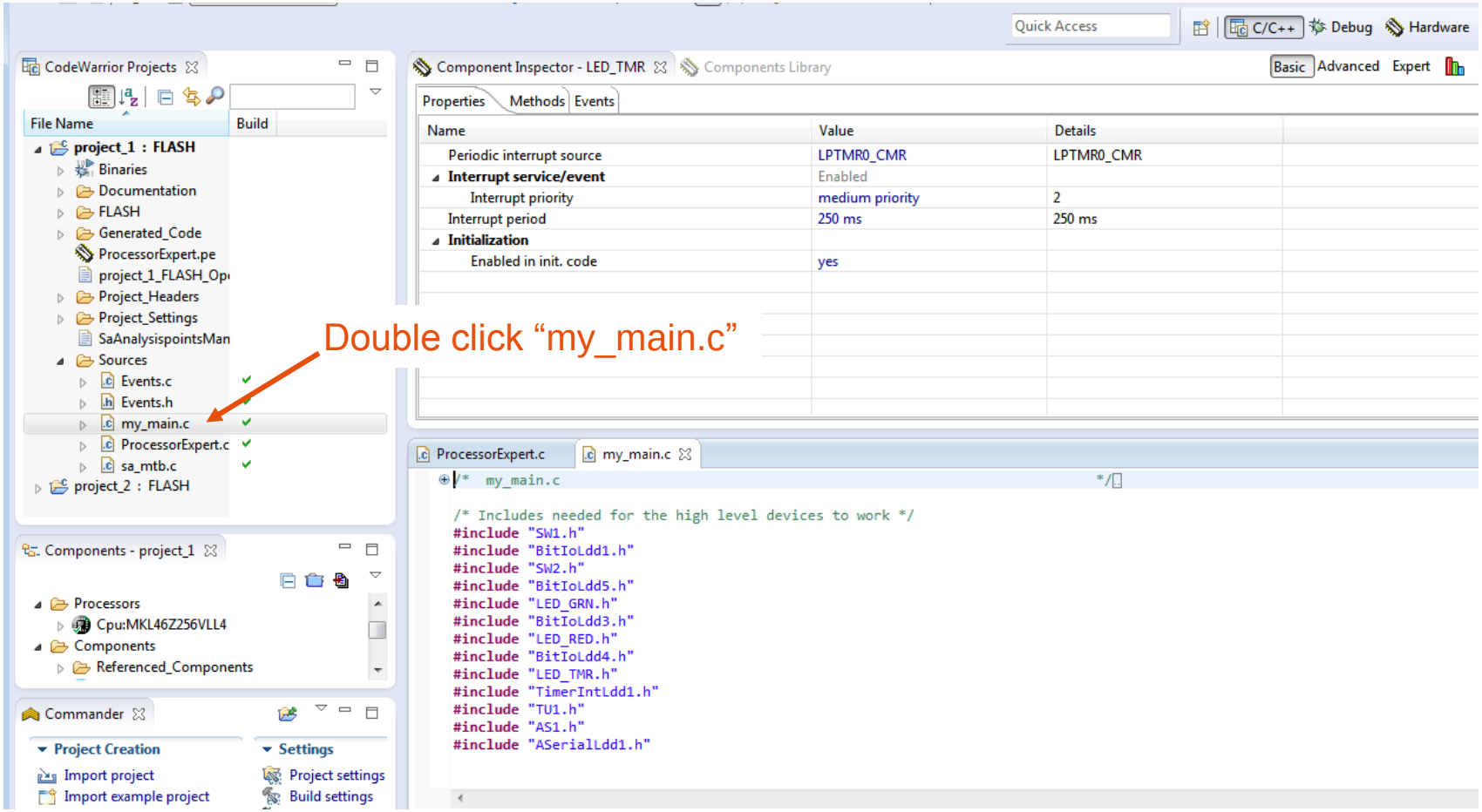
Drag my_main.c to Sources Folder



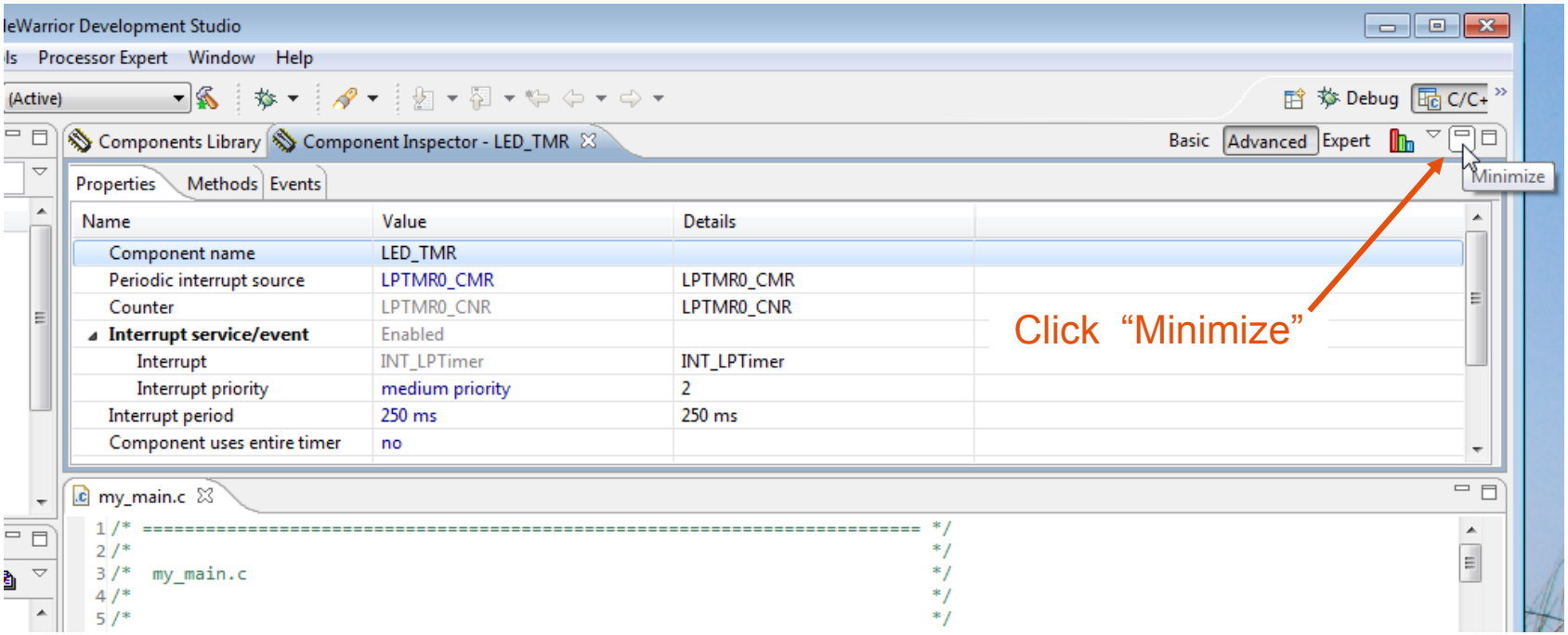
Drag & Drop


File Operation
Select how files should be imported into the project:
Copy files
Link to files
Create link locations relative to:
PROJECT_LOC
Configure Drag and Drop Settings...
OK
Cancel
Select “Copy Files”
Click “OK”

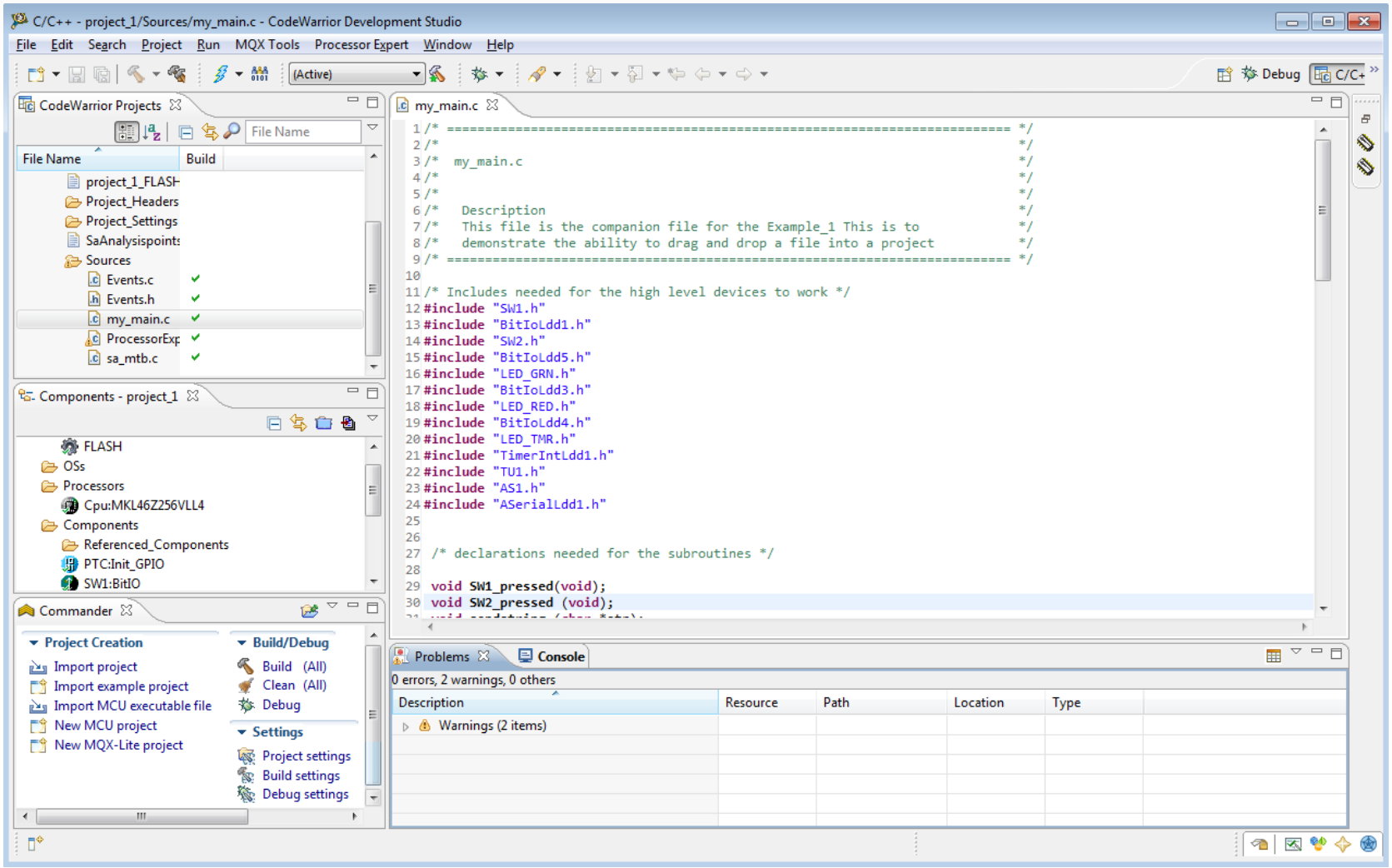
Open “my_main.c”



Minimize Component Inspector



Minimize Component Inspector



Add “Our” Code

- Open “Processor Expert.c to add our code

```
52
53 /* User includes (#include below this line is not maintained by Processor Expert)
54 void my_main(void);
55
56
57 /*lint -save  -e970 Disable MISRA rule (6.3) checking. */
58 int main(void)
59 /*lint -restore Enable MISRA rule (6.3) checking. */
60 {
61   /* Write your local variable definition here */
62
63   /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
64   PE_low_level_init();
65   /*** End of Processor Expert internal initialization.                    ***/
66
67   /* Write your code here */
68   /* For example: for(;;) { } */
69
70   my_main();
71
```

Add declaration here

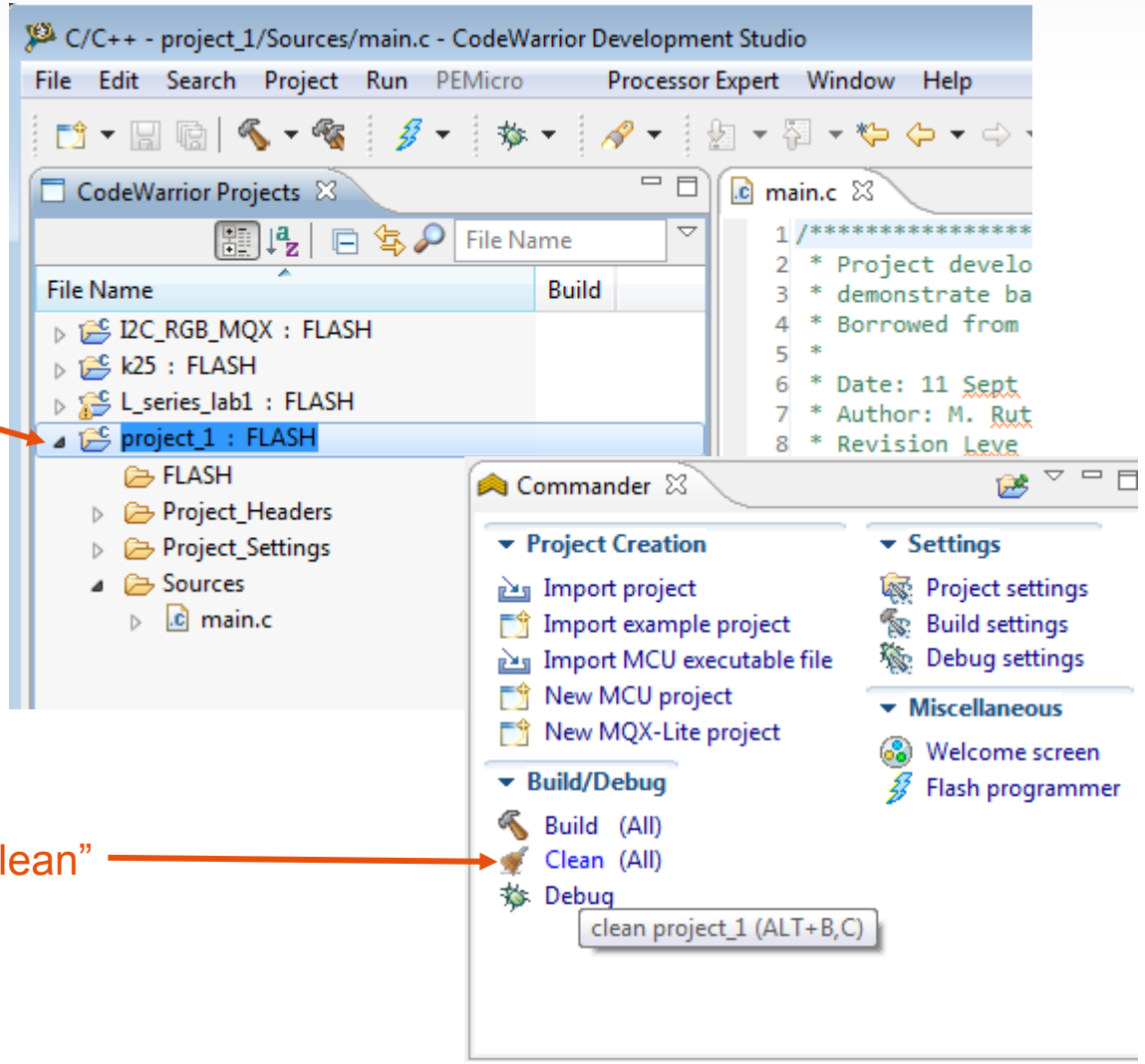
Add call here

Check Point: Create a New Project to Blink an LED

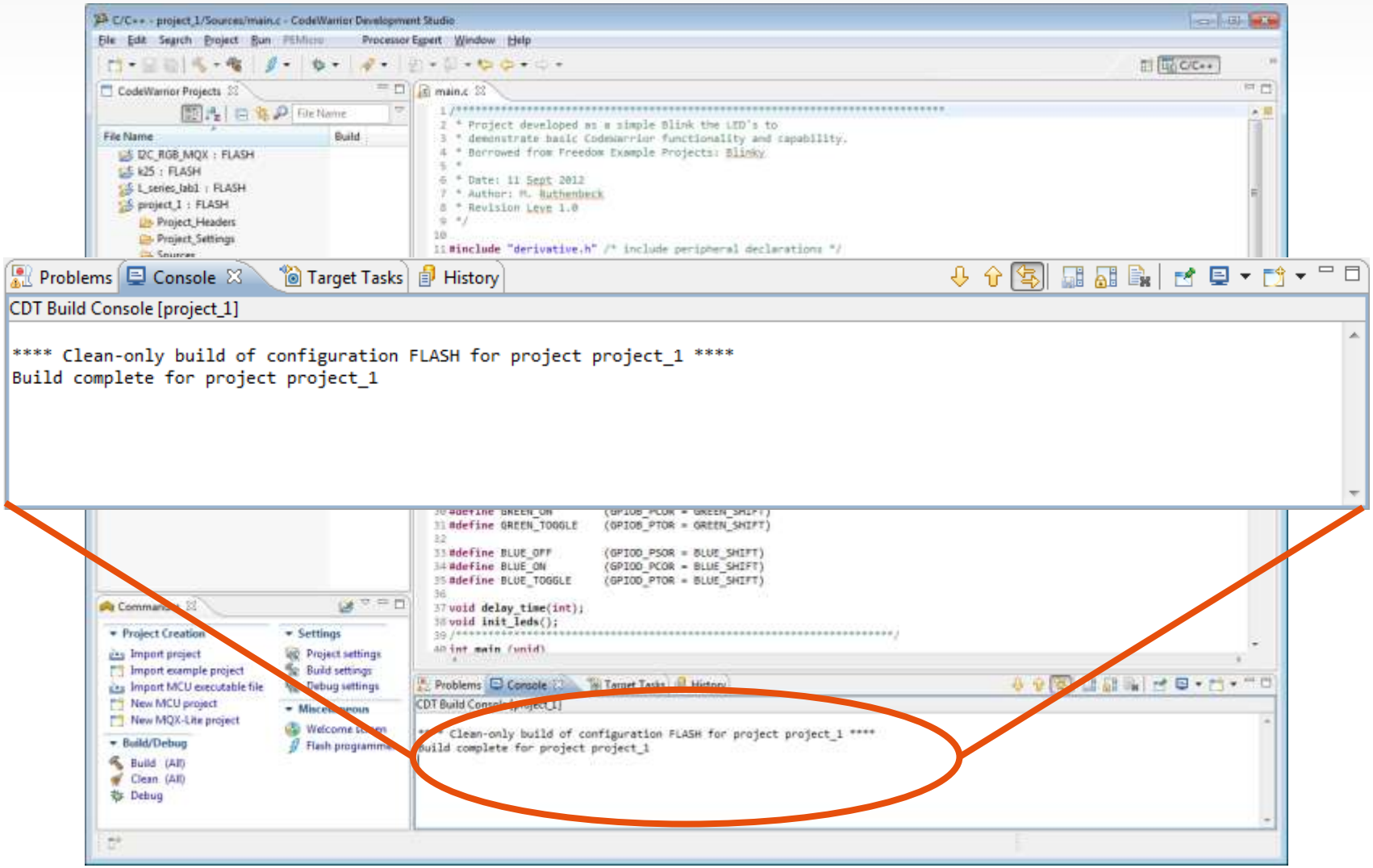
- This hands-on lab shows you how to…
  - Create a new project with the New Project Wizard ✓
  - Select and setup High Level Components ✓
  - Generate Processor Expert Code ✓
  - Import existing files ✓
  - **Build the project**
    - **Select the project**
    - **Clean**
    - **Build**
  - Test the application's functionality

Next up!

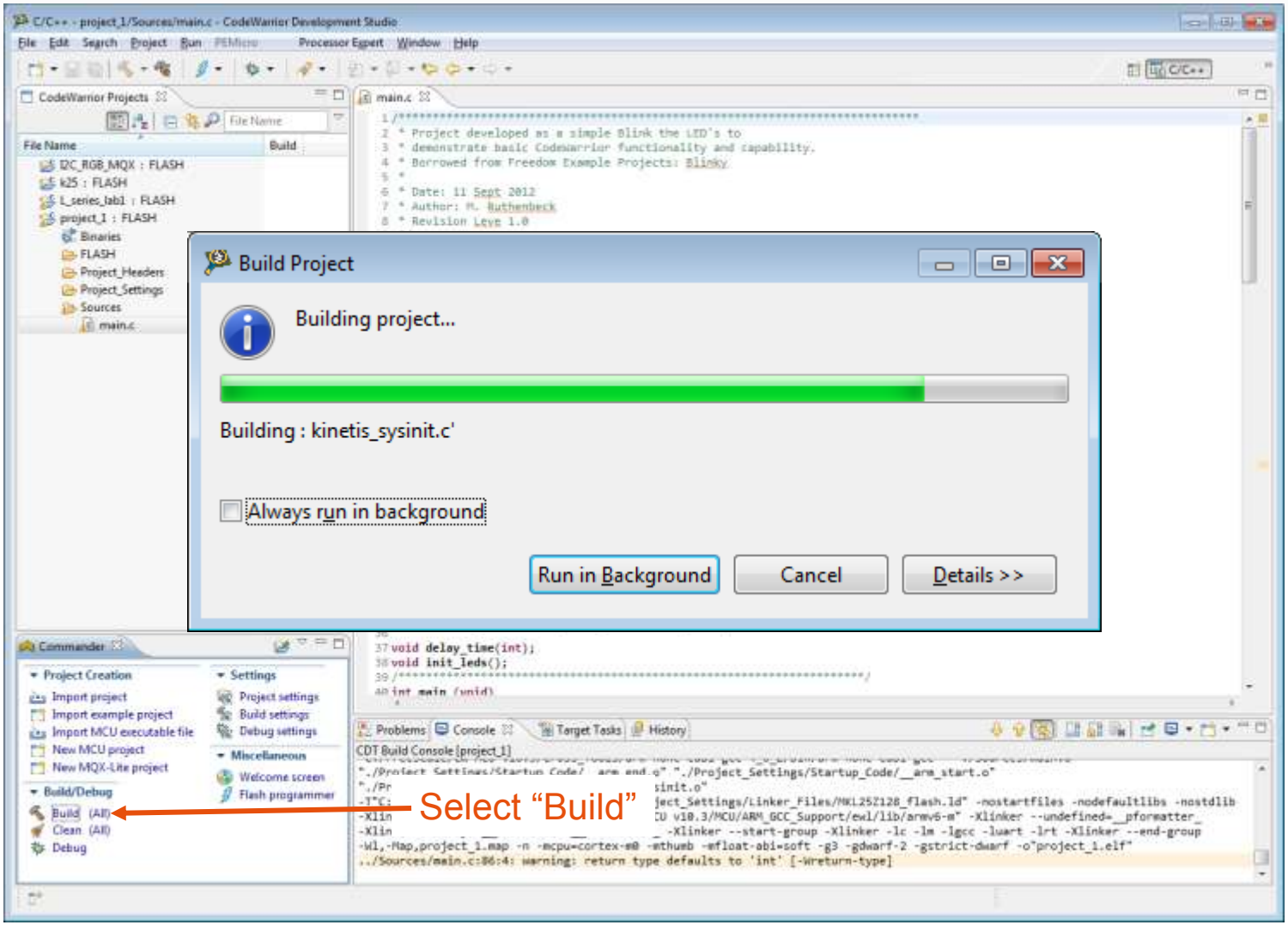
Building the Project: Clean



Building the Project: Clean

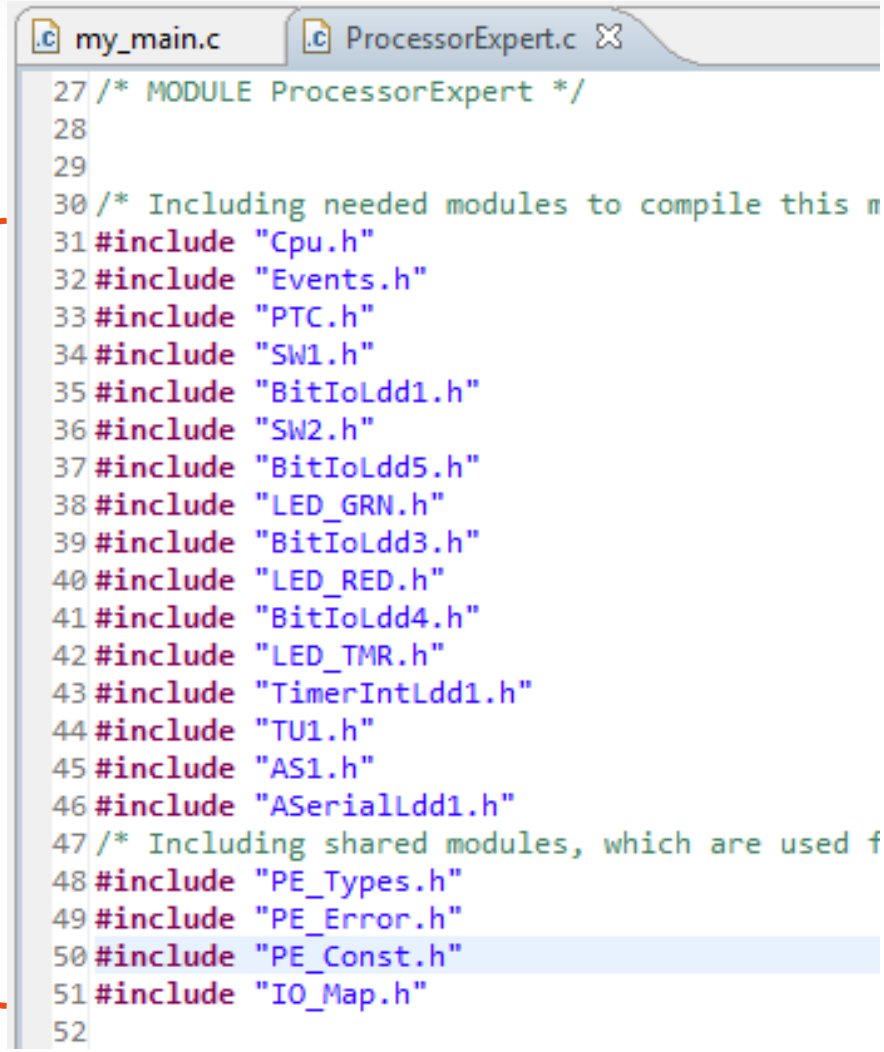


Building the Project



Examine Processor Expert .c

PEx #includes here



Examine Processor Expert .c

Your declarations here

Main.c

Hardware initializations

Your code goes here

```
53 /* User includes (#include below this line is
54 void my_main(void);
55
56
57 /*lint -save  -e970 Disable MISRA rule (6.3)
58 int main(void)
59 /*lint -restore Enable MISRA rule (6.3) check
60 {
61   /* Write your local variable definition her
62
63   /*** Processor Expert internal initializati
64   PE_low_level_init();
65   /*** End of Processor Expert internal initi
66
67   /* Write your code here */
68   /* For example: for(;;) { } */
69
70   my_main();
```

Examine events.c

Timer Interrupt code goes here

```
my_main.c | ProcessorExpert.c | Events.c
22 **         Put your event handler code here.
23 */
24 /*!
25 **  @addtogroup Events_module Events module documentation
26 **  @{
27 */
28 /* MODULE Events */
29
30 #include "Cpu.h"
31 #include "Events.h"
32
33 #ifdef __cplusplus
34 extern "C" {
35 #endif
36
37
38 /* User includes (#include below this line is not maintained by Processor Expert) */
39 #include "LED_GRN.h"
40
41 /*
42 ** ===================================================================
43 **     Event       :  Cpu_OnNMIINT (module Events)
44 **
45 **     Component   :  Cpu [MKL46Z256MC4]
46 */
47 /*!
48 **     @brief
49 **         This event is called when the Non maskable interrupt had
50 **         occurred. This event is automatically enabled when the [NMI
51 **         interrupt] property is set to 'Enabled'.
52 */
53 /* ===================================================================*/
54 void Cpu_OnNMIINT(void)
55 {
56   /* Write your code here ... */
57 }
58
59 /*
60 ** ===================================================================
61 **     Event       :  LED_TMR_OnInterrupt (module Events)
62 **
63 **     Component   :  LED_TMR [TimerInt]
64 **     Description :
65 **         When a timer interrupt occurs this event is called (only
66 **         when the component is enabled - <Enable> and the events are
67 **         enabled - <EnableEvent>). This event is enabled only if a
68 **         <Interrupt service/event> is enabled.
69 **     Parameters  : None
70 **     Returns     : Nothing
71 ** ===================================================================
72 */
73 void LED_TMR_OnInterrupt(void)
74 {
75     /* Write your code here ... */
76     LED_GRN_NegVal();
77
78 }
79
80 /* END Events */
81
82 #ifdef __cplusplus
83 }  /* extern "C" */
84 #endif
85
86 /*!
87 ** @}
88 */
89 /*
90 ** ###################################################################
91 **
```

Examine My_main.c

```
/* Includes needed for the high level devices to work, copied from ProcessorExpert.c */
#include "SW1.h"
#include "BitIoLdd1.h"
#include "SW2.h"
#include "BitIoLdd5.h"
#include "LED_GRN.h"
#include "BitIoLdd3.h"
#include "LED_RED.h"
#include "BitIoLdd4.h"
#include "LED_TMR.h"
#include "TimerIntLdd1.h"
#include "TU1.h"
#include "AS1.h"
#include "ASerialLdd1.h"


/* declarations needed for the subroutines */

void SW1_pressed(void);
void SW2_pressed (void);
void sendstring (char *str);
void delay(void);
```

#includes copied from ProcessorExpert.c

Declarations needed for our code

Examine My_main.c

```
void my_main()
{

  sendstring("Welcome to Processor Expert and CodeWarrior! \r\n\n");

 while (1) {
   if ((SW1_GetVal())==0)  /* Check to see if SW1 has been pressed */
   SW1_pressed();

   else
     LED_RED_PutVal(1);


   if ((SW2_GetVal()) ==0) /* If SW2 has been pressed */
     SW2_pressed();

   else
     LED_RED_PutVal(1); /* make sure led is OFF  */

   delay();  /* for demo purposes only, not recommended in real life :) */
 }
}

/* what to do when SW1 is pressed!  ***************************/
void SW1_pressed(){
 LED_RED_PutVal(0);
 sendstring ("\r\nSW1 has been pressed!\r\n"); /* announce to the world switch has been pressed */

}

/* what to do when SW2 is pressed!  ***************************/
void SW2_pressed(){
 LED_RED_PutVal(0);

}


void sendstring (char *str){
 int x=0;
 while (str[x] != '\0'){
   while (AS1_SendChar(str[x])==ERR_TXFULL); /* wait for buffer to empty before moving on */
   x++;
 }
}



/******************** function for short delay ******************/
/** Not recommended for real life coding - for demonstration only */

void delay(){

 unsigned int i, n;
 for(i=0;i<3000;i++)
 {
   for(n=0;n<20;n++)
   {
     asm("nop");
   }
 }
}
```

Main loop – watch for switch presses

Switch press routine

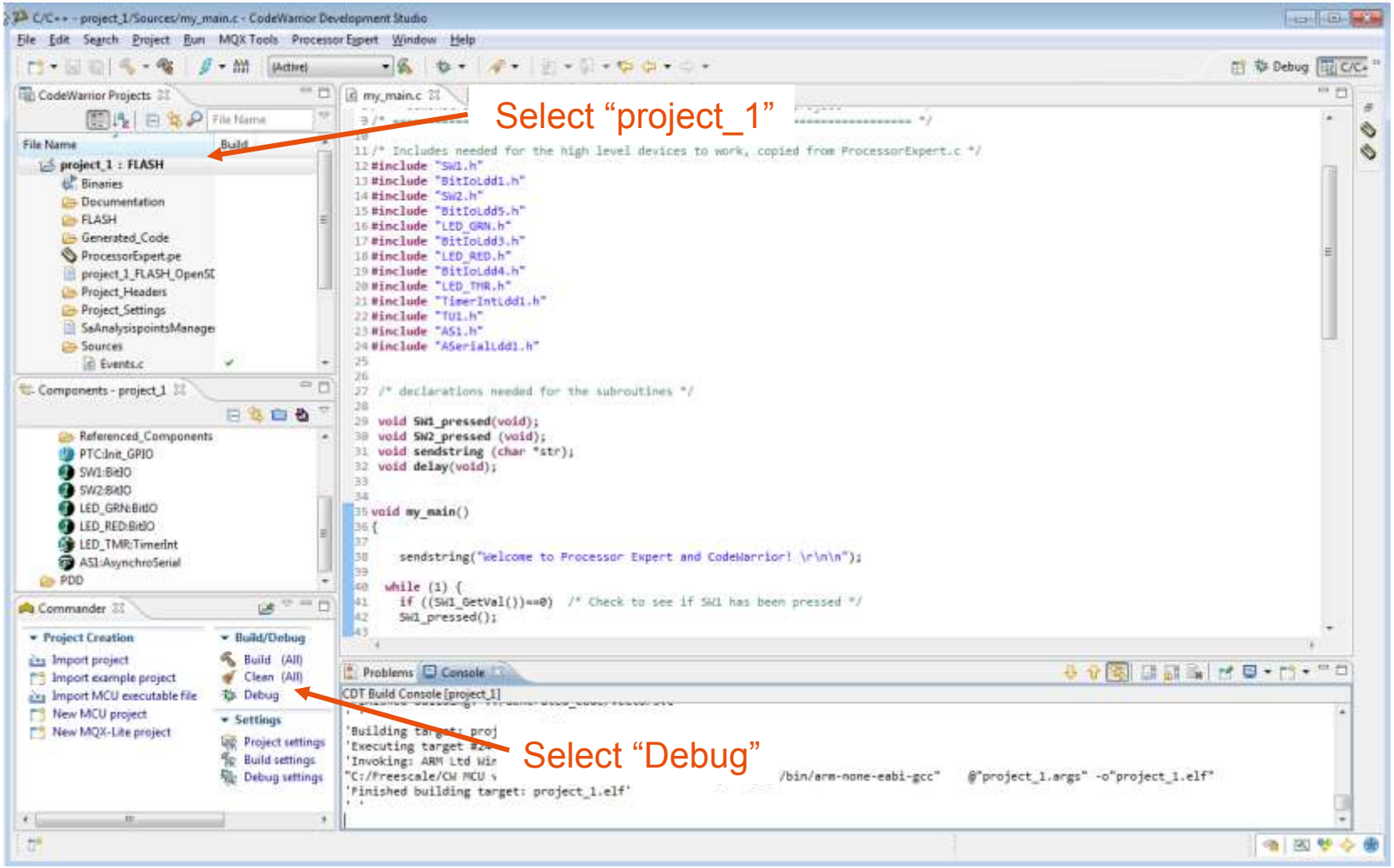
Delay loop

Check Point: Create a New Project to Blink an LED

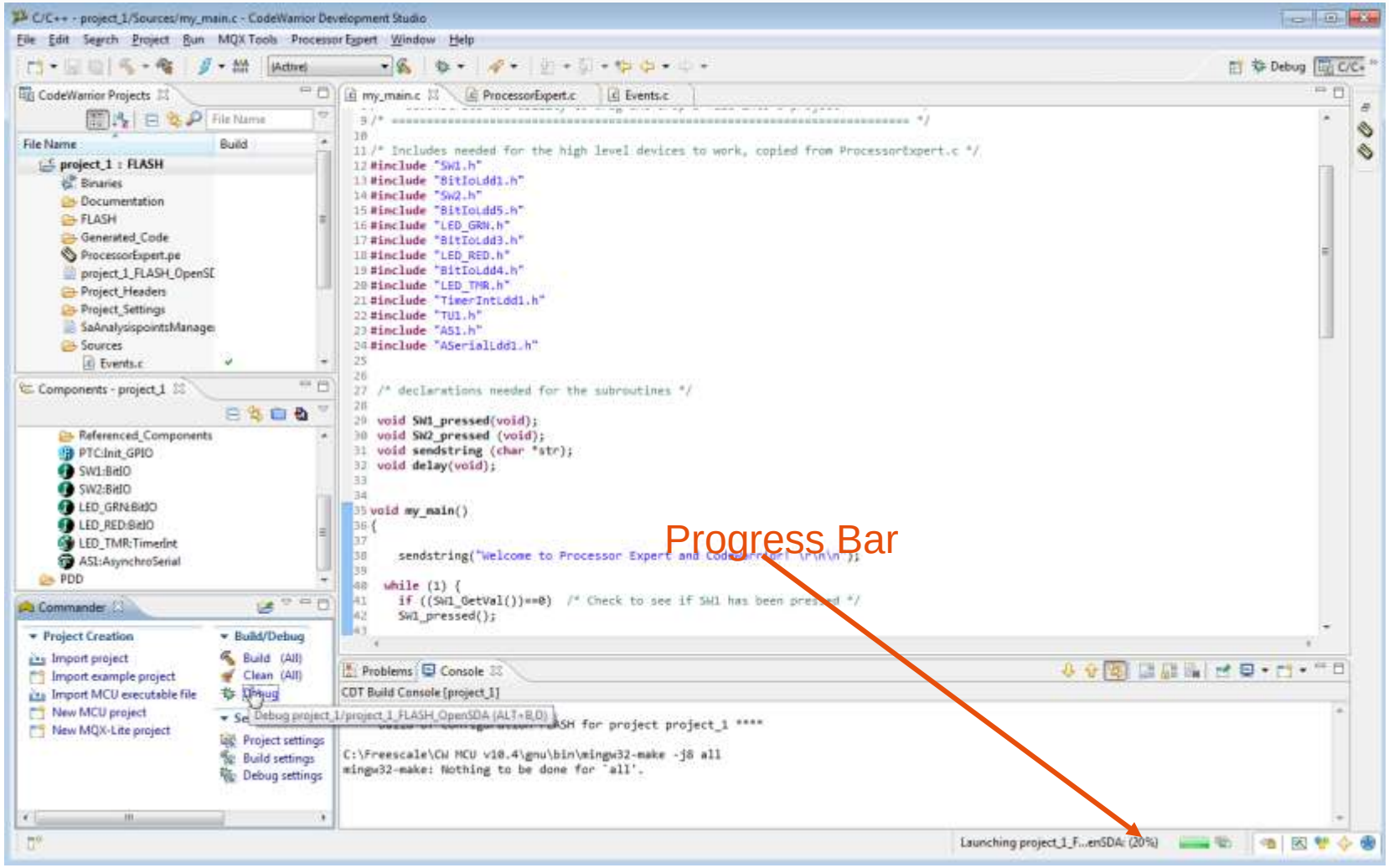
- This hands-on lab shows you how to…
  - Create a new project with the New Project Wizard ✓
  - Select and setup High Level Components ✓
  - Generate Processor Expert Code ✓
  - Import existing files ✓
  - Build the project ✓
  - **Test the application's functionality**
    - **Download and debug**
    - **Inspect the registers**
    - **Setting breakpoints**

Next up!

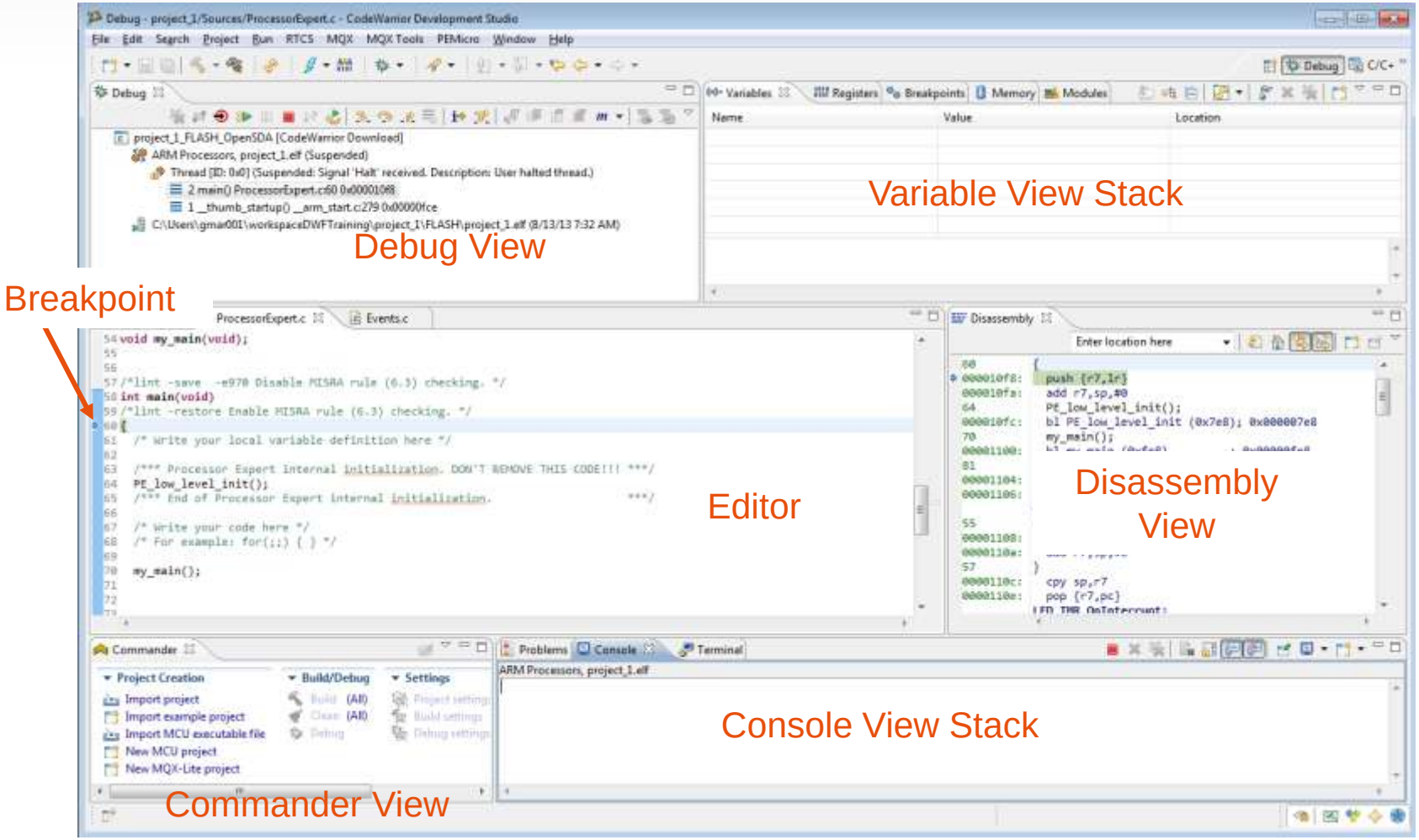
Application Test: Download/Debug



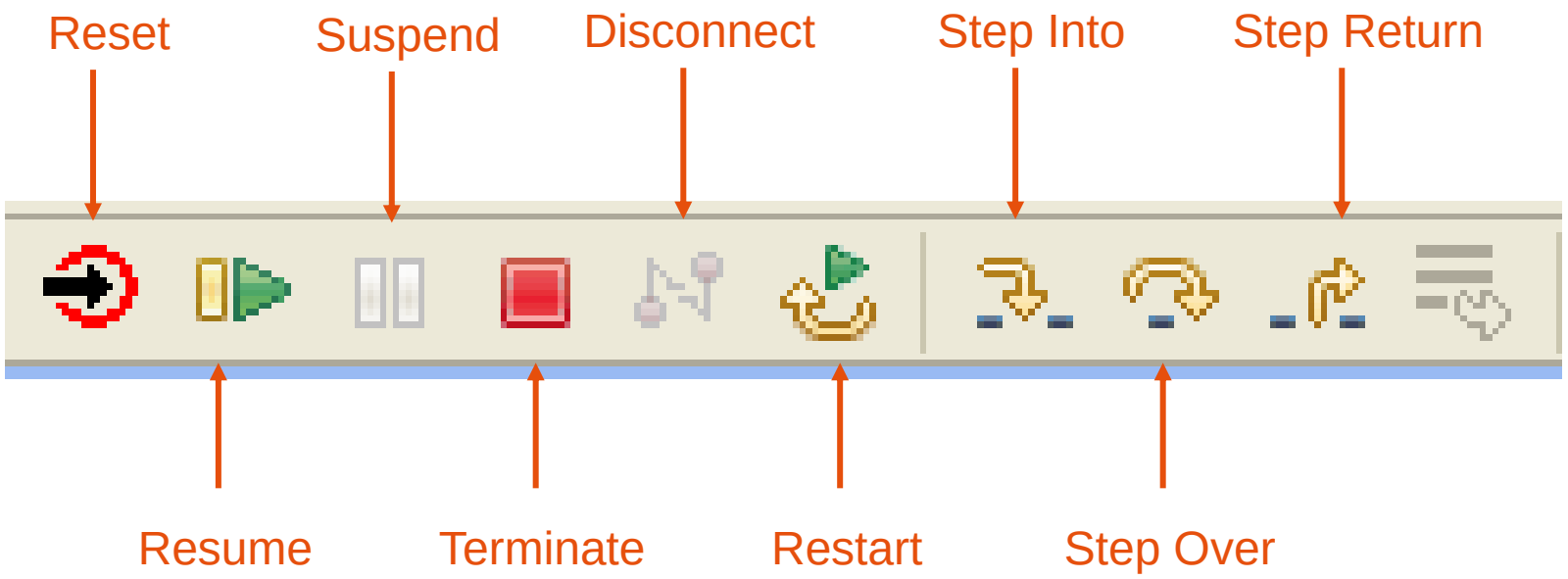
Application Test: Download/Debug



Application Test: Debug Perspective



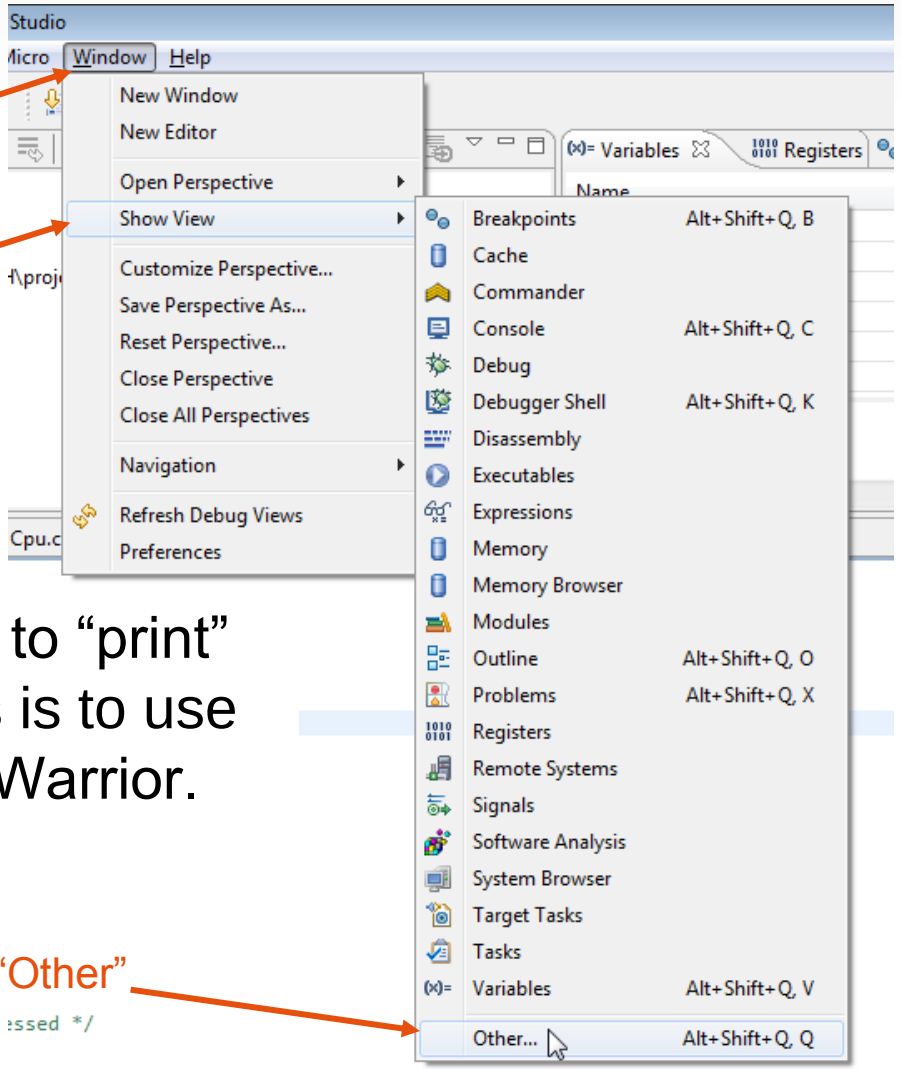
Application Test: Debug Perspective



Application Test: Setting up Terminal

Select “Window”

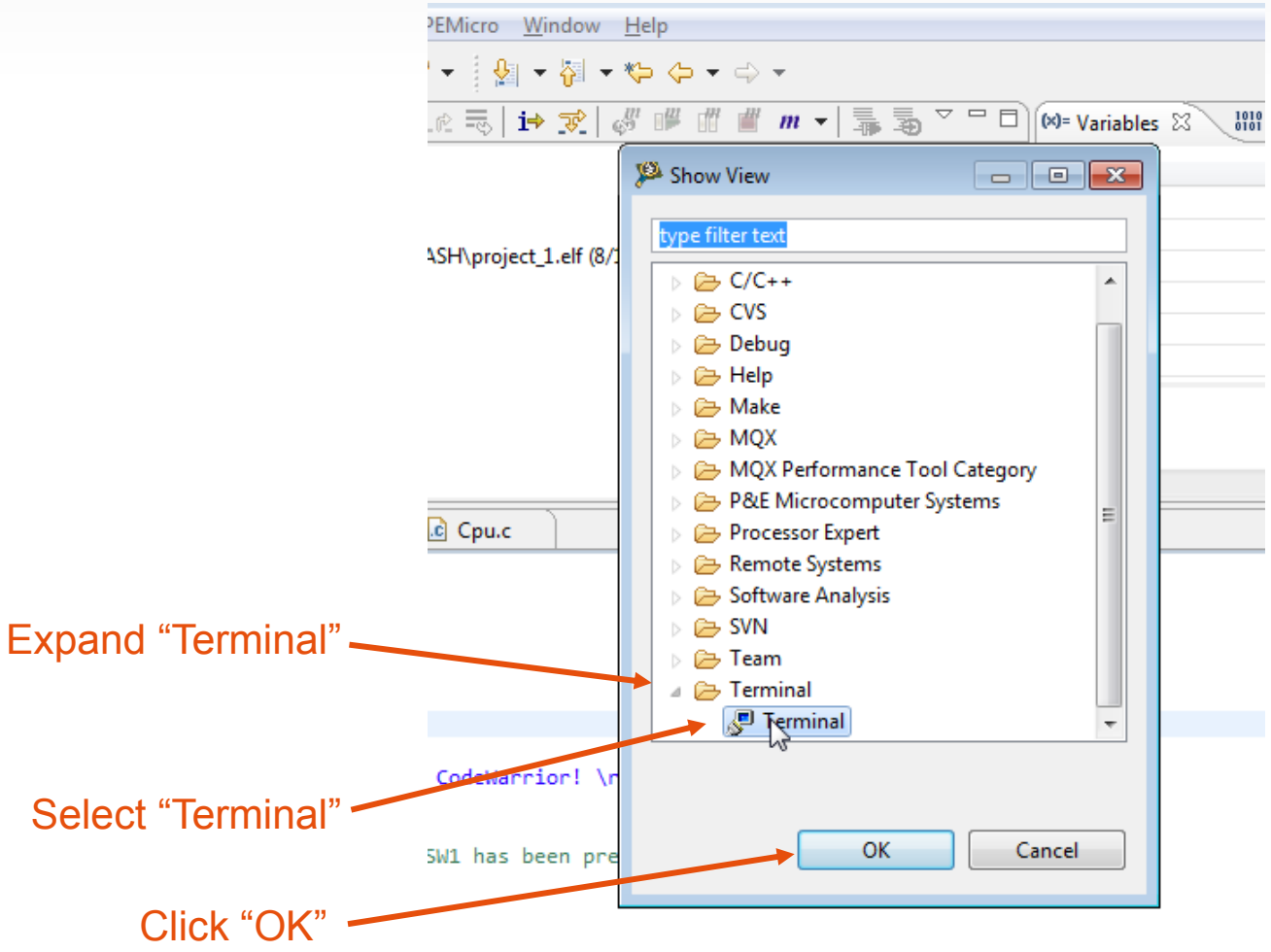
Select “Show Views”



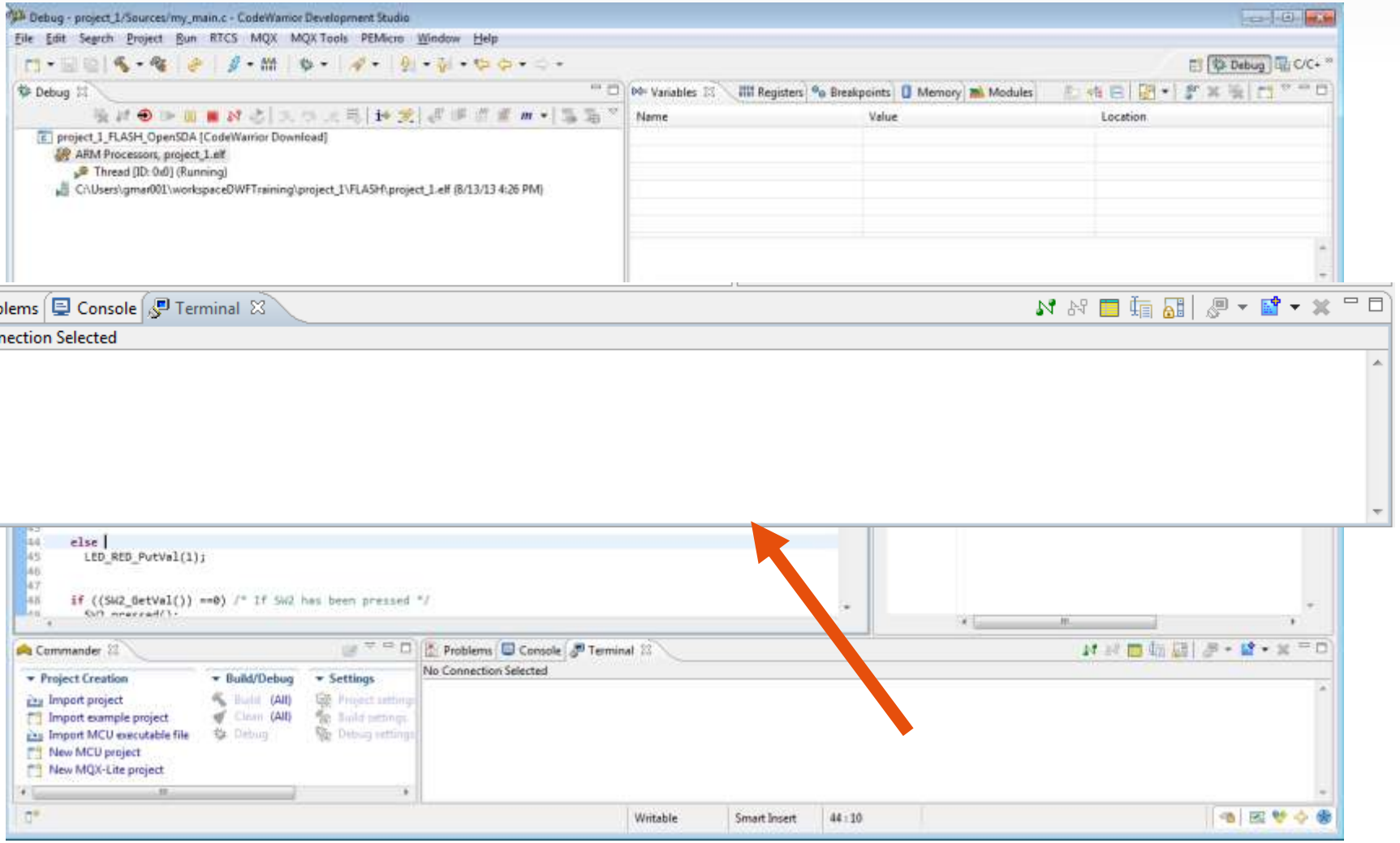
The application uses the uart to “print” data. An easy way to see this is to use the built in “terminal” in CodeWarrior.

Select “Other”

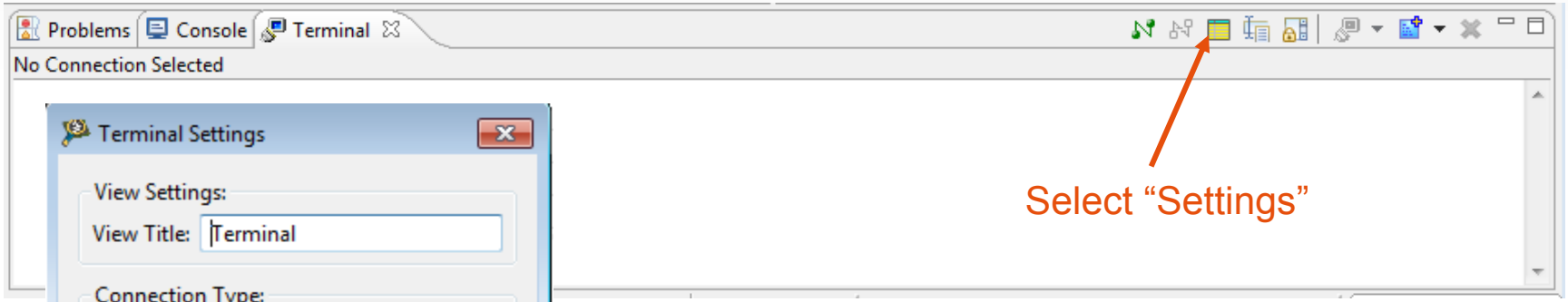
Application Test: Setting up Terminal



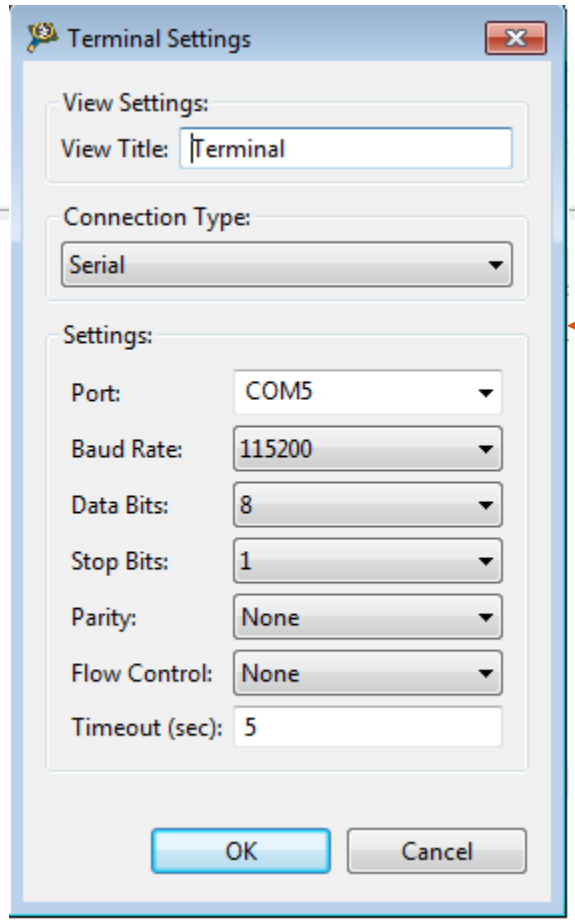
Application Test: Setting up Terminal



Application Test: Setting up Terminal



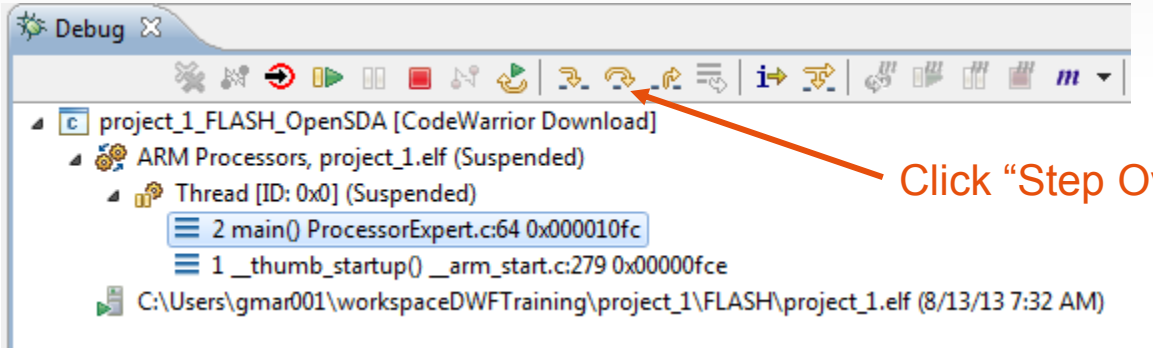
Select “Settings”

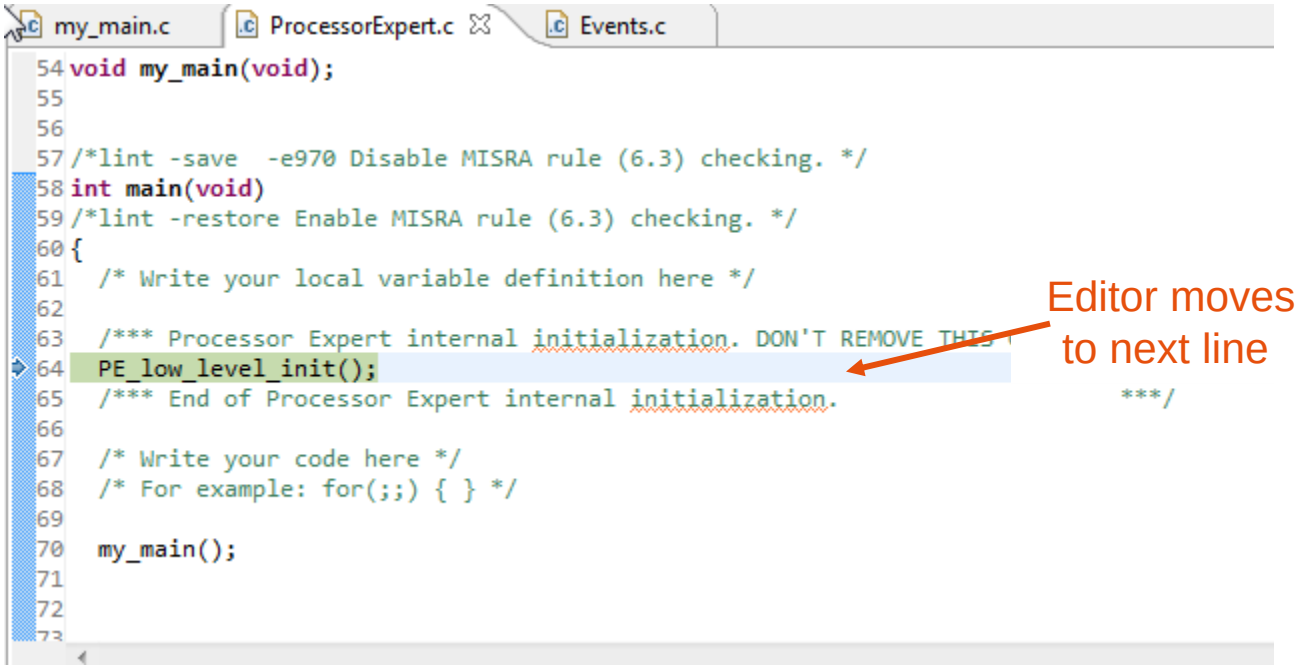


Setup the parameters

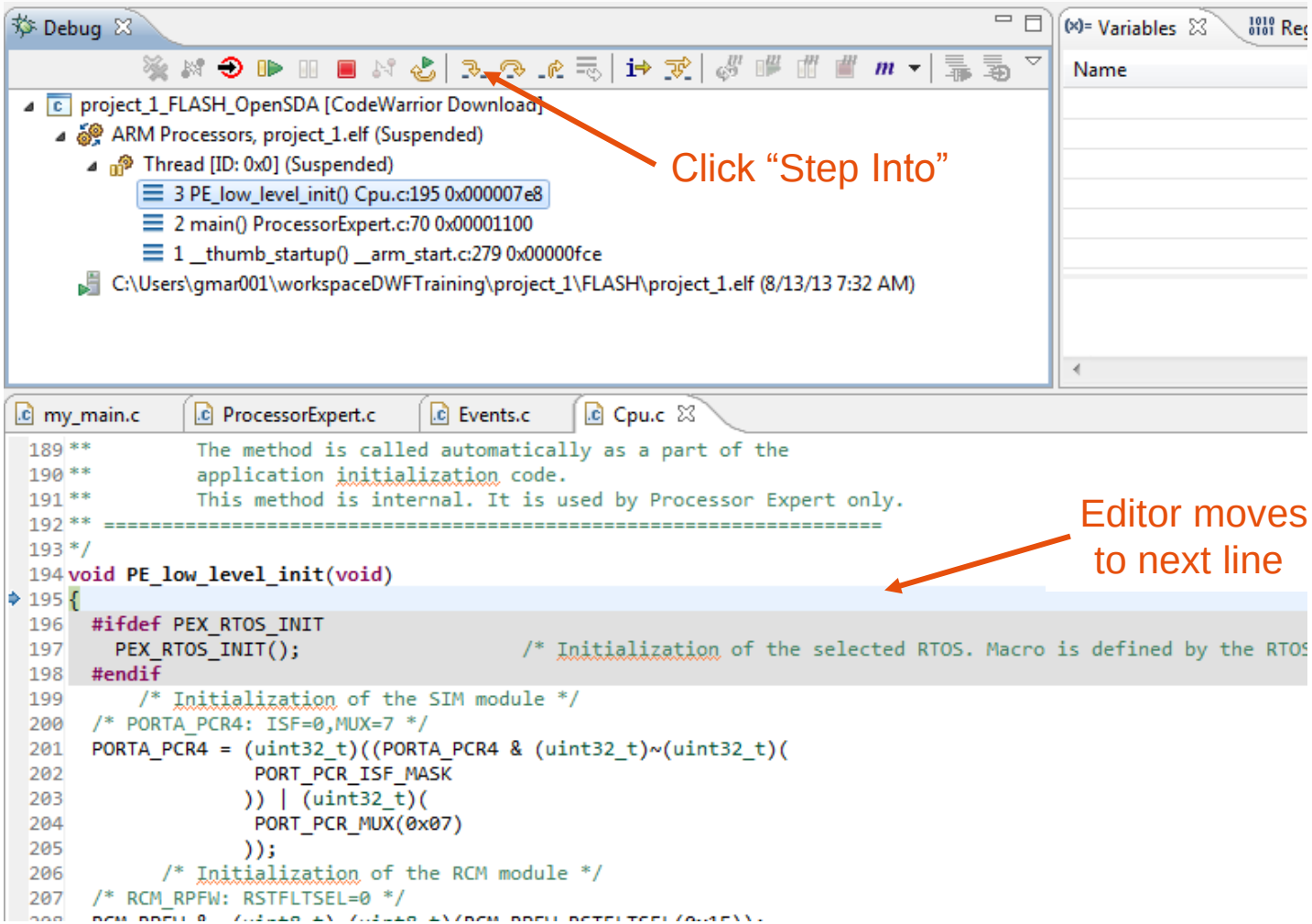
- Connection Type: Serial
- Port – See your Device Manager
- Baud Rate: 115200
- 8,1, none, none

Application Test: Single Step

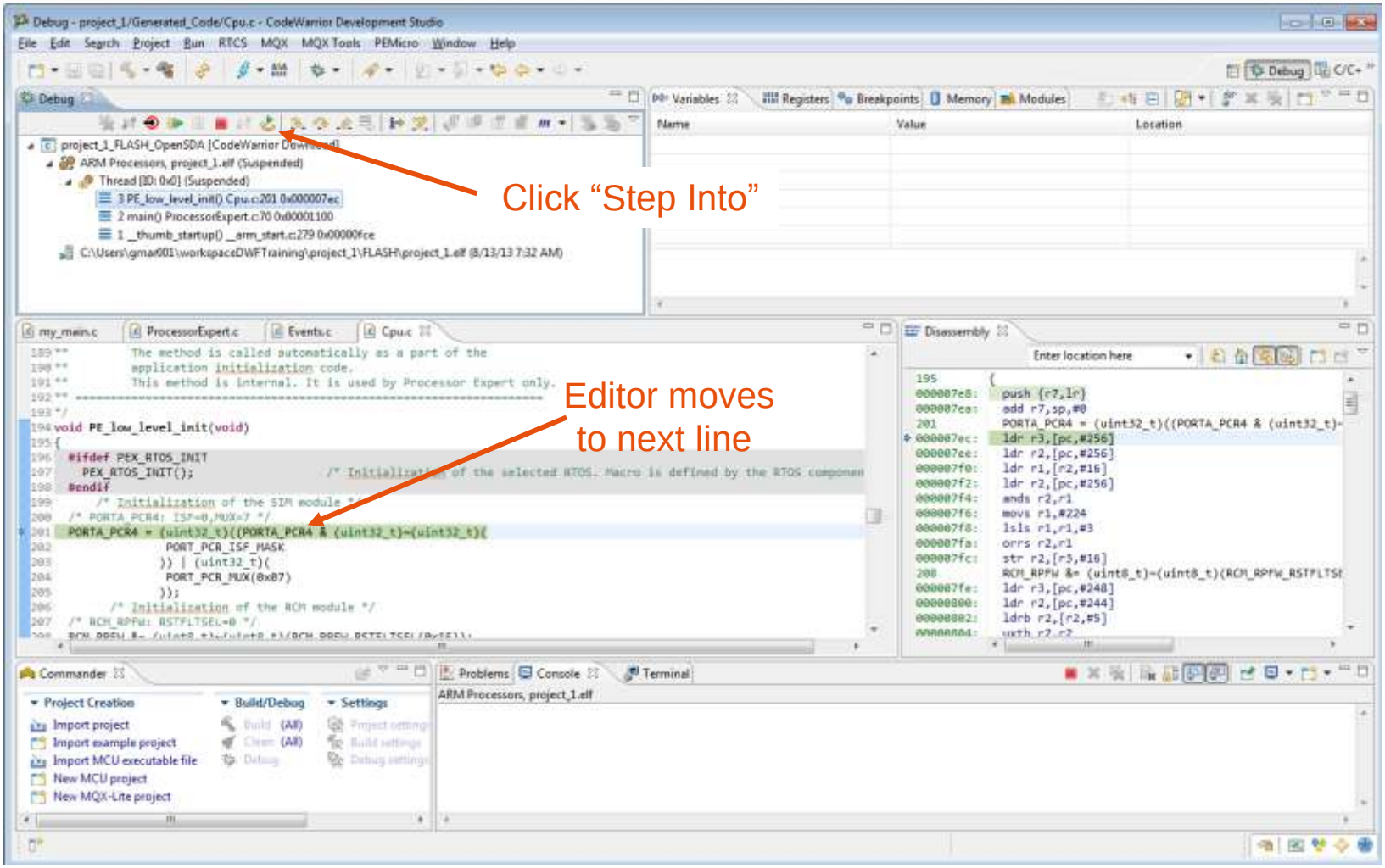




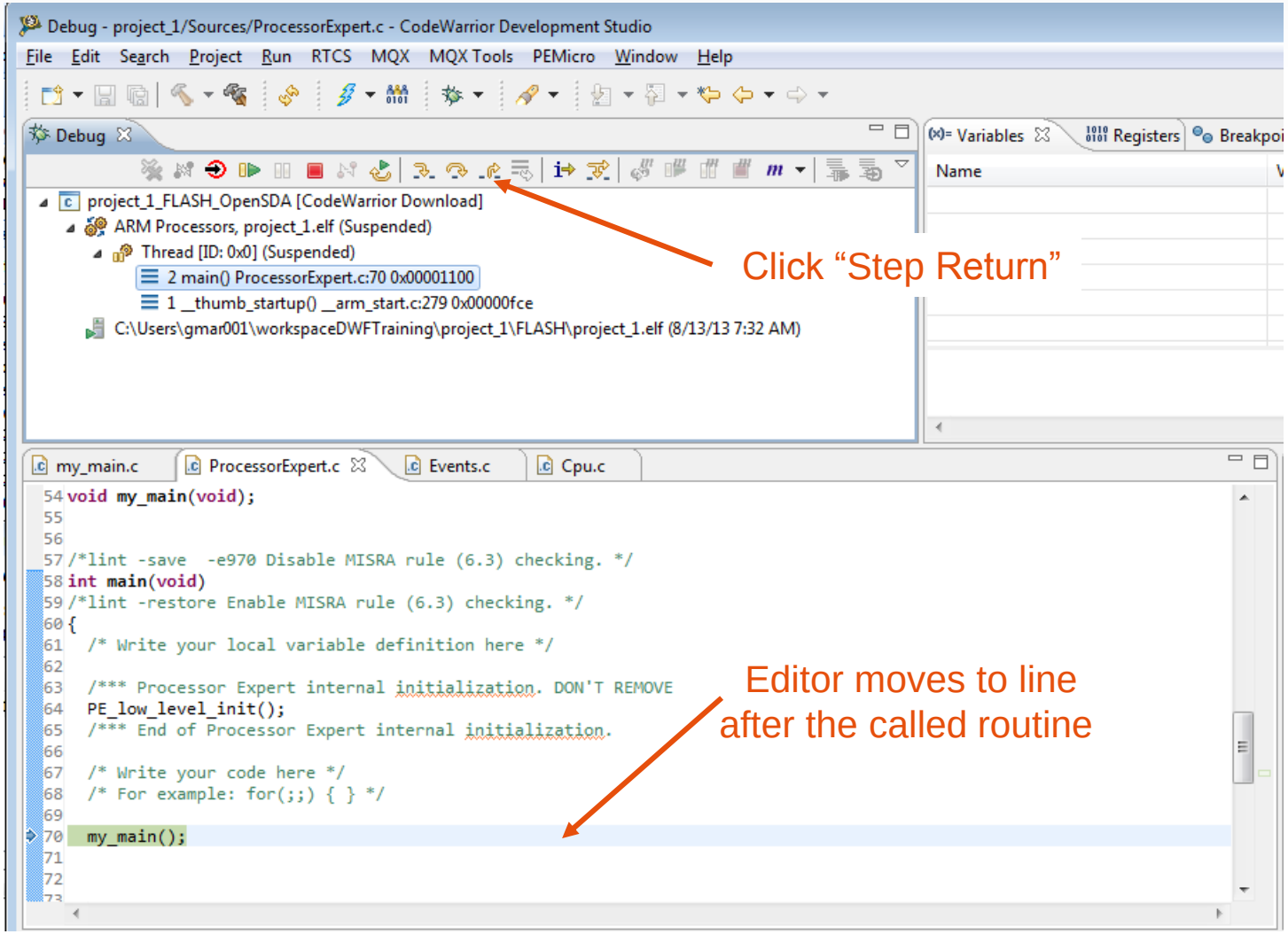
Application Test: Single Step/Step Into



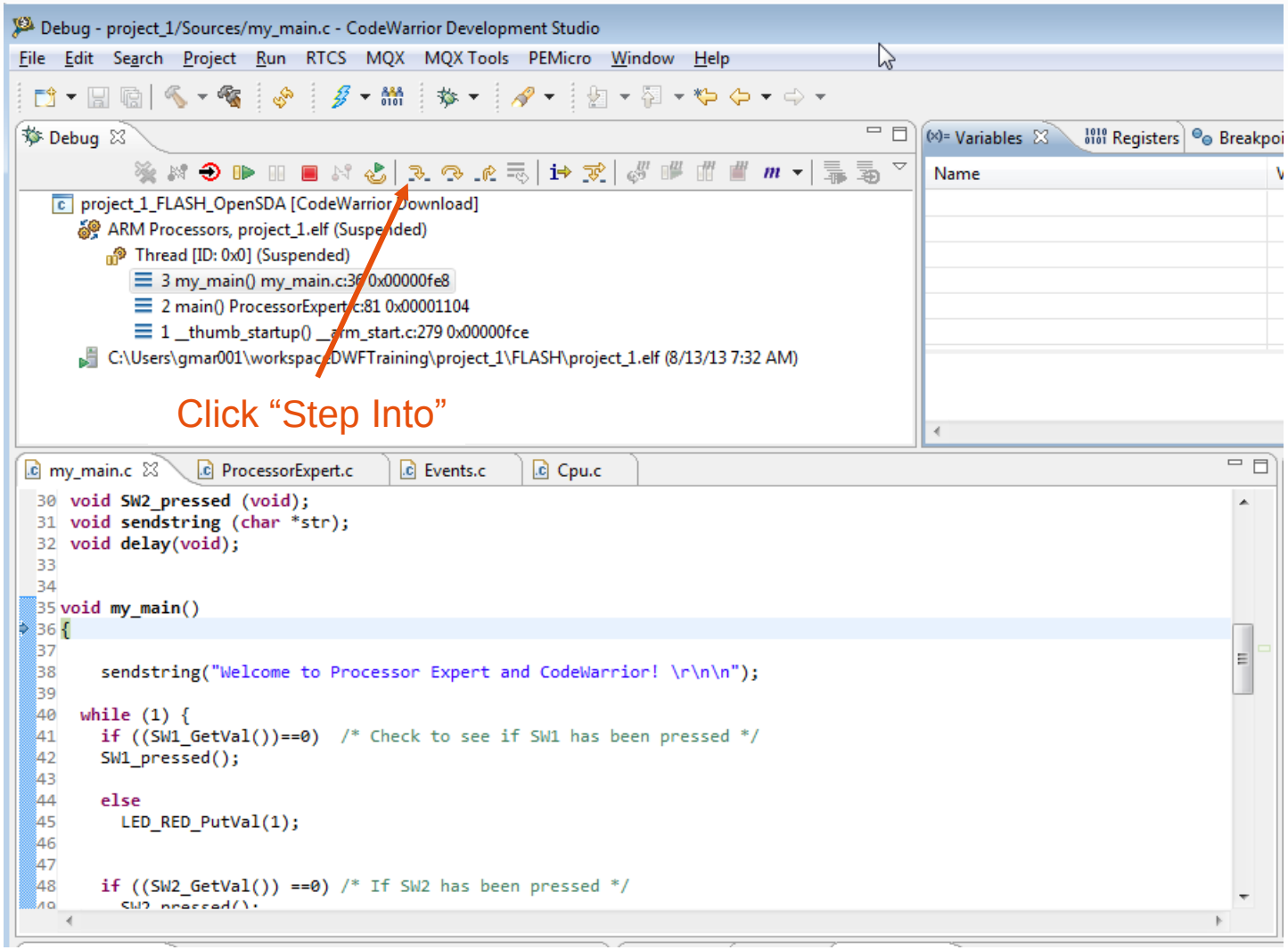
Application Test: Single Step



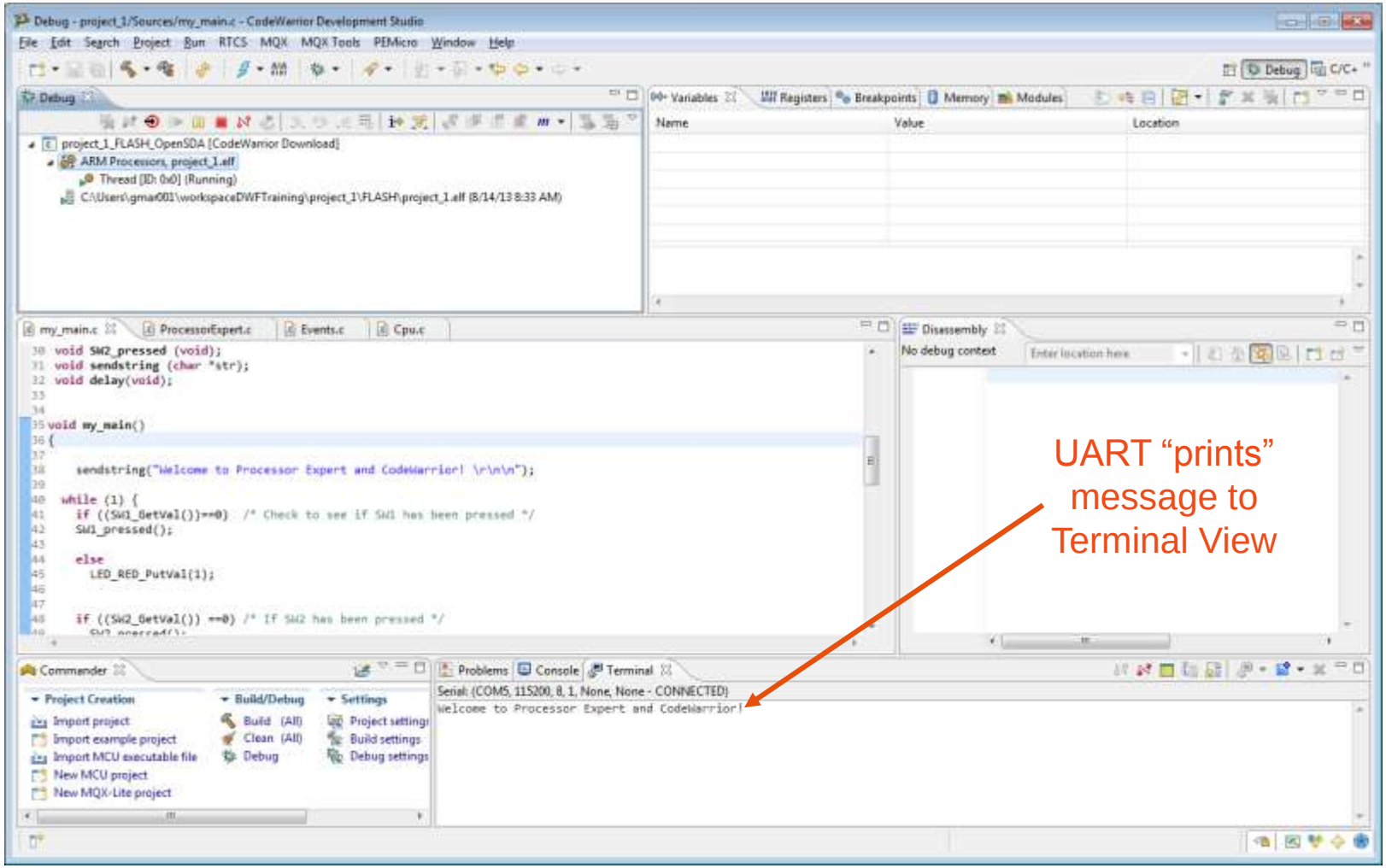
Application Test: Single Step – Step Return



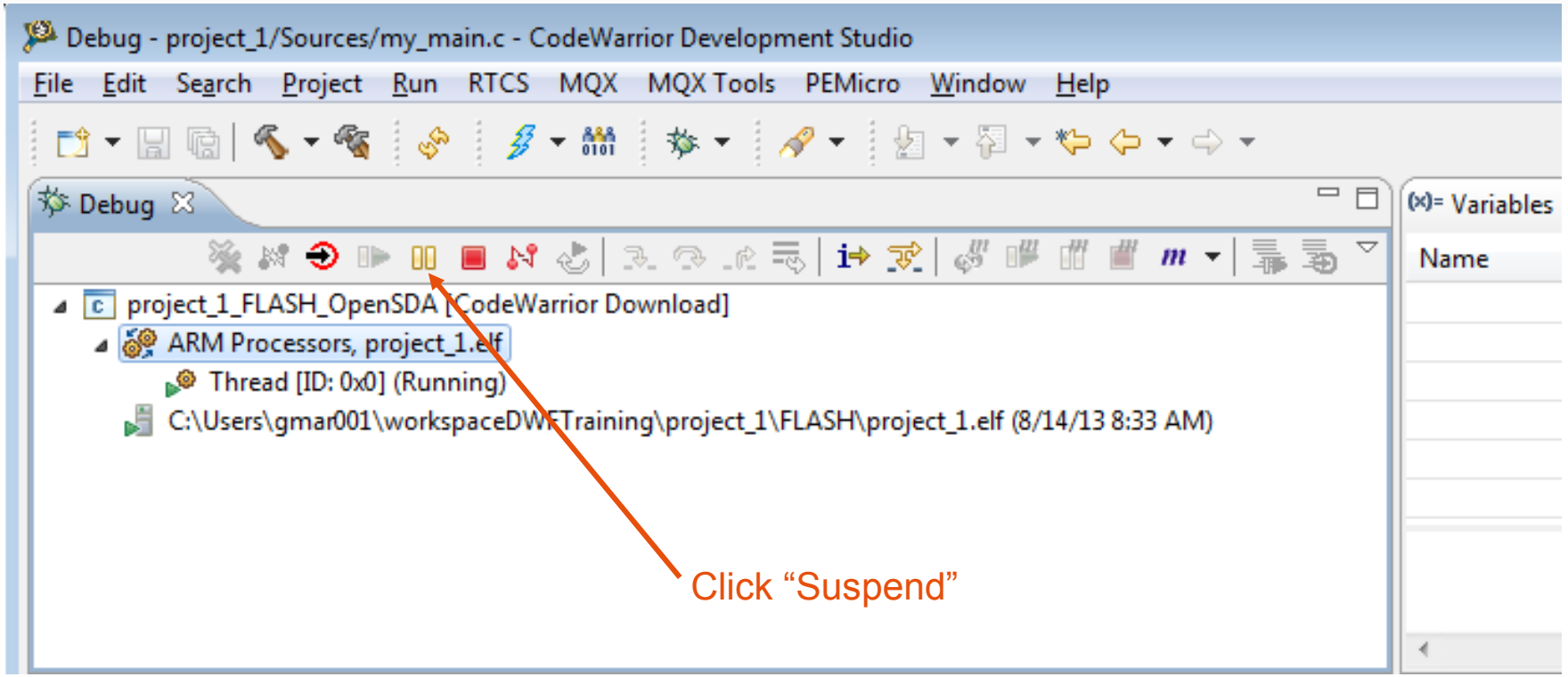
Application Test: Single Step – Step Return



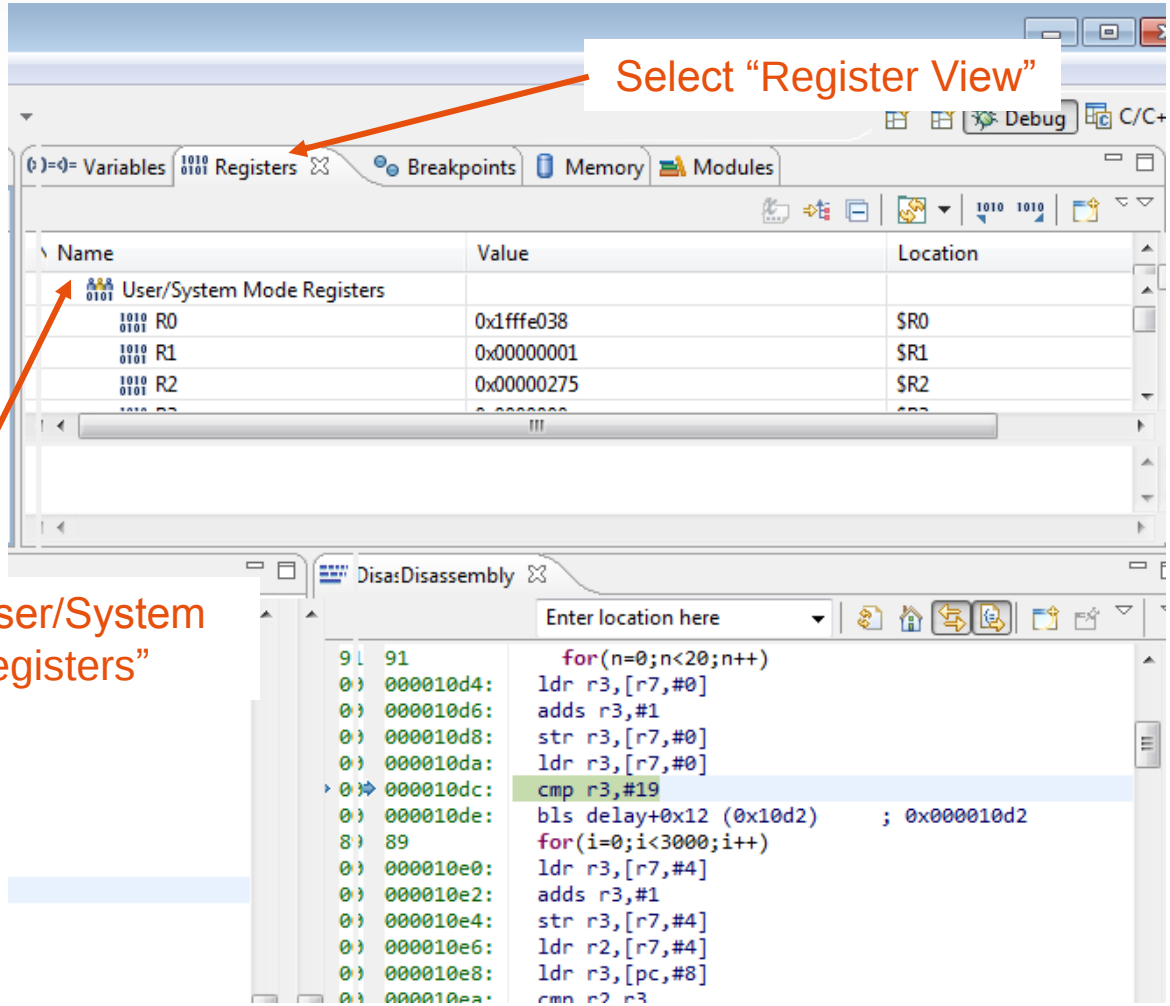
Application Test: Run



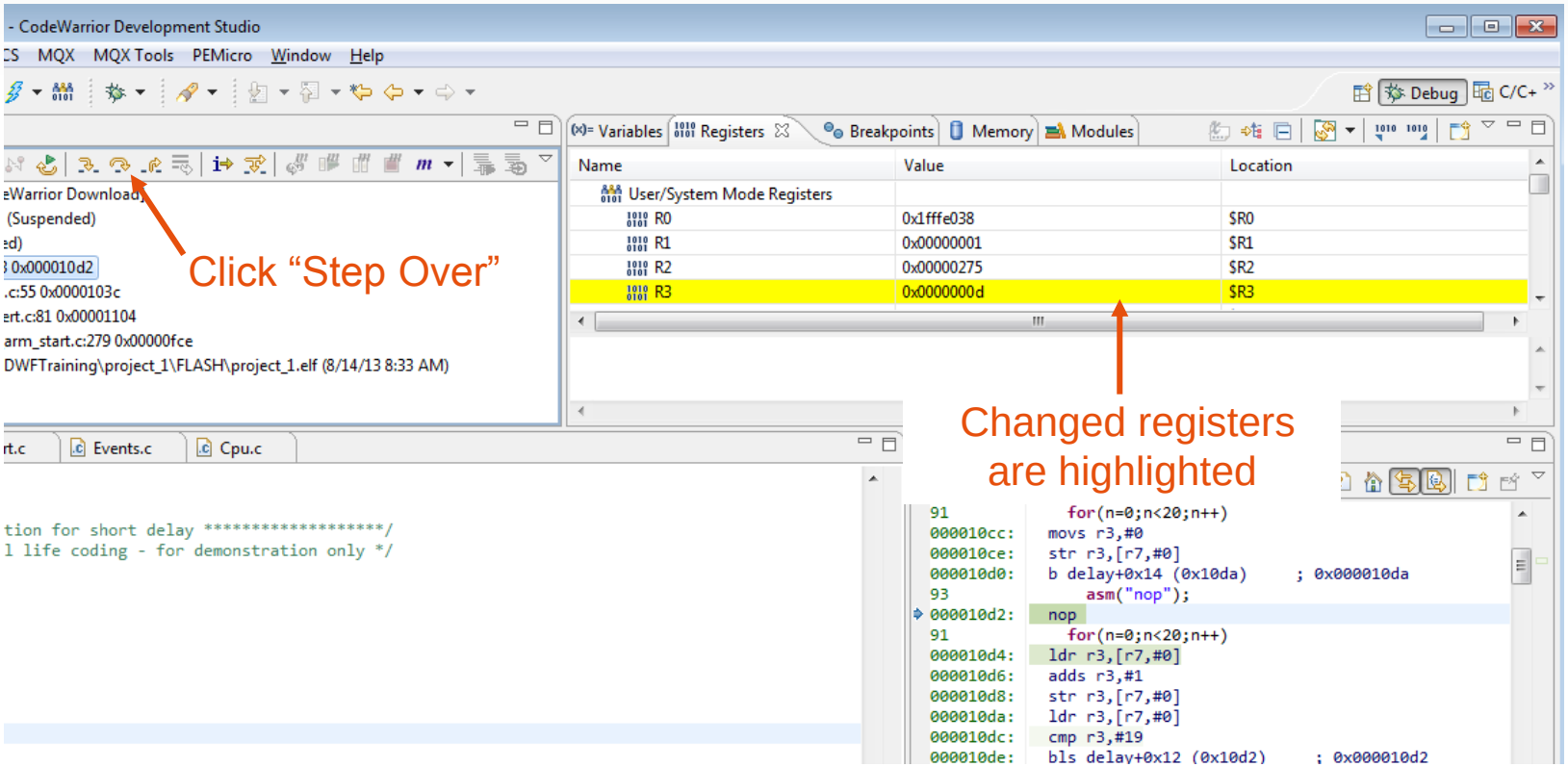
Application Test: Inspect Registers



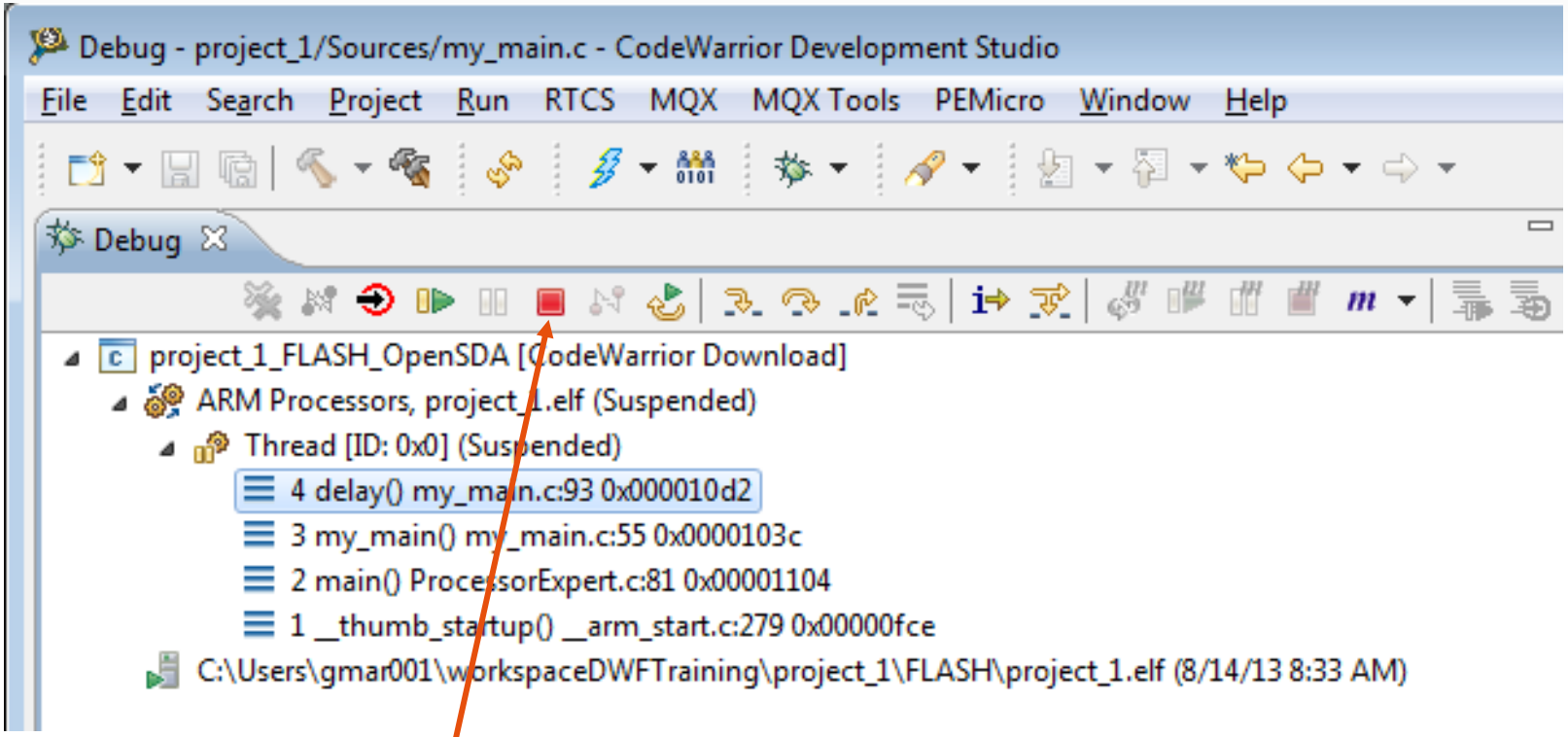
Application Test: Inspect Registers



Application Test: Inspect Registers



Application Test: Inspect Registers



Click “Terminate” to stop debug session

Application Test: Setting Breakpoints

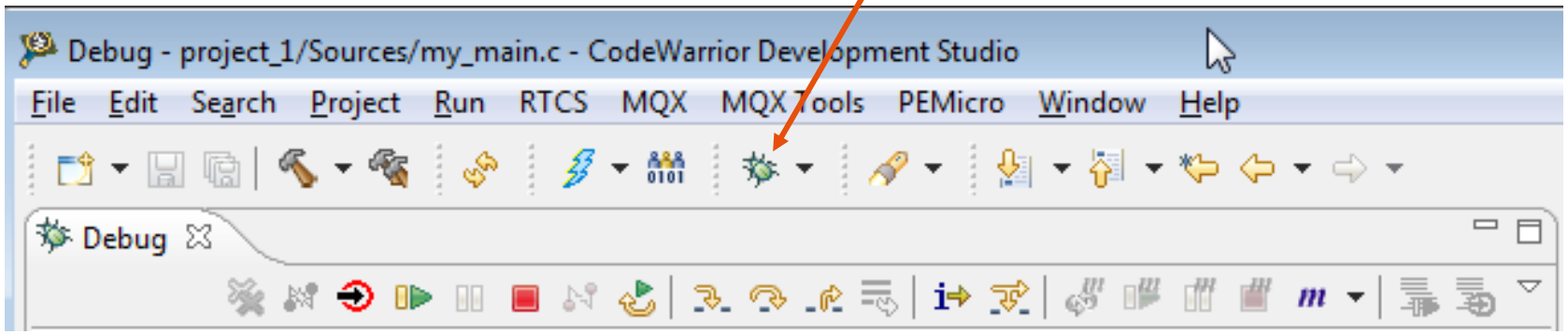
Double click
at line 38
and line 41
to set breakpoints

my_main.c | ProcessorExpert.c | Events.c | Cpu.c

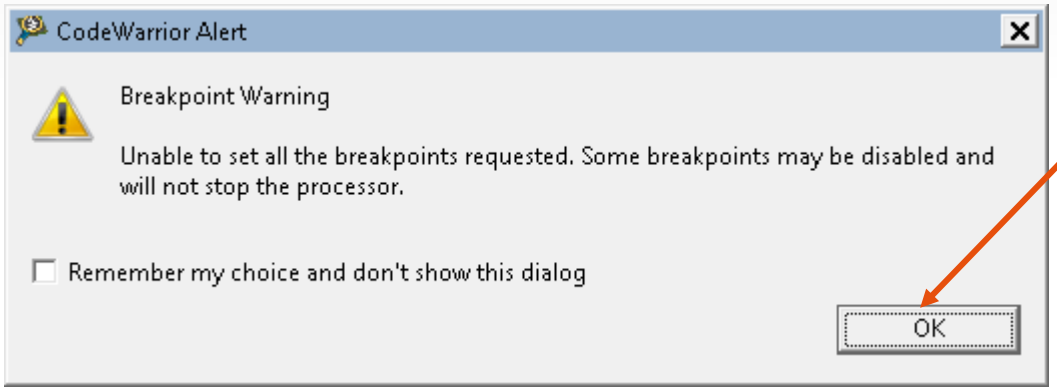
```
34
35 void my_main()
36 {
37
38     sendstring("Welcome to Processor Expert and CodeWarrior! \r\n\n");
39
40   while (1) {
41     if ((SW1_GetVal())==0)  /* Check to see if SW1 has been pressed */
42     SW1_pressed();
43
44     else
45       LED_RED_PutVal(1);
46
47
48     if ((SW2_GetVal()) ==0) /* If SW2 has been pressed */
49       SW2_pressed();
50
51     else
52       LED_RED_PutVal(1); /* make sure led is OFF  */
53
54     delay();  /* for demo purposes only, not recommended in real life :) */
55   }
56 }
57
```

Application Test: Setting Breakpoints

Click “Debug” to restart the debug session



Application Test: Setting Breakpoints



- This error occurs because the number of break points set exceeds the number of breakpoints available on the chip.
- The KL46 only has 2 breakpoints available.
- There are 3 breakpoints set
  - At main
  - At line 38
  - At line 41

Application Test: Setting Breakpoints

- Code stopped at first user breakpoint and not at the system set breakpoint at “main”.

my_main.c | ProcessorExpert.c | Events.c | Cpu.c

```
34
35 void my_main()
36 {
37
38    sendstring("Welcome to Processor Expert and CodeWarrior! \r\n\n");
39
40  while (1) {
41    if ((SW1_GetVal())==0)  /* Check to see if SW1 has been pressed */
42    SW1_pressed();
43
44    else
45      LED_RED_PutVal(1);
46
47
48    if ((SW2_GetVal()) ==0) /* If SW2 has been pressed */
49      SW2_pressed();
```

- Click “Resume”

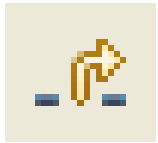
Application Test: Setting Breakpoints

- Click “Step Return”

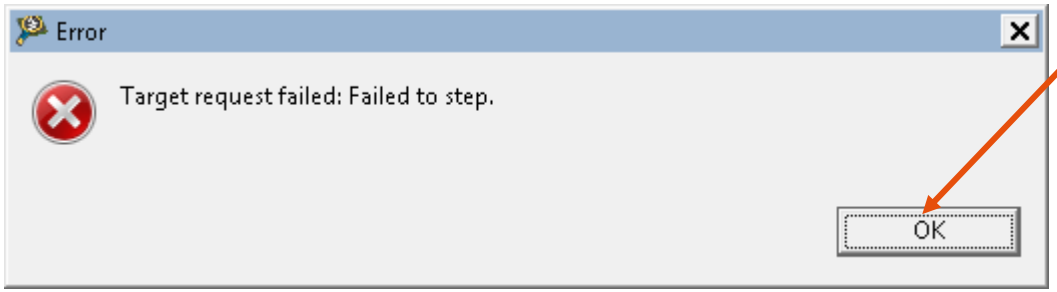
my_main.c | ProcessorExpert.c | Events.c | Cpu.c

```
34
35 void my_main()
36 {
37
38    sendstring("Welcome to Processor Expert and CodeWarrior! \r\n\n");
39
40  while (1) {
41    if ((SW1_GetVal())==0)  /* Check to see if SW1 has been pressed */
42    SW1_pressed();
43
44    else
45      LED_RED_PutVal(1);
46
47
48    if ((SW2_GetVal()) ==0) /* If SW2 has been pressed */
49      SW2_pressed();
50
51    else
52      LED_RED_PutVal(1); /* make sure led is OFF  */
53
54    delay();  /* for demo purposes only, not recommended in real life :) */
55  }
```

Stopped at the next Break Point



Application Test: Setting Breakpoints



- This error message is displayed since there are too many break points.

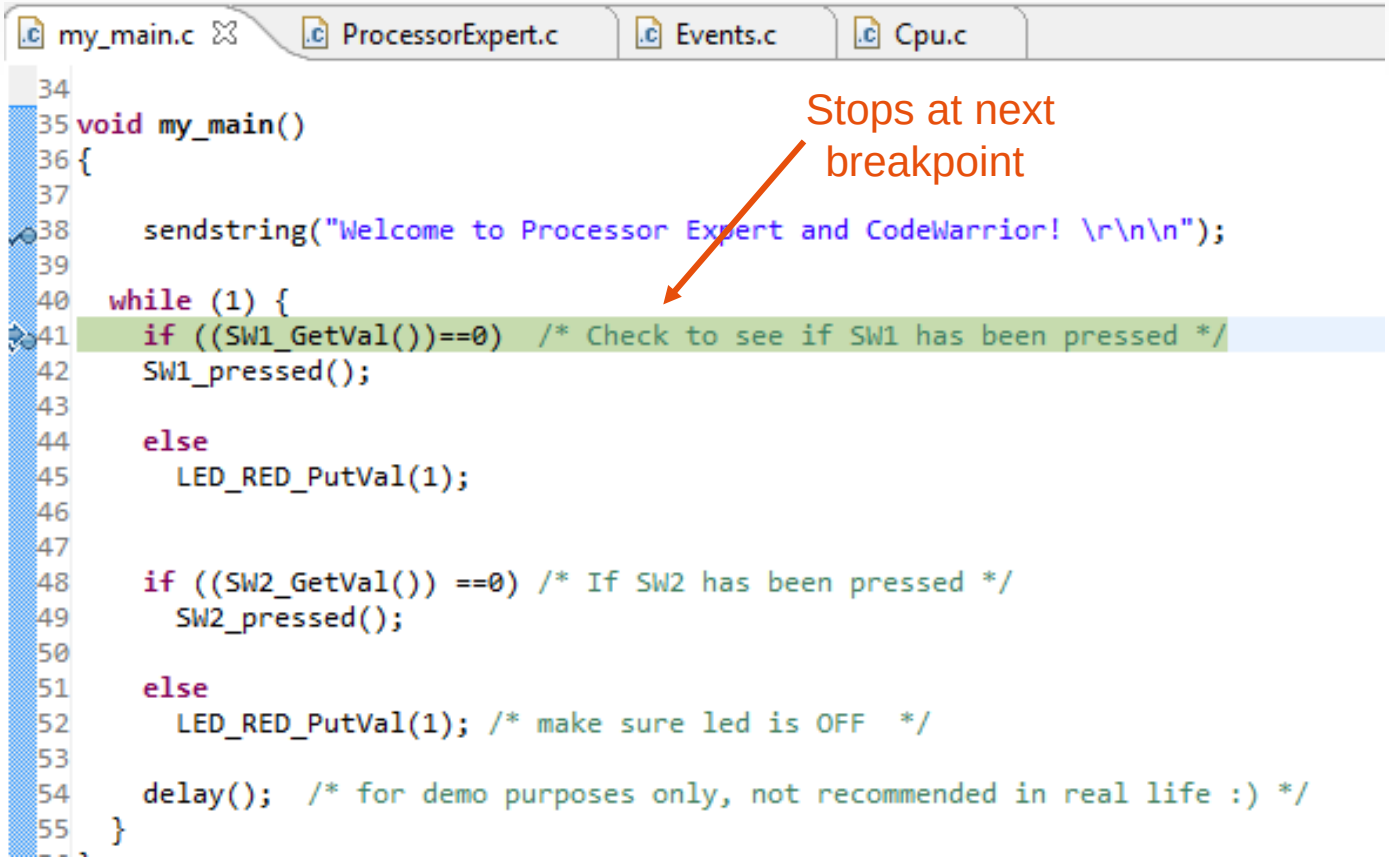
Application Test: Setting Breakpoints

- Click “Resume”

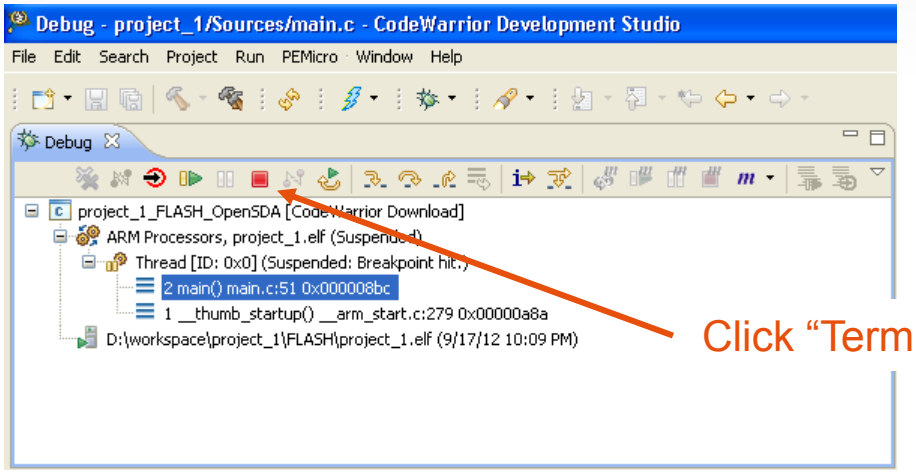
my_main.c | ProcessorExpert.c | Events.c | Cpu.c

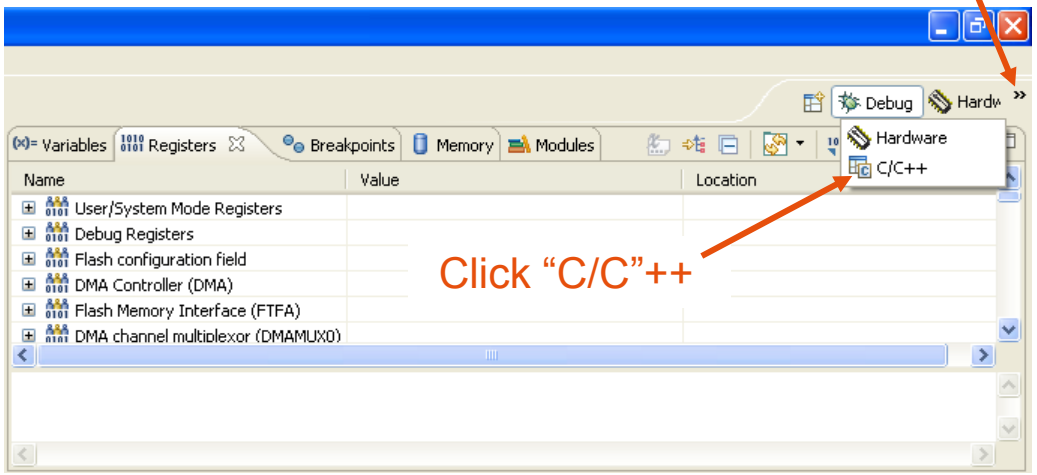
```
34
35 void my_main()
36 {
37
38     sendstring("Welcome to Processor Expert and CodeWarrior! \r\n\n");
39
40   while (1) {
41     if ((SW1_GetVal())==0)  /* Check to see if SW1 has been pressed */
42     SW1_pressed();
43
44     else
45       LED_RED_PutVal(1);
46
47
48     if ((SW2_GetVal()) ==0) /* If SW2 has been pressed */
49       SW2_pressed();
50
51     else
52       LED_RED_PutVal(1); /* make sure led is OFF  */
53
54     delay();  /* for demo purposes only, not recommended in real life :) */
55   }
```

Application Test: Setting Breakpoints



Application Test: Setting Breakpoints





Check Point: Create a New Project to Blink an LED

- This hands-on lab shows you how to…
  - Create a new project with the New Project Wizard ✓
  - Select and setup High Level Components ✓
  - Generate Processor Expert Code ✓
  - Import existing files ✓
  - Build the project ✓
  - Test the application's functionality ✓
  - **This concludes our Lab session**
  - **Question?**

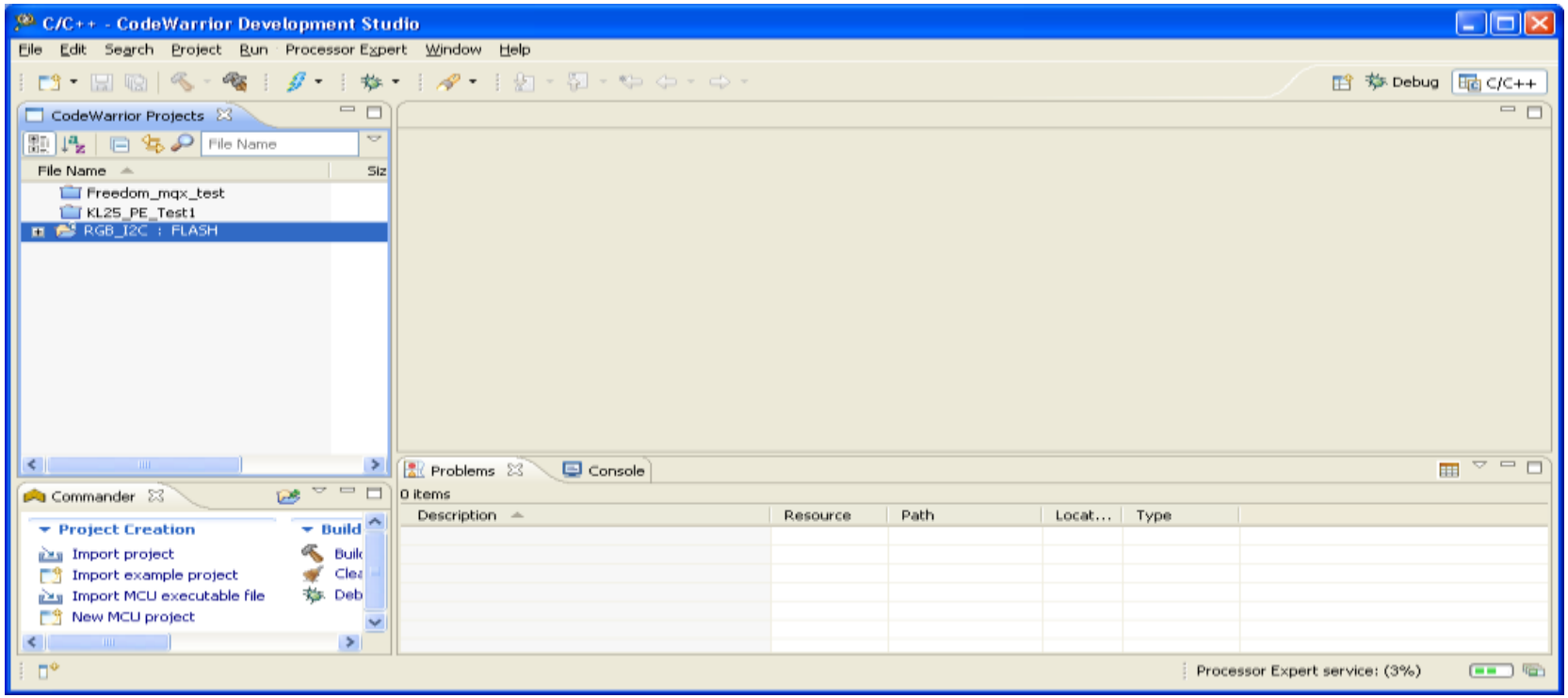

freescale™

Appendix

Basic Eclipse Terms

Workbench

- **Workbench** = desktop development environment
  - Contains all C/C++ development-related tools
  - Shows different perspectives of the working environment
  - Main window that appears when CodeWarrior Eclipse starts:



Workspace

- **Workspace** = directory that stores the source code, files and settings related to your work
  - Specify the workspace on startup
  - More than one project can be in a workspace
  - To switch workspaces select **File > Switch Workspace**

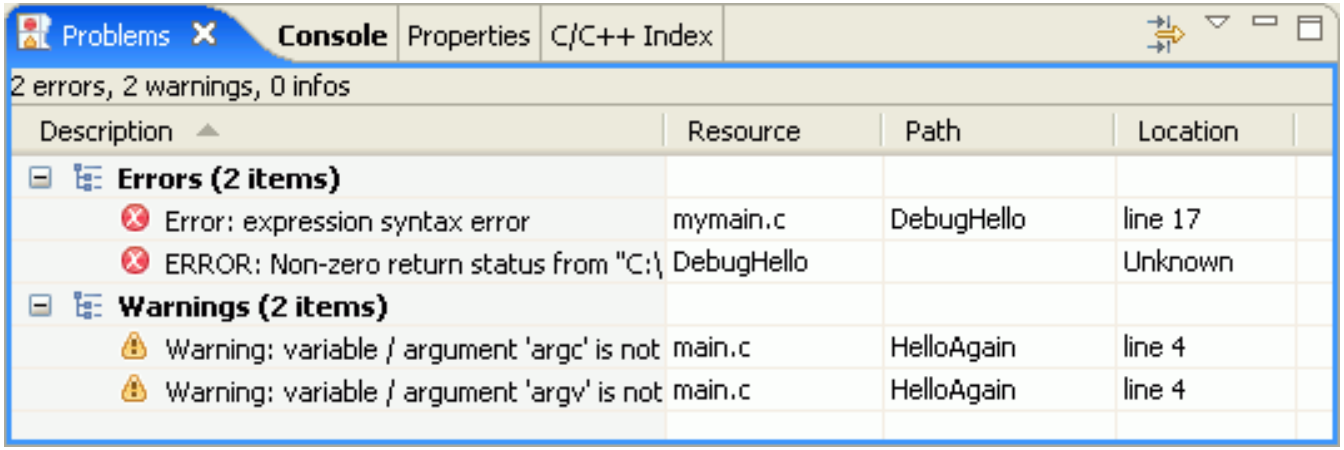
157

Project

- **Project** = container for organizing files and folders
  - All C/C++ work is done in the context of projects
  - CodeWarrior creates projects in new folders by default
  - New files and folders can be added to a project
    - **File > New > File**: Creates a new file in project directory within the workspace directory
    - **File > New > Folder**: Creates a new directory in project directory within the workspace directory.
    - Files dragged into a project are physically copied into the project directory
  - Files outside the project folder (or outside the workspace) can be linked to a project
    - **File > New > File > Advanced > Link to File System**: Creates new link in project directory within the workspace, which refers to a file in the file system
    - **File > New > Folder > Advanced > Link to File System**: Creates new link to a directory in project folder within the workspace. The link points to a directory in the user's file system. Creating this link will pull the directory and all sub-components into the CodeWarrior project.

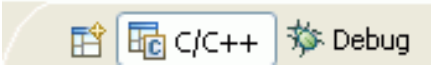
View

- **View** = a “visual” panel in the Workbench
  - Displays information about the contents of your workbench
  - Different views are provided for different tasks
    - Problems – Compile or other problems
    - Console – Standard run window console
    - C/C++ Projects – View of projects in workspace
    - Breakpoints – Debugging breakpoints in open projects
  - Editors are special “views” for editing files



Perspectives

- **Perspective** = collection of Eclipse views and action sets organized into a layout that suits an assigned task
  - Current perspective is highlighted at the top of the perspective window
  - Perspectives are shown on perspective shortcut bar (top right)



  - Click the perspective icon to switch among open perspectives
  - To open a perspective:
    - Select **Window > Open Perspective > [select** *-or-* **Other]**
    - On the perspective view, choose **Open Perspective > [select** *-or-* **Other]**

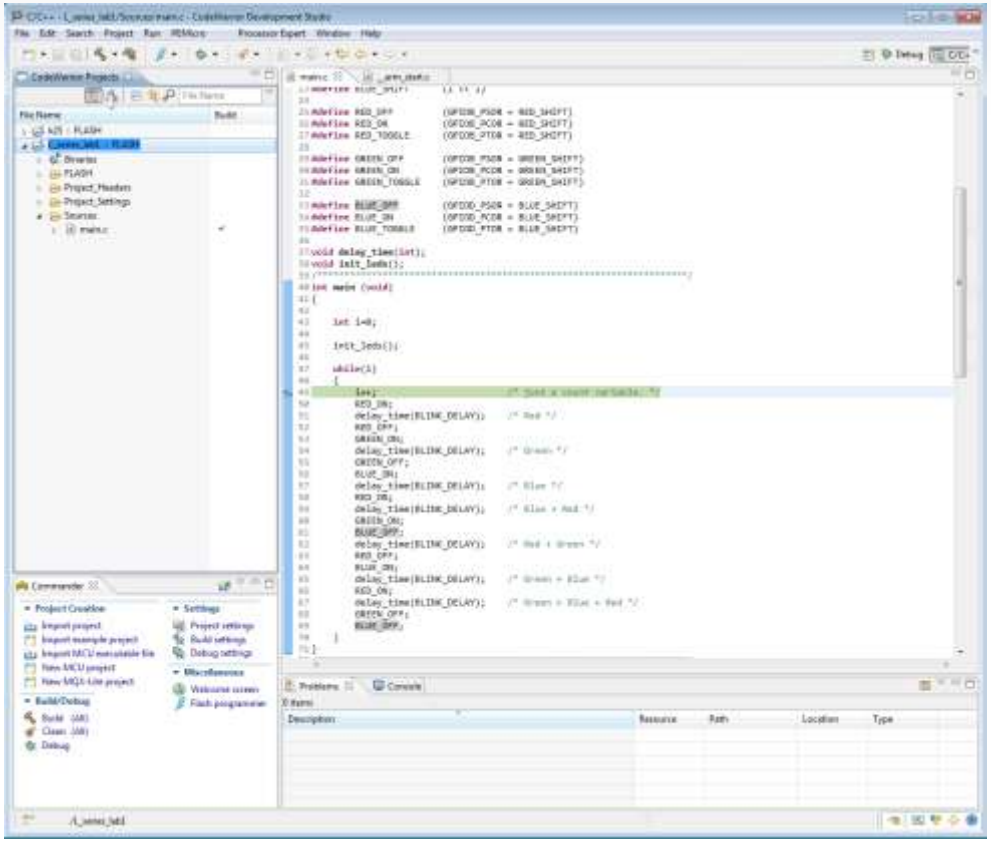
Open Perspective Icon 

  - To close a perspective:
    - Select **Window > Close Perspective**

C/C++ Perspective

C/C++ Perspective

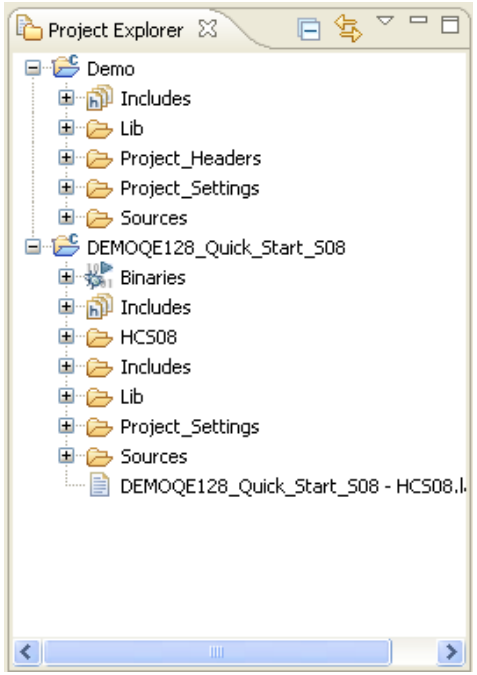
- Default views:
  - CodeWarrior Projects View
  - Editor (Area)
  - Problems View
  - Console View
  - Commander View



- Each view has an optional toolbar and menu that is separate from those on the main workbench window.

CodeWarrior Project View

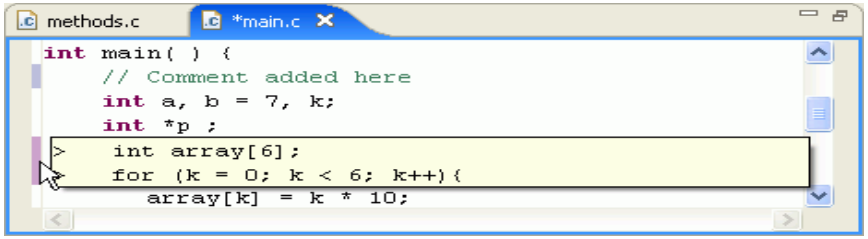
- File space (folder) organization for source and object files
- Project folders are at the top level, but you can drill down: Right-click folder > Go Into
- Filters types/status of resources to show (Toolbar menu > Filters)
- Other folders include:
  - Debug
  - Includes (compiler dependent) - contains makefiles
  - Binaries (generated by the compiler)



Editor Area

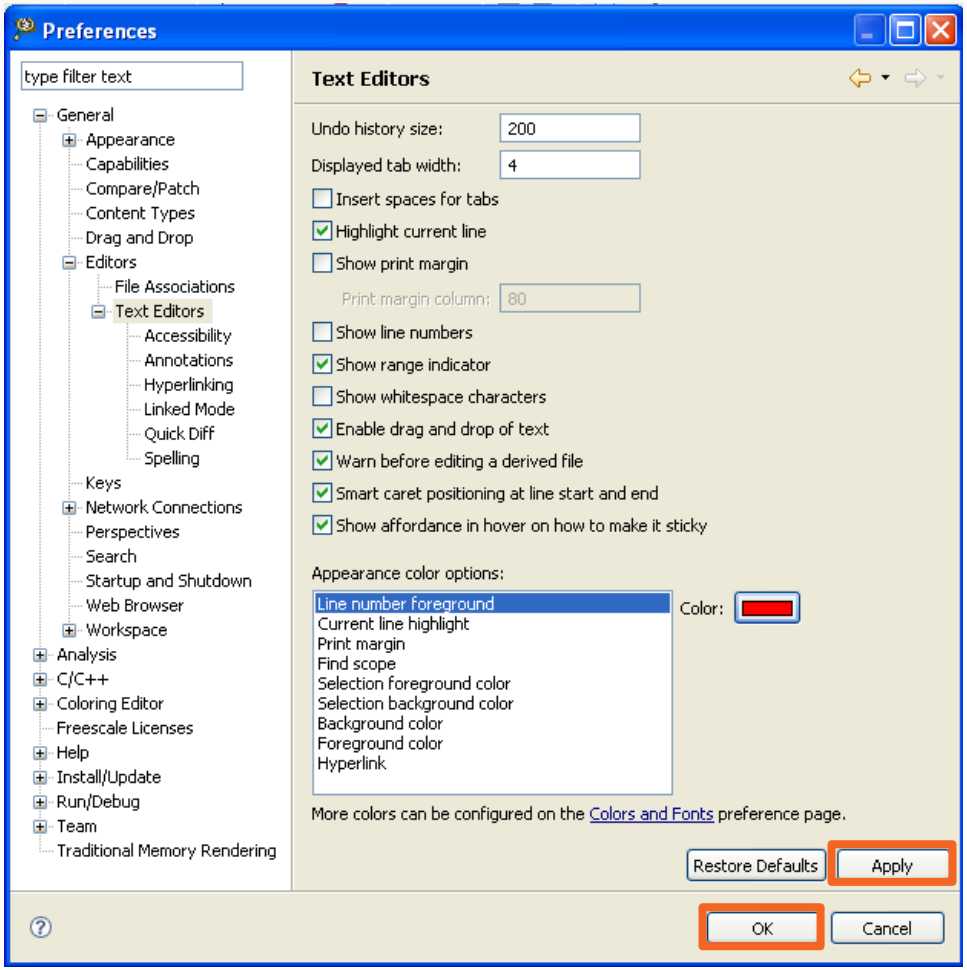
- There are multiple editors for multiple file types
  - C/C++ editor for source code
  - Text editor for text files
- To open a file:
  - Double click on the file in CodeWarrior Projects or Navigator view

    - or -
  - Select File > Open File
- Each file opens up in a new tab in editor area
- Editors show changes in files since last save
  - See the change bar (left side of text)
  - Additions and modifications are color coded
  - Hover cursor over change bar to see previous text
  - Dirty file – indicated by * in editor tab.
- Marker area on left side of editor indicates task, bookmarks or breakpoint



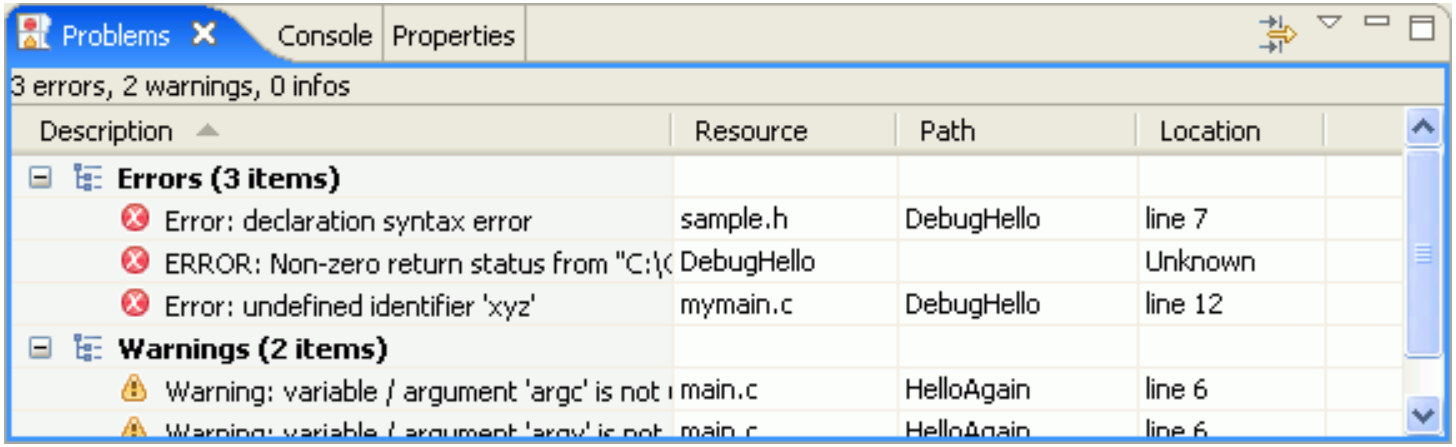
Configure Editors

- To configure editors:
  - Select **Window > Preferences**
    - Expand **General folder**
    - Select **Editors > Text Editors**
  - Configure options:
    - Show line numbers
    - Highlight current line
    - Color options
    - Etc.



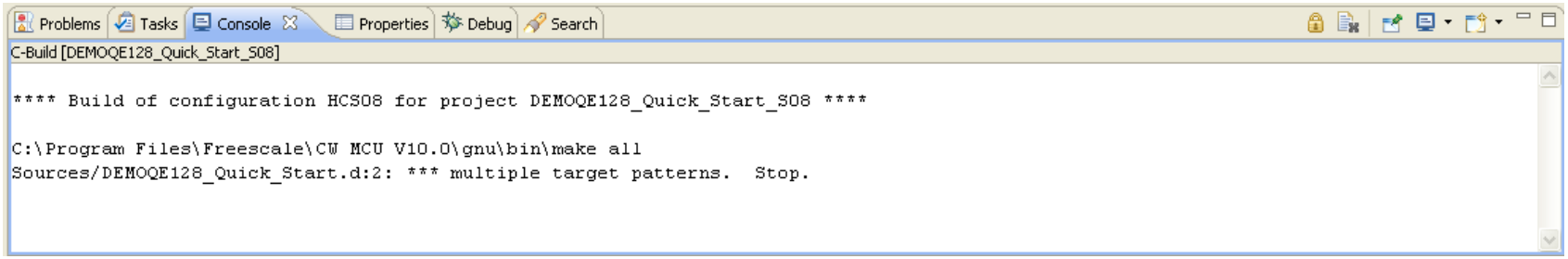
Problems View

- Shows errors and warnings
  - Filters on problems to display:
    - Location of defective resource
    - Type of defect
    - Type of problem
  - Double click an error/warning message to navigate to offending source code



Console View

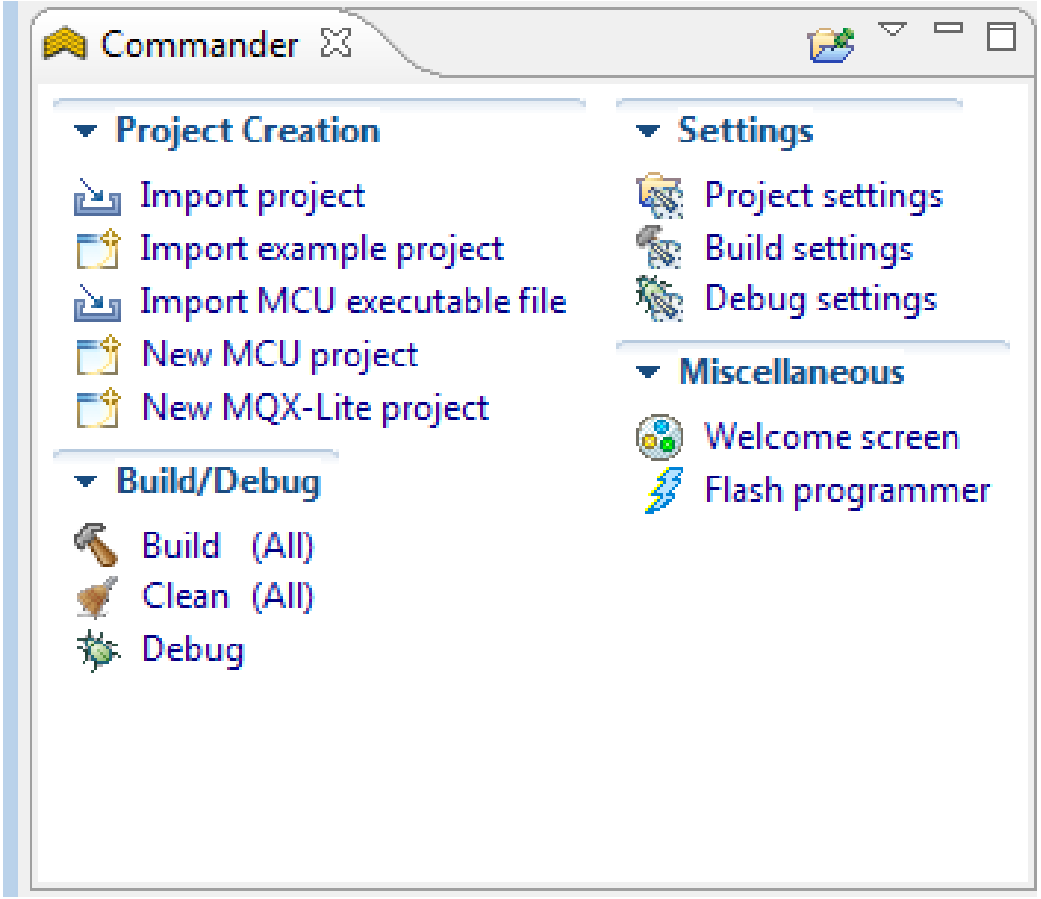
- Displays standard I/O
- Echoes Make files
- Displays program output
- Console toolbar:
  - Scroll Lock
  - Clear Console
  - Pin Console (save this console in separate view)
  - Display Selected Console
  - Open Console



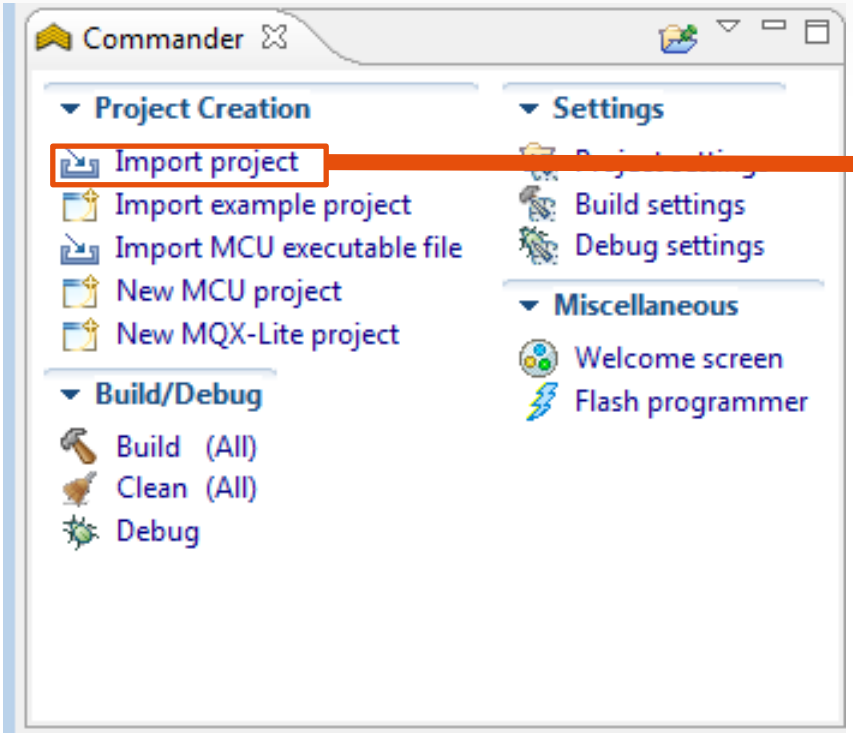
Commander View

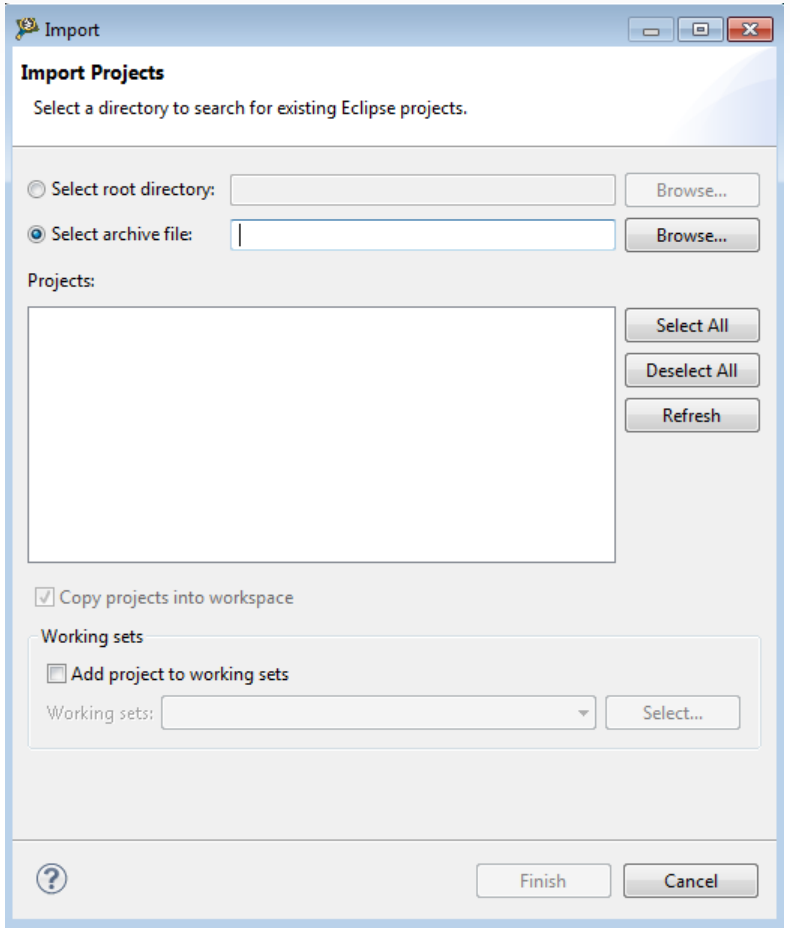
Commander View

- One click access to most common operations
  - Project Creation
  - Build/Debug
  - Settings
  - Miscellaneous

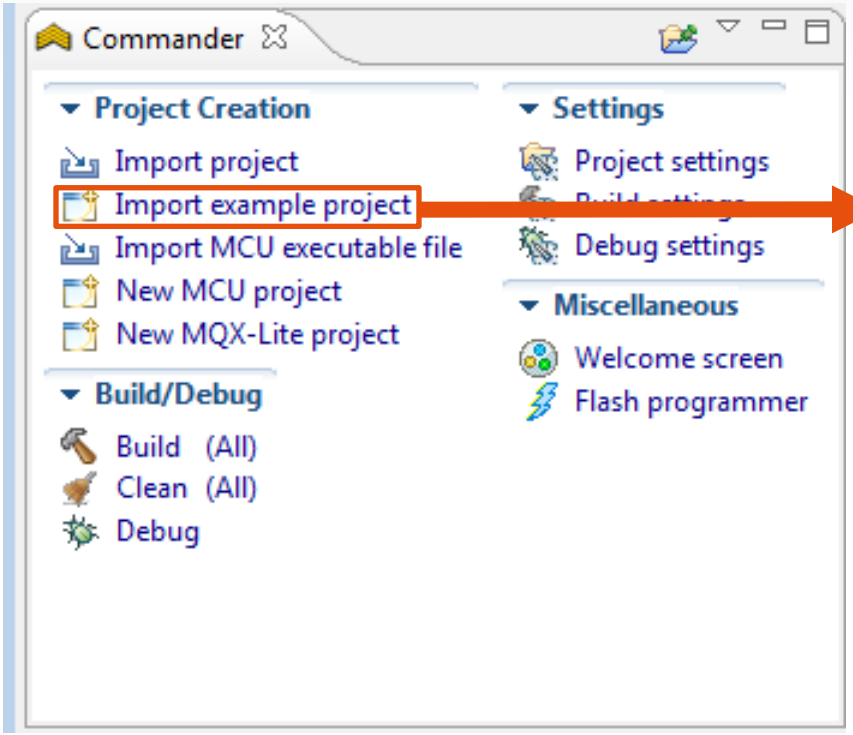


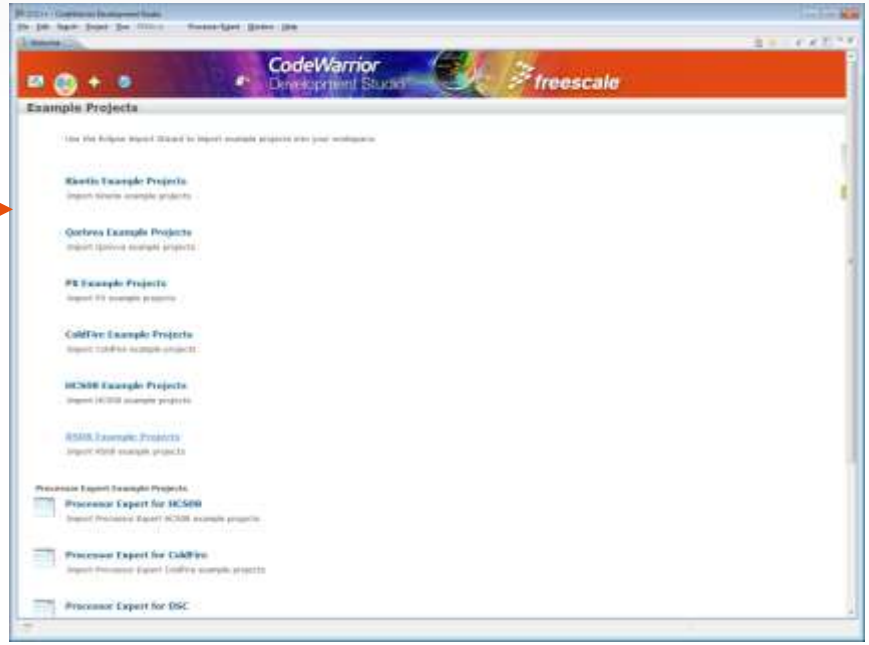
Commander View – Import Project



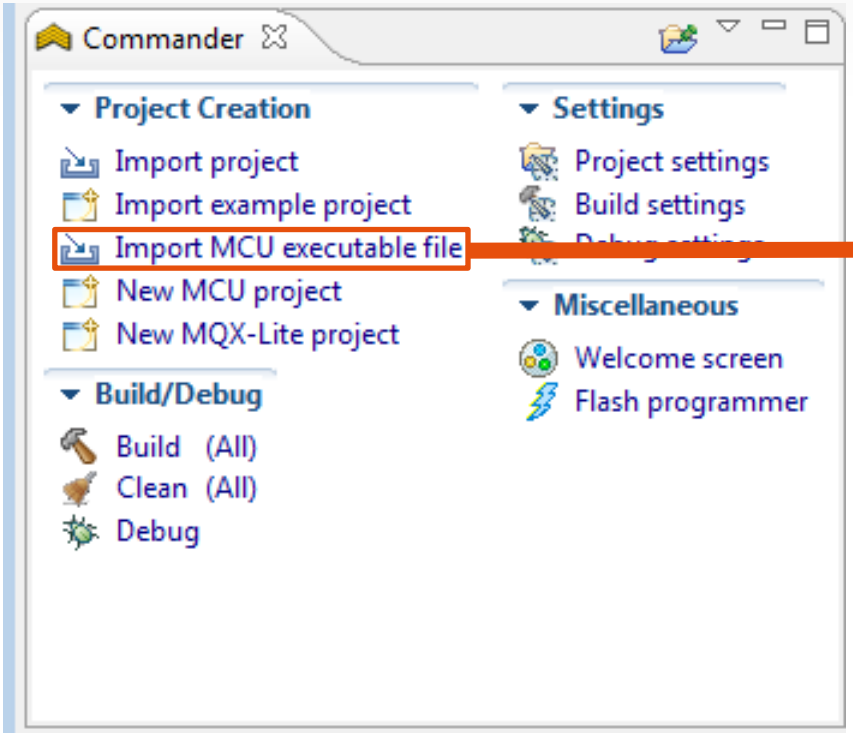


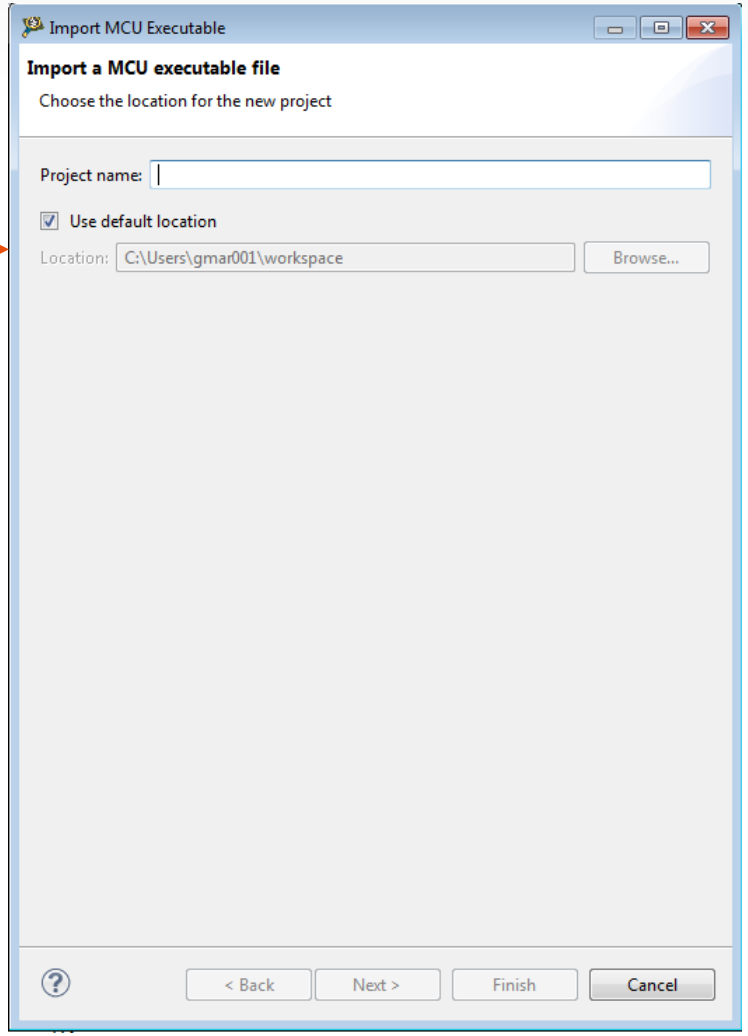
Commander View – Import Example Project



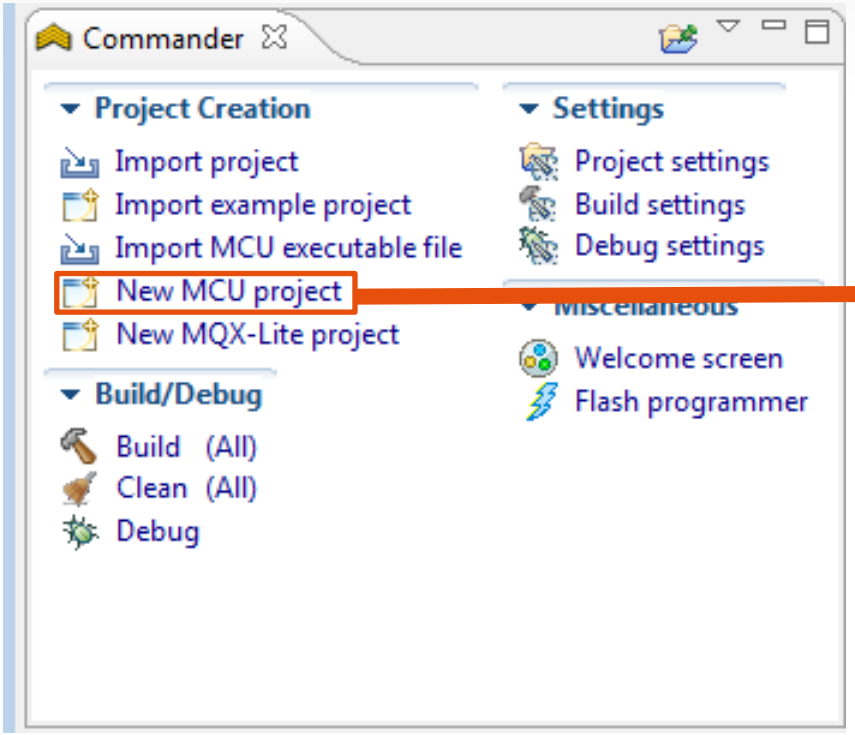


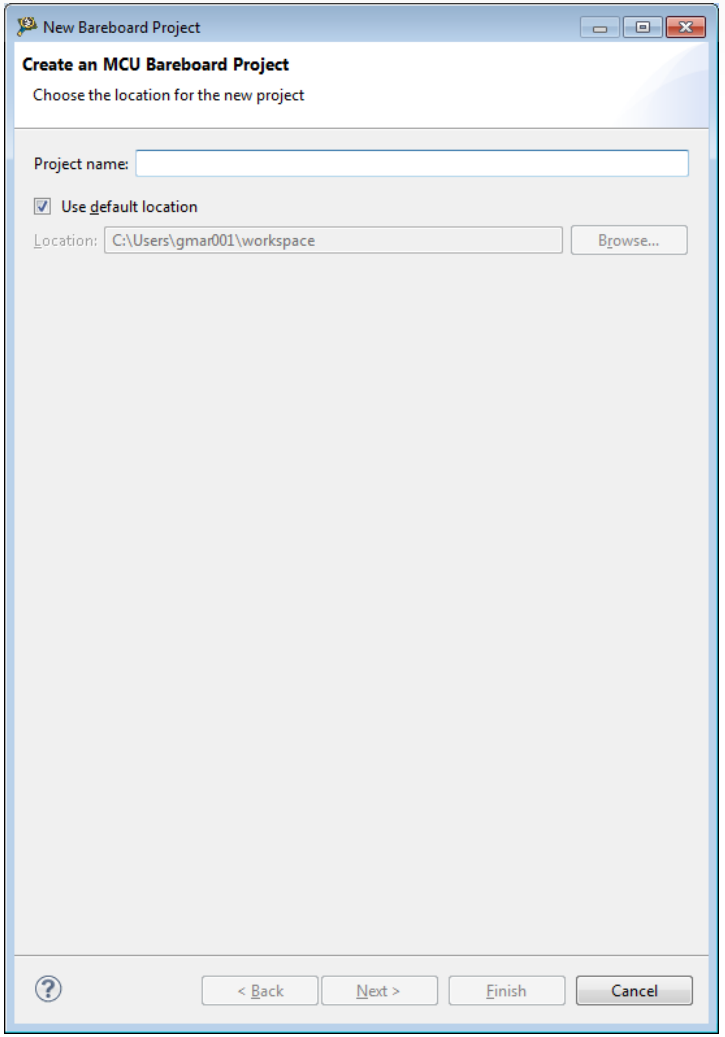
Commander View – Import MCU Executable File



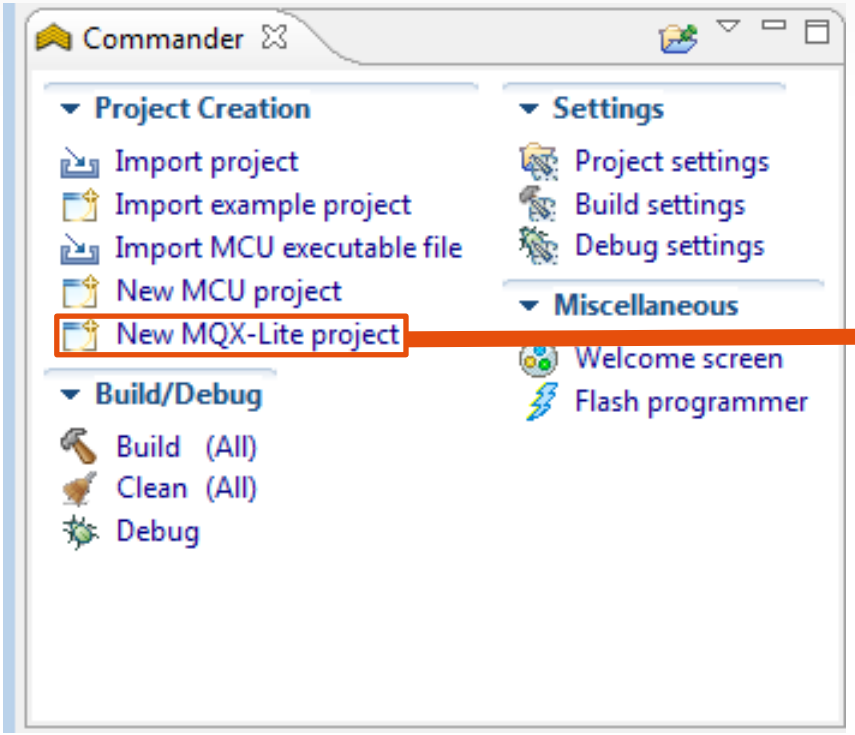


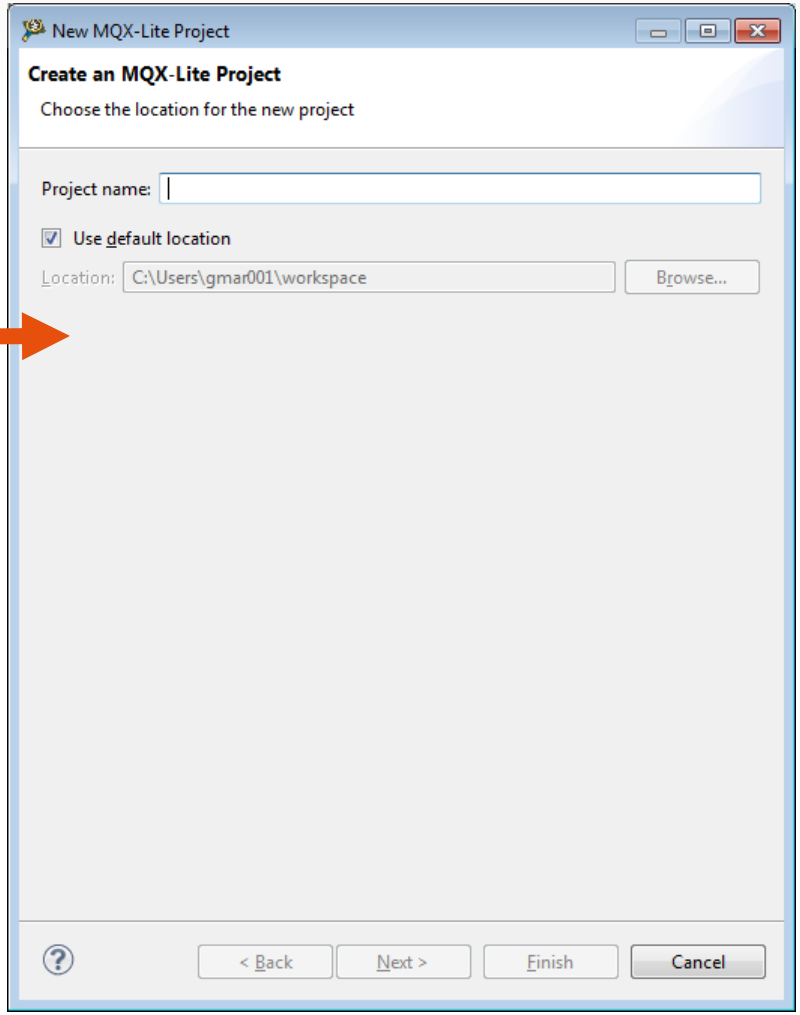
Commander View – New MCU Project



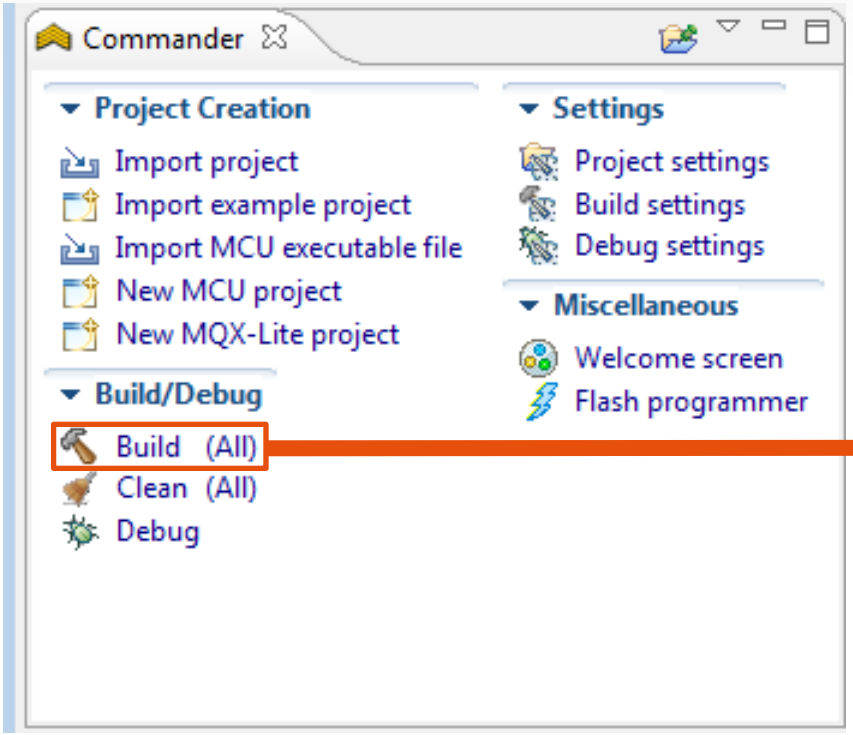


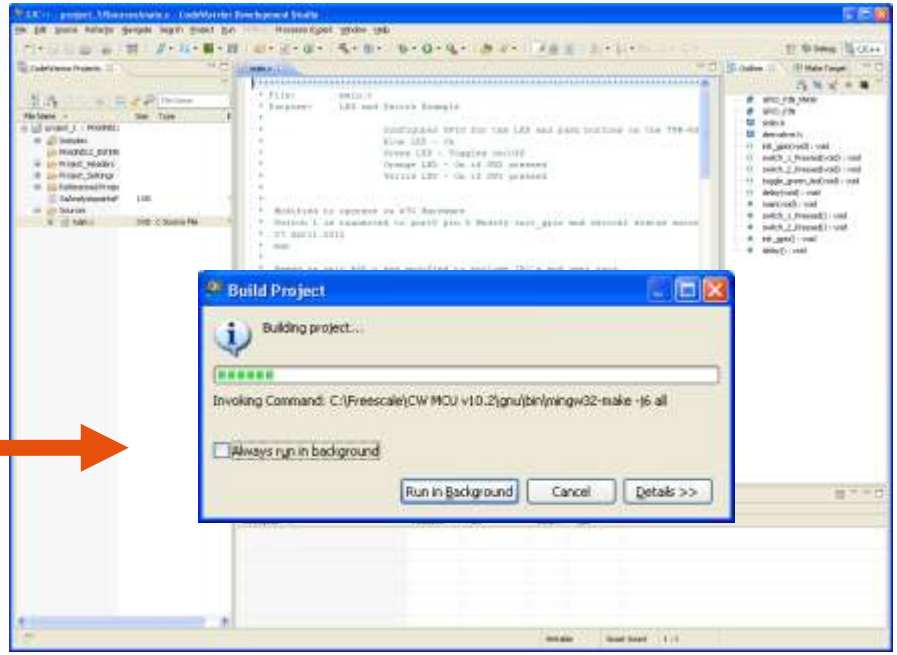
Commander View – New MQX-Lite Project



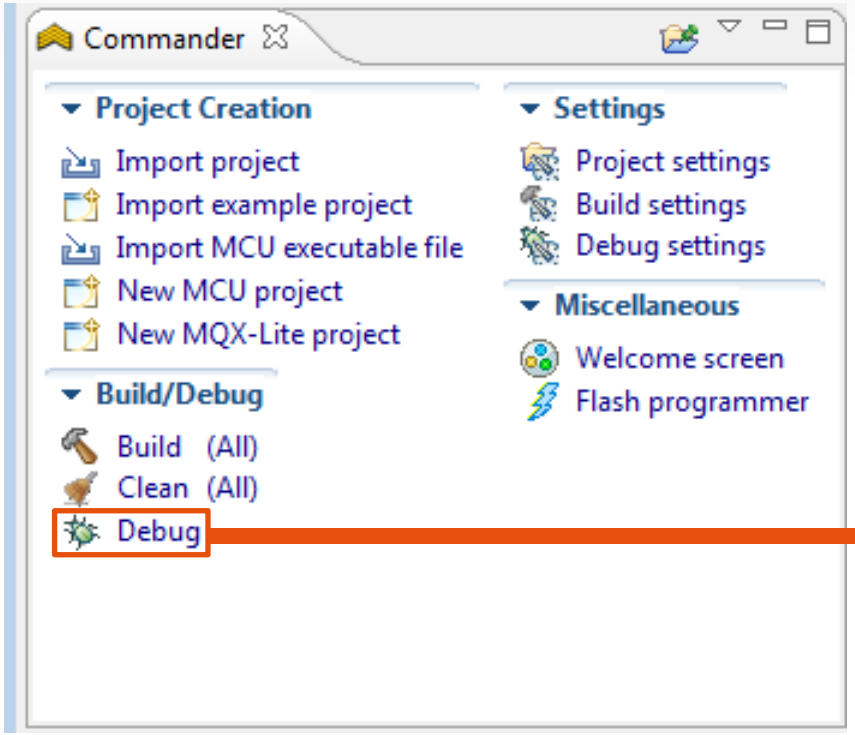


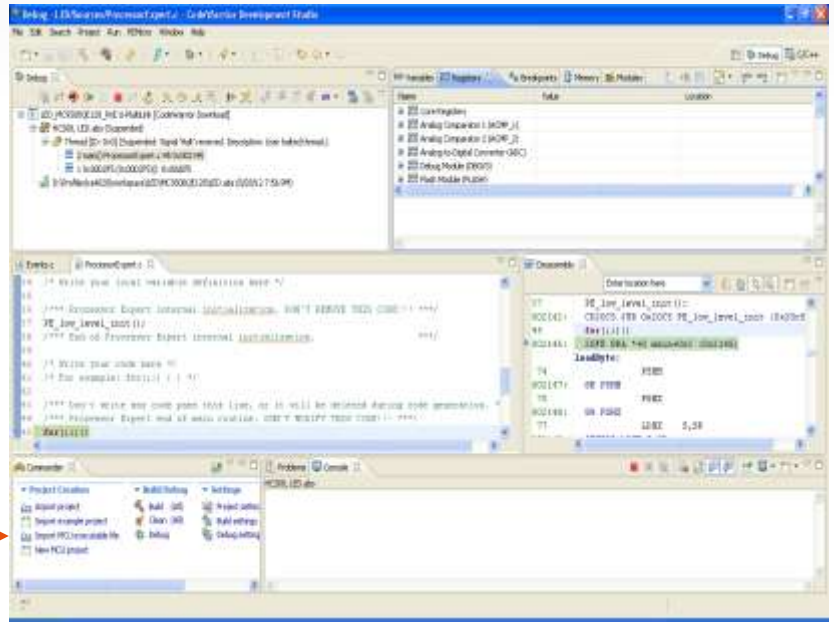
Commander View – Build



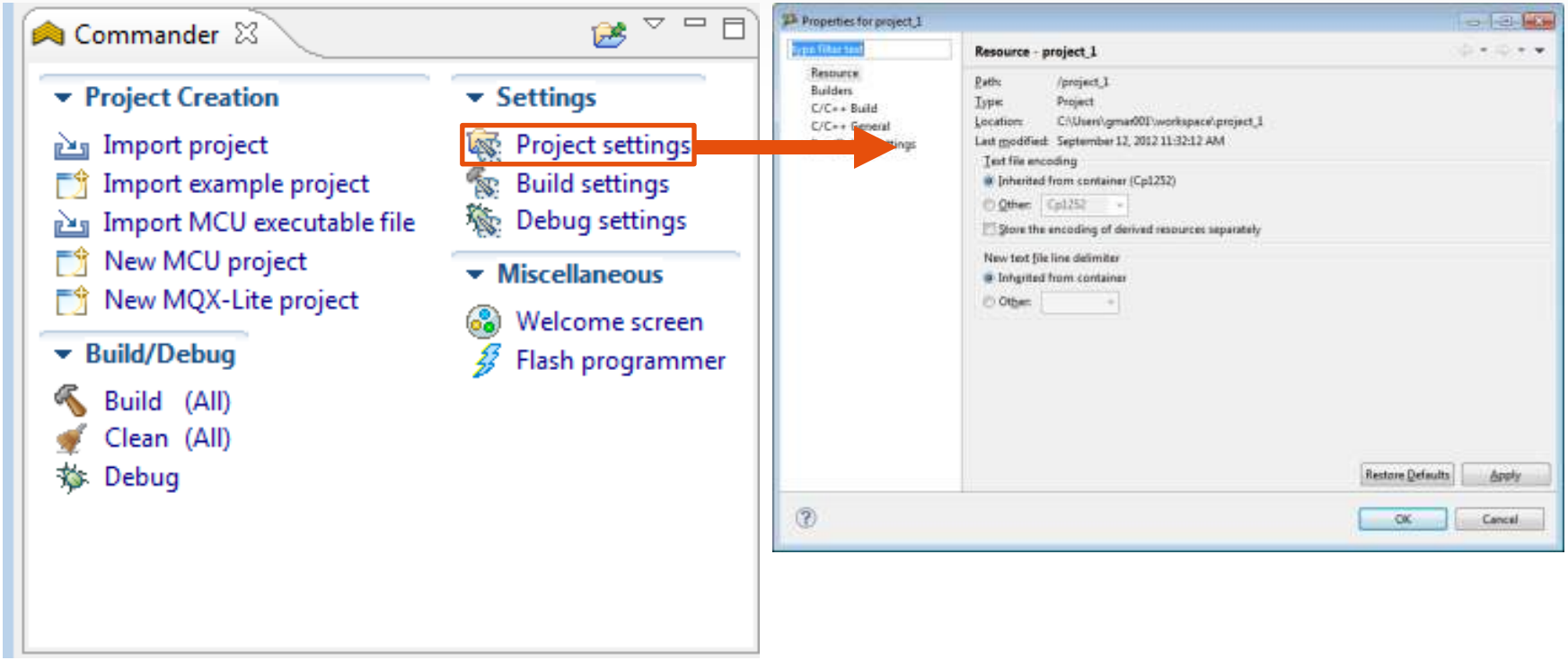


Commander View – Debug

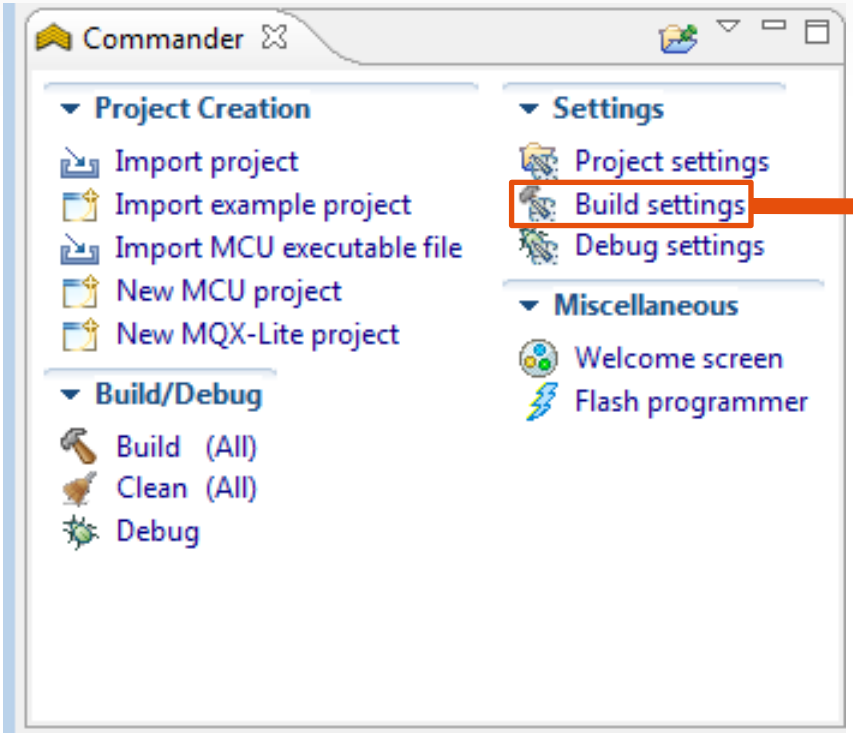


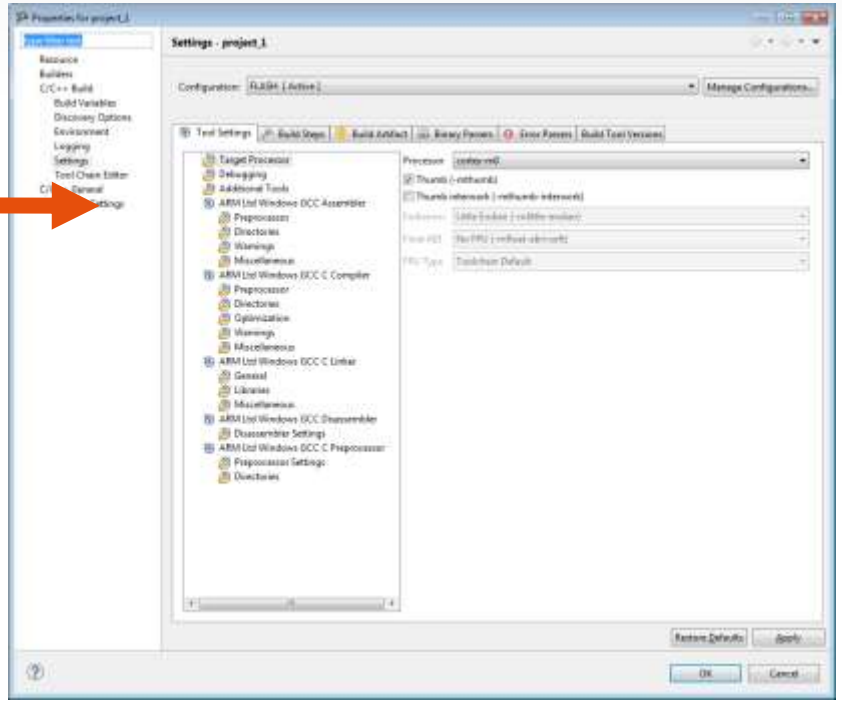


Commander View – Project Settings

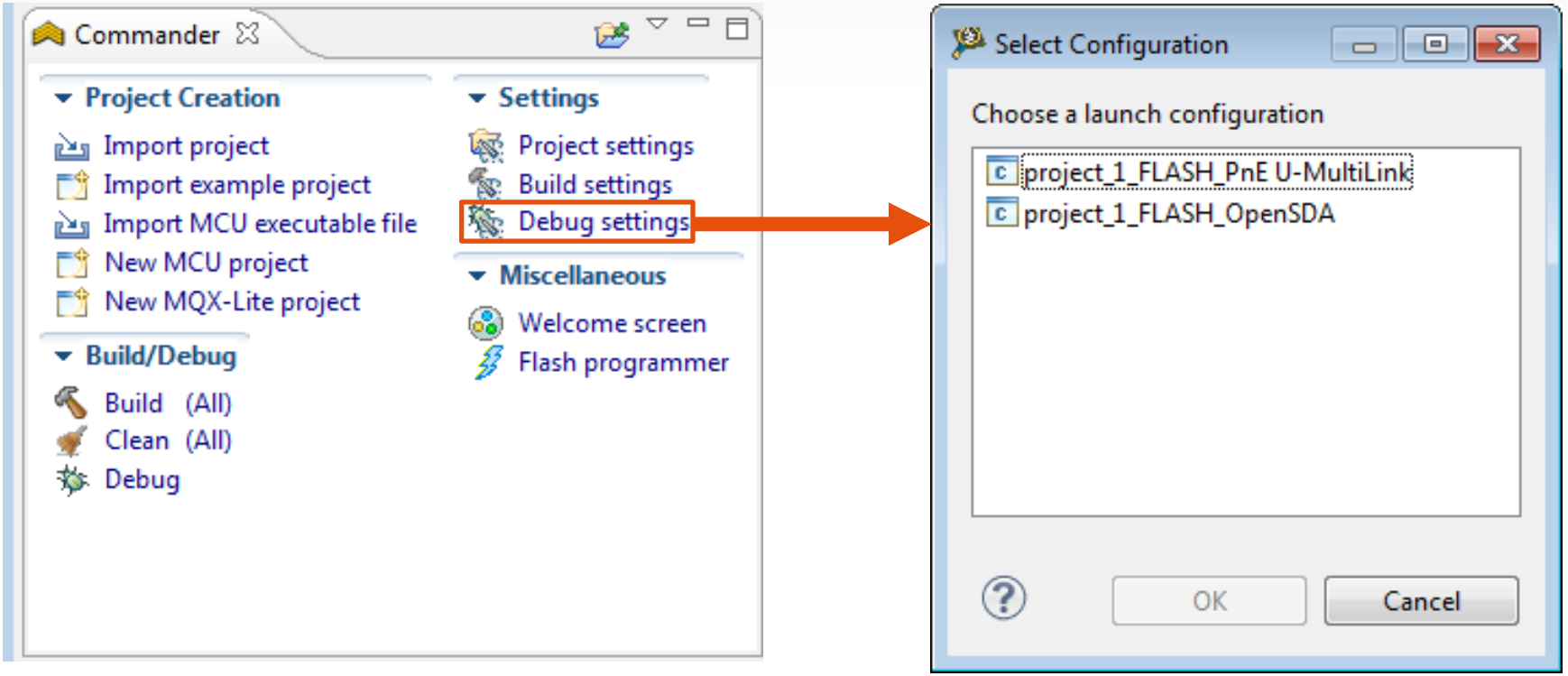


Commander View – Build Settings

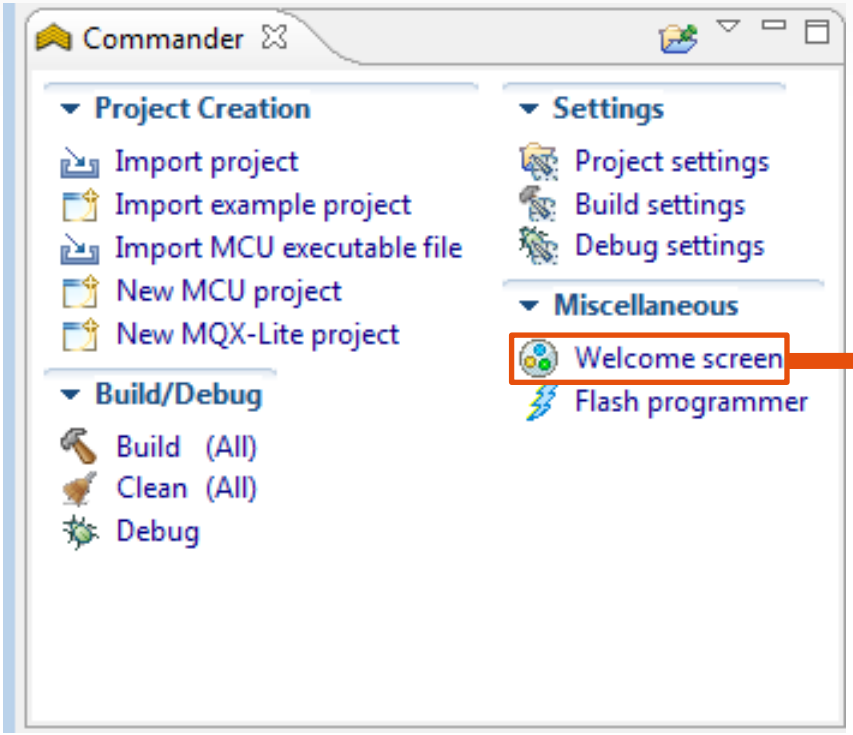


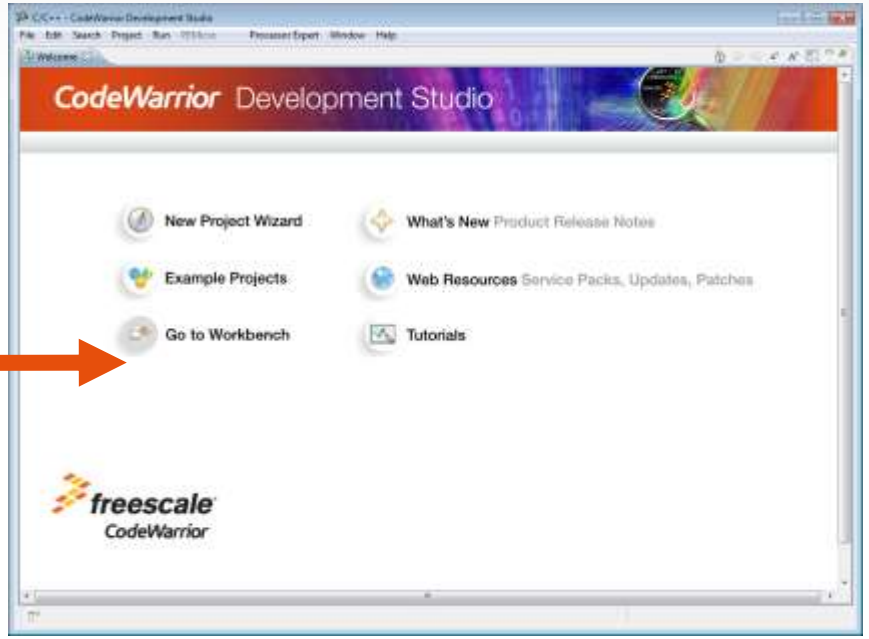


Commander View – Debug Settings

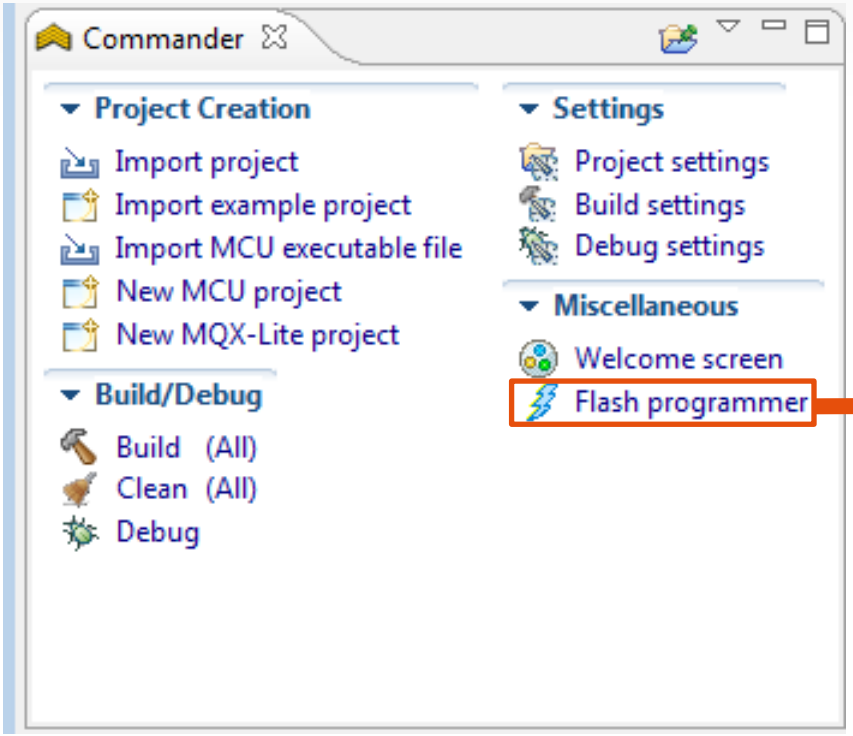


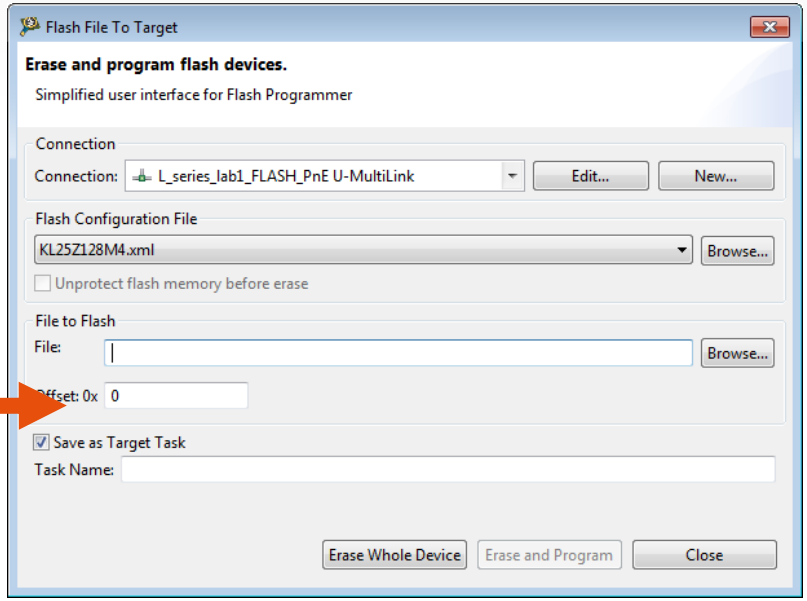
Commander View – Welcome Screen





Commander View – Flash Programmer

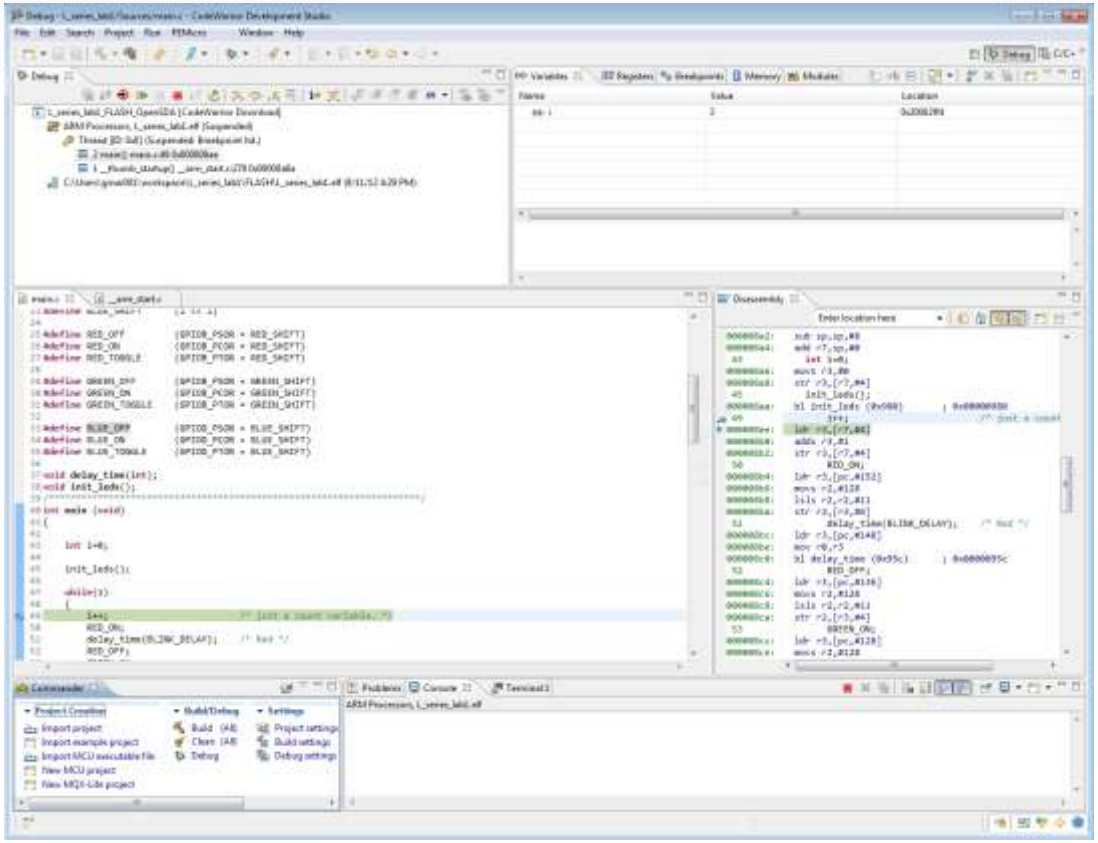




Debug Perspective

Debug Perspective

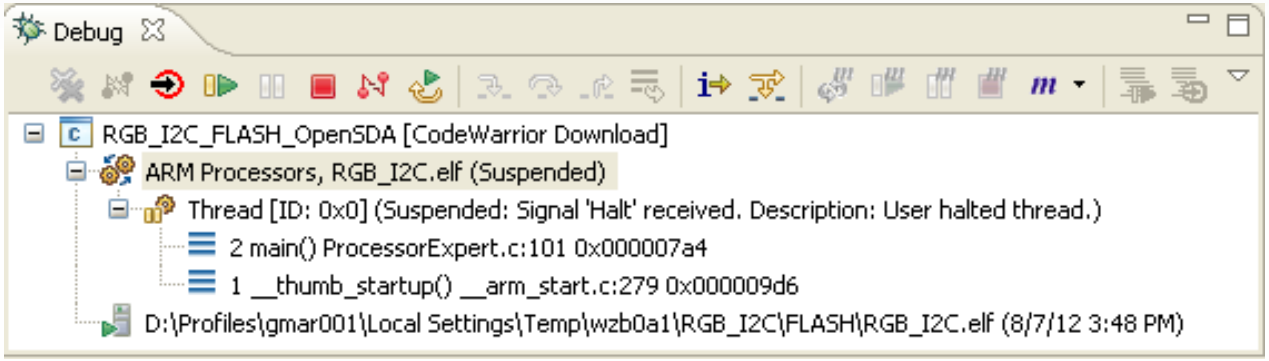
- Default views:
  - Debug View
  - Variables View
  - Breakpoints View
  - Memory View
  - Modules View
  - Editor (Area)
  - Disassembly View
  - Problems View
  - Console View
  - Commander View



- Each view has an optional toolbar and menu that is separate from those on the main workbench window.

Debug View

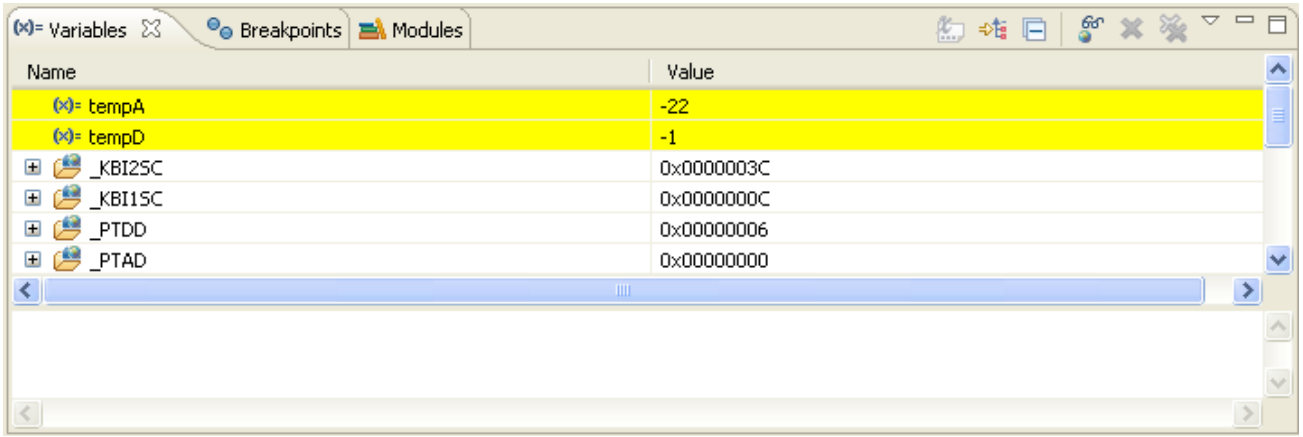
- Shows the target debugging information in a tree hierarchy



- Use this view to perform the following tasks:
  - Clear all terminated processes
  - Start a new debug session for the selected process
  - Resume execution of the currently suspended debug target
  - Halt execution of the currently selected thread in a debug target
  - Terminate the selected debug session and/or process
  - Detach the debugger from the selected process
  - Execute the current line, including any routines, and proceed to the next statement
  - Execute the current line, following execution inside a routine
  - Re-enter the selected stack frame
  - Examine a program as it steps into disassembled code

Variables View

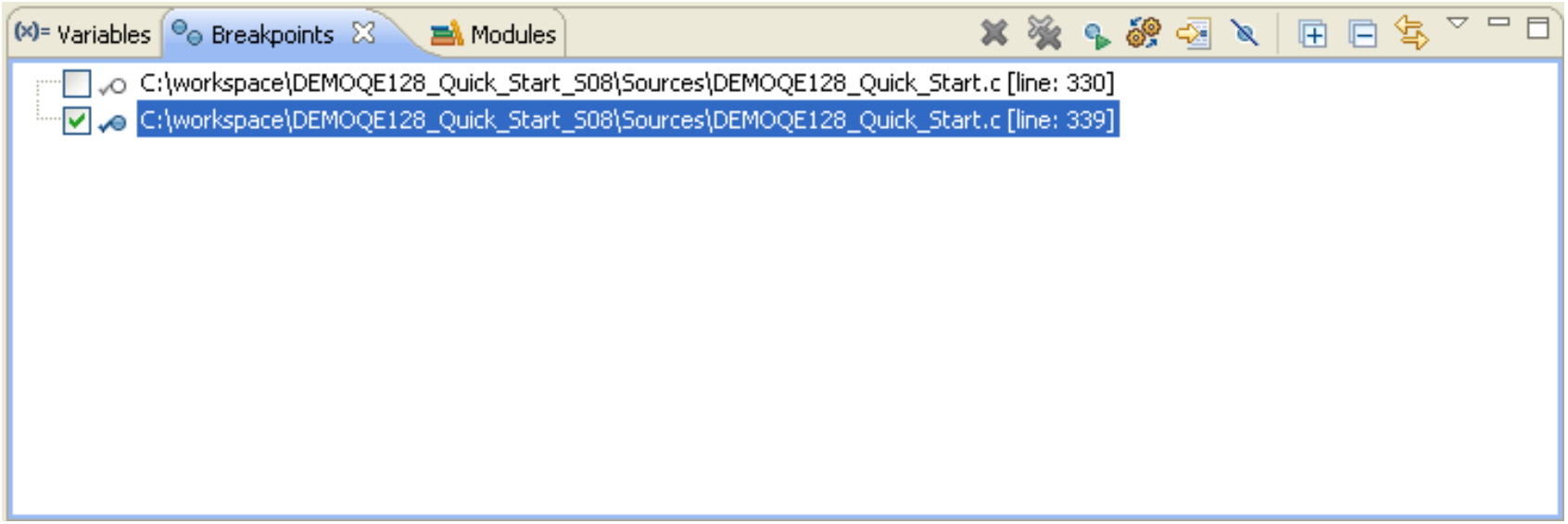
- Lists all global and static variables for each process



- Variables changed since last update are highlighted
- To add a global variable:
  - Select  in the Variable View
  - Select the global variables from the Global Variable dialog box

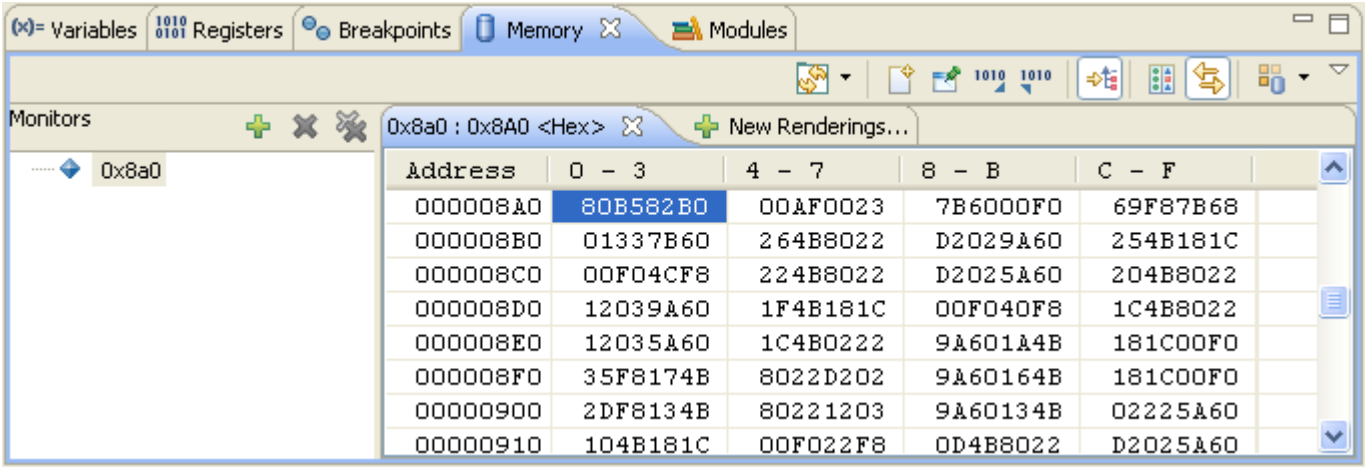
Breakpoints View

- Lists all breakpoints set in the workbench projects
- Allows breakpoints to be grouped by type, project, file, or working sets
- A breakpoint can temporarily be disabled without losing the information it contains



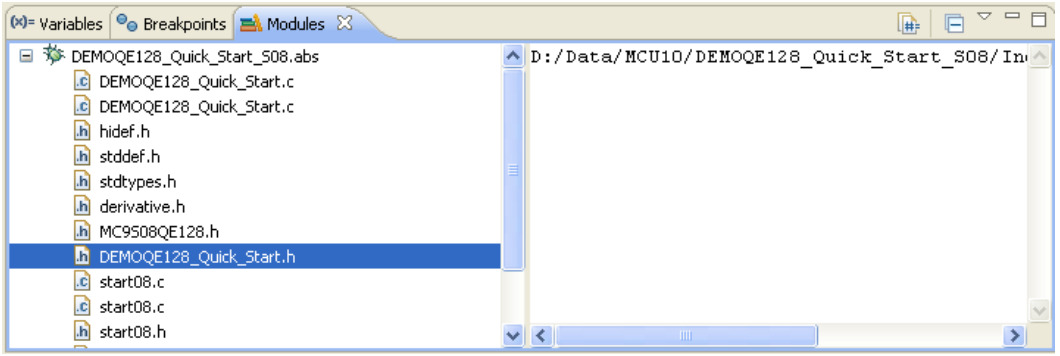
Memory View

- Allows memory to be monitored and modified
- Contains two panes:
  - Monitors panel - Displays the list of memory monitors added to the debug session currently selected in the Debug view
  - Renderings panel - Displays memory renderings.
    - Content is controlled by the selection in the Monitors panel.
    - Can be configured to display two renderings simultaneously.



Modules View

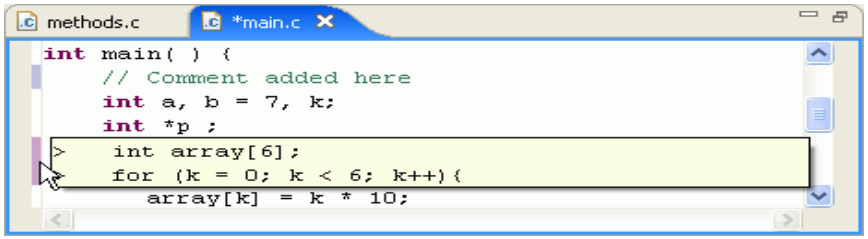
- Displays information about the modules loaded in the current debug session
  - Executables
  - Shared libraries
- View consists of two areas:
  - Modules tree
  - Detail pane



- Detail pane displays the detail information for the module selected in the modules tree
- Expanding a module enables users to view the module's internals
  - Functions
  - Global variables
  - Associated source files

Editor Area

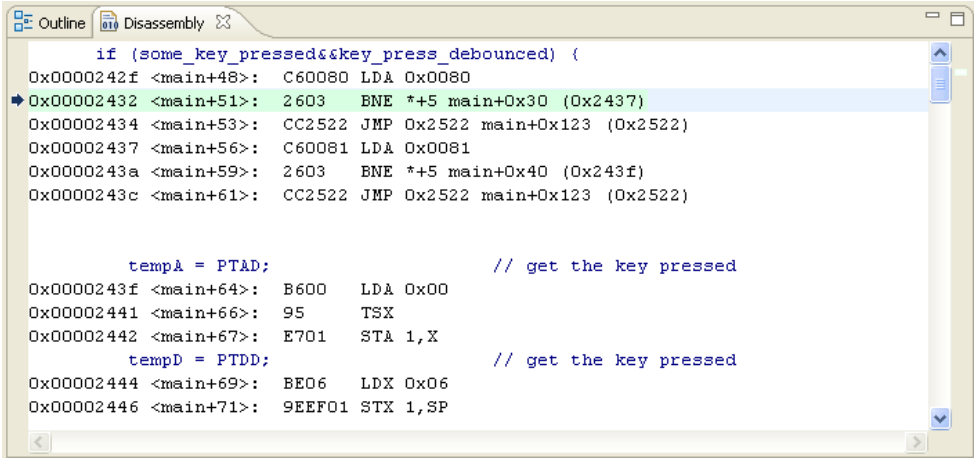
- There are multiple editors for multiple file types
  - C/C++ editor for source code
  - Text editor for text files
- To open a file:
  - Double click on the file in CodeWarrior Projects or Navigator view
    - or -
  - Select File > Open File
- Each file opens up in a new tab in editor area
- Editors show changes in files since last save
  - See the change bar (left side of text)
  - Additions and modifications color coded.
  - Hover cursor over change bar to see previous text
  - Dirty file – indicated by * in editor tab



- Marker area on left side of editor indicates task, bookmarks, or breakpoint

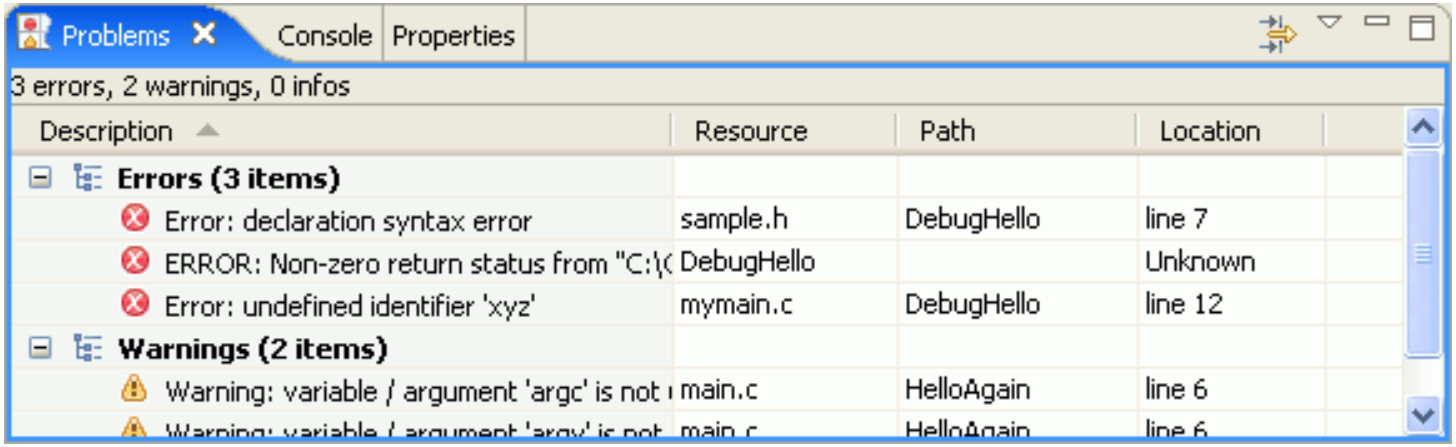
Disassembly View

- Shows the loaded program as assembler instructions mixed with source code for comparison
- The currently executing line is indicated by an arrow marker and highlighted in the view
- The following tasks can be performed:
  - Set breakpoints at the start of any assembler instruction
  - Enable and disable breakpoints and set their properties
  - Step through the disassembly instructions of the program
  - Jump to specific instructions in the program



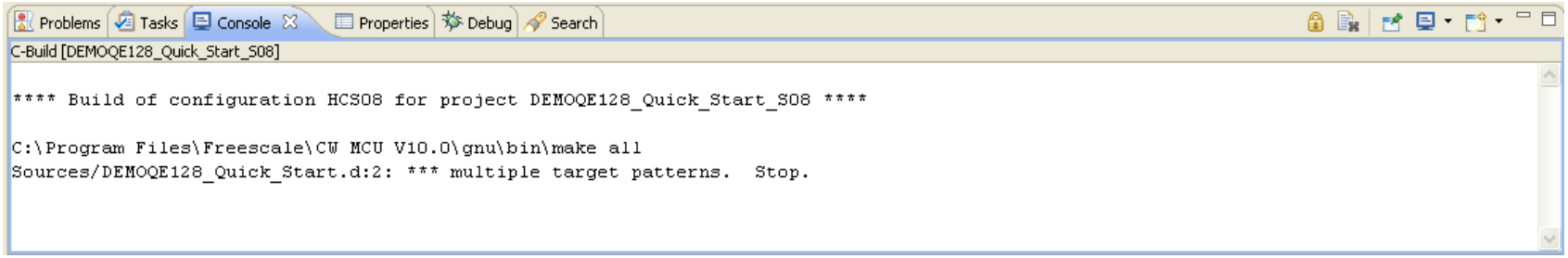
Problems View

- Shows errors and warnings
  - Filters on problems to display:
    - Location of defective resource
    - Type of defect
    - Type of problem
  - Double click an error/warning message to navigate to offending source code



Console View

- Displays standard I/O
- Echoes Make files
- Displays program output
- Console toolbar:
  - Scroll Lock
  - Clear Console
  - Pin Console (save this console in separate view)
  - Display Selected Console
  - Open Console




freescale™