Hands-On Workshop

Kinetis Dev Tools: KDS, KSDK, PE, FreeMASTER

Erich Styger

May 2015

Public

Overview

- Overview
  - Kinetis SDK V1.2.0
  - Kinetis Design Studio V3.0.0

- Hands-On (FRDM-K64F)
  - Kinetis Design Studio
  - Kinetis SDK
  - Processor Expert
  - FreeMASTER

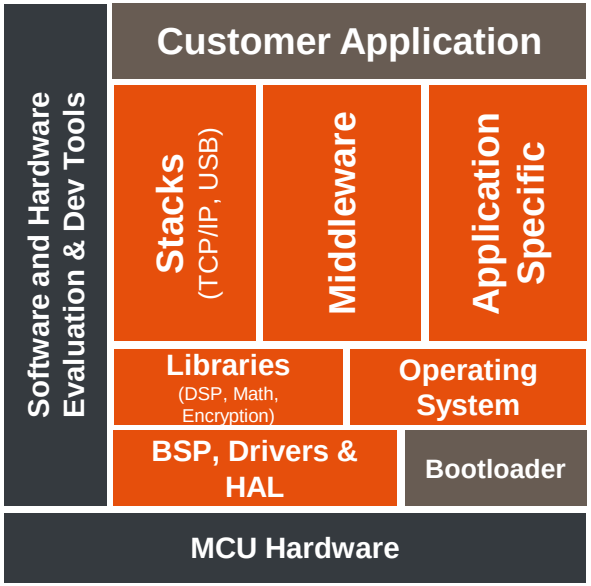
Kinetis Software Development Kit (SDK)

New V1.2.0

SDK

A software framework for application development across all Kinetis MCUs

Hardware abstraction, peripheral drivers, stacks, RTOS's, utilities, and usage examples; delivered in C source



Open Source Initiative

CMSIS COMPLIANT ARM® Cortex™ Microcontroller Software Interface Standard

CERTIFIED USB

Product Features

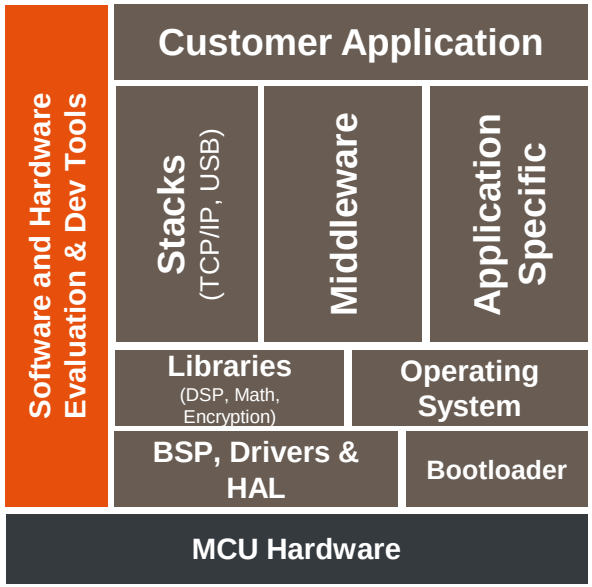
- Open source hardware abstraction layer (HAL) provides APIs for all Kinetis hardware resources
- BSD-licensed set of peripheral drivers with easy-to-use C-language APIs
- Comprehensive HAL and driver usage examples and sample applications for RTOS and bare-metal
- GUI configurable projects and peripheral drivers using Processor Expert
- CMSIS-CORE compatible startup plus CMSIS-DSP library and examples
- RTOS Abstraction Layer (OSA) with support for Freescale MQX, FreeRTOS, Micrium uC/OS, and bare-metal
- Integrates new Freescale unified USB stack, open source TCP/IP stack (lwIP), open source FAT file system, encryption math/DSP libraries, and more and
- Support for multiple toolchains: GNU GCC, IAR, Keil, Atollic, and Kinetis Design Studio

Kinetis Design Studio

New V3.0.0

No-cost integrated development environment (IDE) for Kinetis MCUs

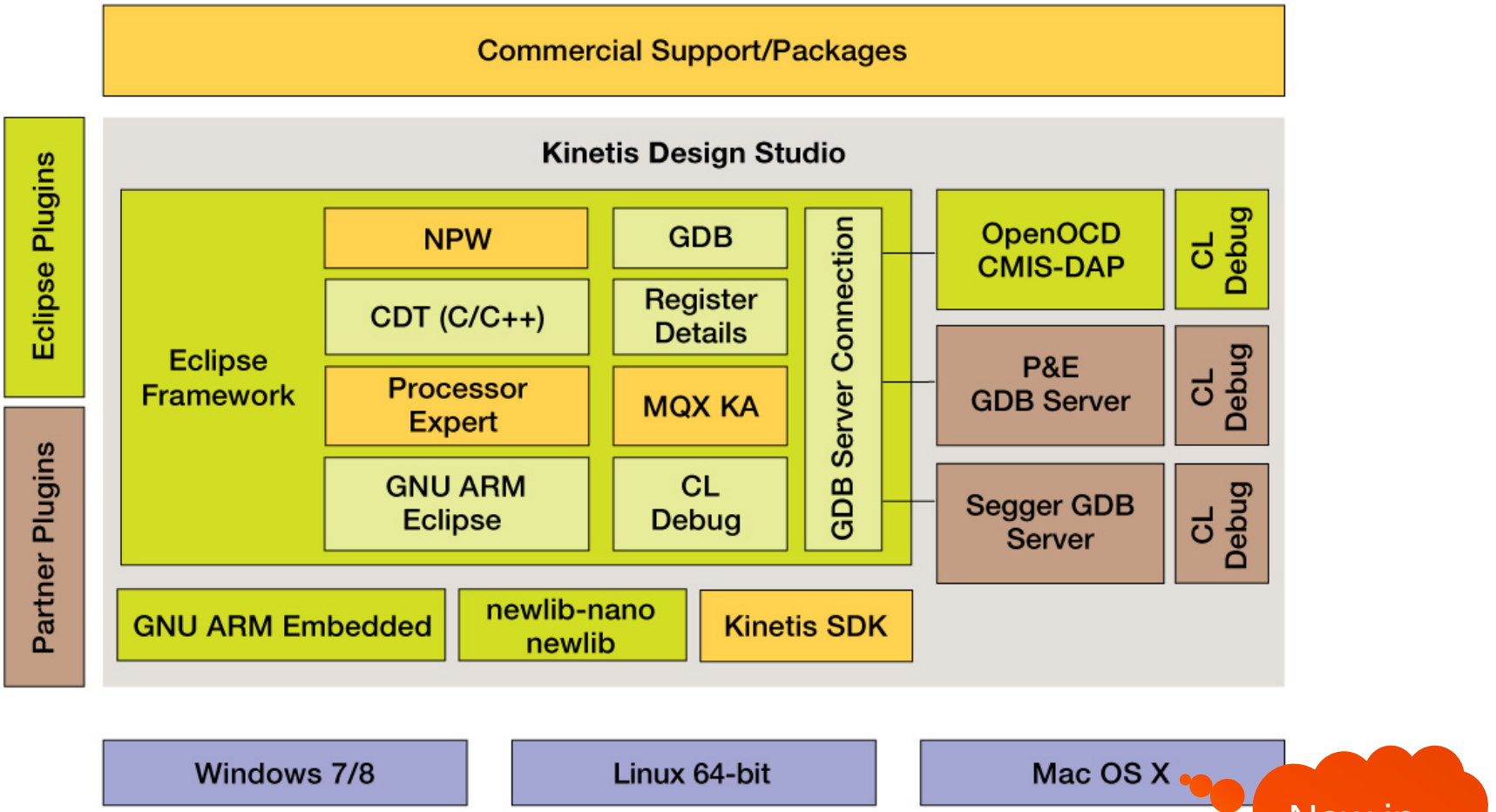
Eclipse and GCC-based IDE for C/C++ editing, compiling and debugging



Product Features

- ~~A~~ free of charge and unlimited IDE for Kinetis MCUs
- A basic IDE that offers robust editing, compiling and debugging
- Based on Eclipse, GCC, GDB and other open-source technologies
- Includes Processor Expert with Kinetis SDK integration
  - Supports all existing Kinetis devices via Processor Expert and new project wizard
  - All new Kinetis devices will also feature the Kinetis SDK with Processor Expert configurability
- Host operating systems:
  - Windows 7/8 (32 and 64-bit)
  - Linux (Ubuntu, Redhat, Centos) (64bit)
  - Mac OS X (with v3.0) and Segger
- Support for SEGGER, P&E and Open SDA/CMSIS-DAP debugger targets
- Support for Eclipse plug-ins including RTOS-awareness (i.e. MQX, FreeRTOS)

Kinetis Design Studio Block Diagram



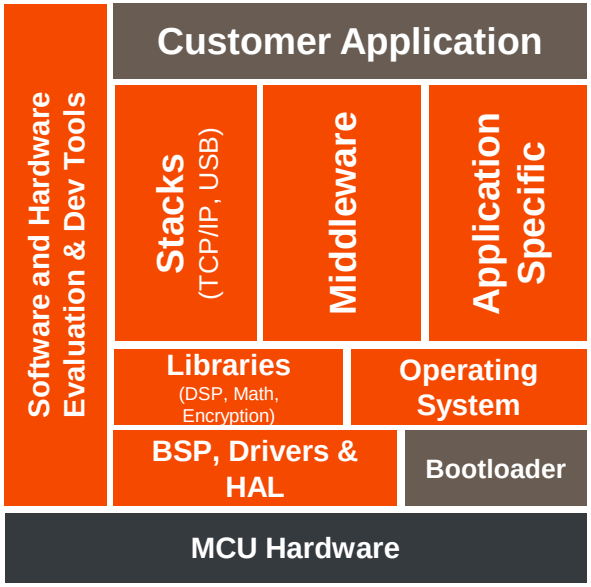
KDS V3.0.0

- NEW Release: Launched 6-May-2015
- Hosts
  - Windows 7/8 (32bit and 64bit)
  - Linux 64bit (Ubuntu 14.04, RedHat/Centos7) (32bit host libs for Launchpad)
  - Mac OS X ("Yosemite") and Segger Run Control
- Eclipse Luna 4.4 Framework
  - Welcome view, Split view, Black Theme, …
- Processor Expert for Kinetis V3.0
  - Simpler and Easier SDK usage (NPW)
  - Component Repositories
  - Keil and IAR external IDE/build support
- Launchpad GNU Tools for ARM Embedded (4.8)
  - Newlib nano for devices <2 KByte RAM
- Improved Debugging support
  - Updated Segger and P&E (same OpenOCD as in v2.0.0)
  - JTAG Daisy Chaining, Memory Range Protection, Attach to running target
  - Register Details Viewer

Freescale Processor Expert Software

Create, configure, generate software and drivers for Freescale microcontrollers.

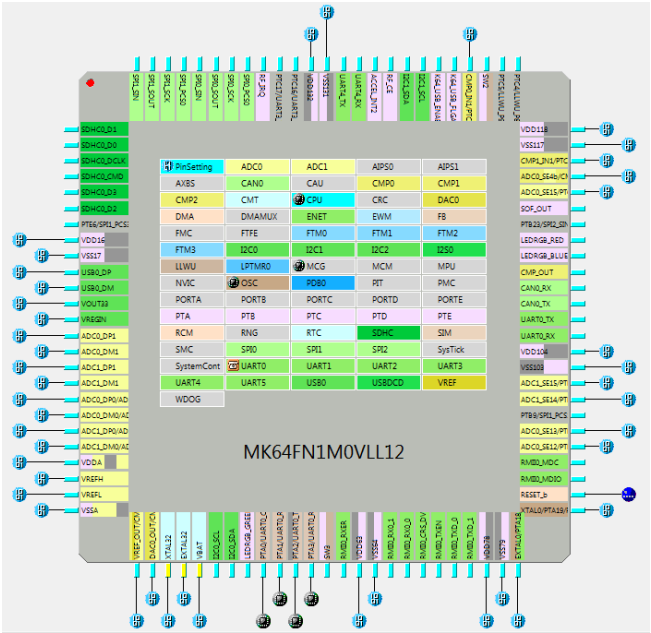
Master complex peripherals with a few mouse clicks, without the need to read thousands of data sheet pages.



Product Features

- Standalone and integrated for
  - Eclipse based IDE's
  - Freescale CodeWarrior
  - IAR Embedded Workbench
  - Keil MDK
- Easy configuration of Kinetis SDK with Processor Expert Components
- Supports Kinetis, Vybrid, S08, S12, S12Z, ColdFire, DSC and Power Architecture with reusable software components
- Initialization and driver code generation with design time consistency checking
- Bare Metal and RTOS drivers
- Middleware and Stacks: RTOS, TSS libraries and communication stacks
- Component Development Environment (CDE) to create and distribute own components

Kinetis SDK and Processor Expert



- Processor Expert is a complimentary PC-hosted software configuration tool (Eclipse plugin)
- Processor Expert (PEx) provides a time-saving option for software configuration through a graphical user interface (GUI)
- Board configuration and driver tuning capabilities include:
  - Optional generation of low-level device initialization code for post-reset configuration
  - Pin Muxing tools to generate pin muxing functions
  - Components based on Kinetis SDK drivers
    - Users configure the SoC and Peripherals in a GUI
    - PEx creates the configuration data structures for driver config and init

Hands-On: KDS, KSDK, Processor Expert, FreeMASTER

Public

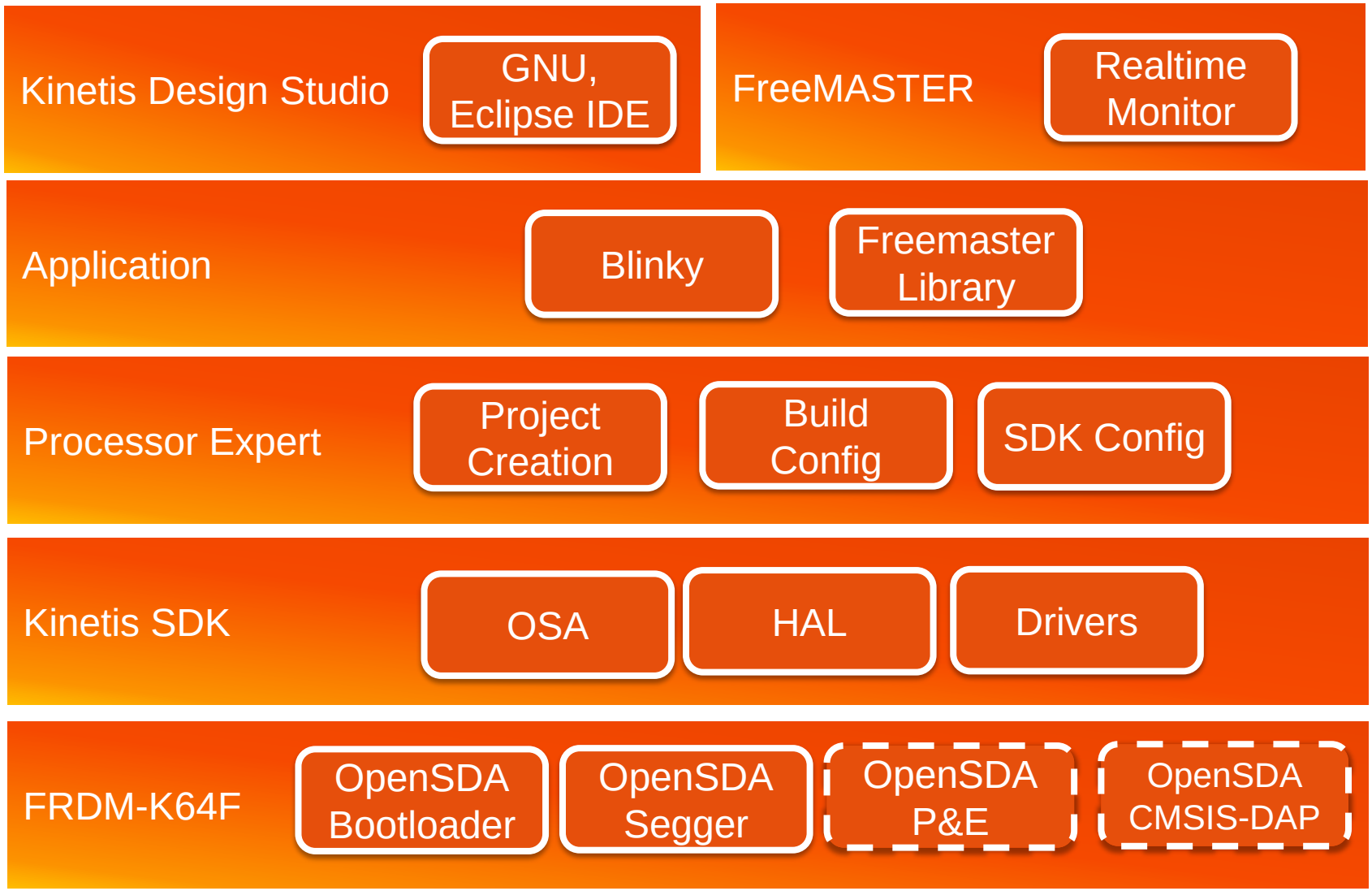
Lab Outline

- Lab Setup
  - Kinetis Design Studio v3.0.0, Kinetis SDK v1.2.0
  - Eclipse Update of SDK v1.2.0
  - FreeMaster v1.4.4 & Communication Driver V1.9
  - FRDM-K64F with Segger J-Link OpenSDA
- Lab
  - Create SDK+PEx project
  - Add Blinky LED
  - Add FreeMASTER
  - Control LED with FreeMASTER
- Appendix/Backup
  - Additional Eclipse Tips
  - KDS Register Display

Lab Stack of Tools

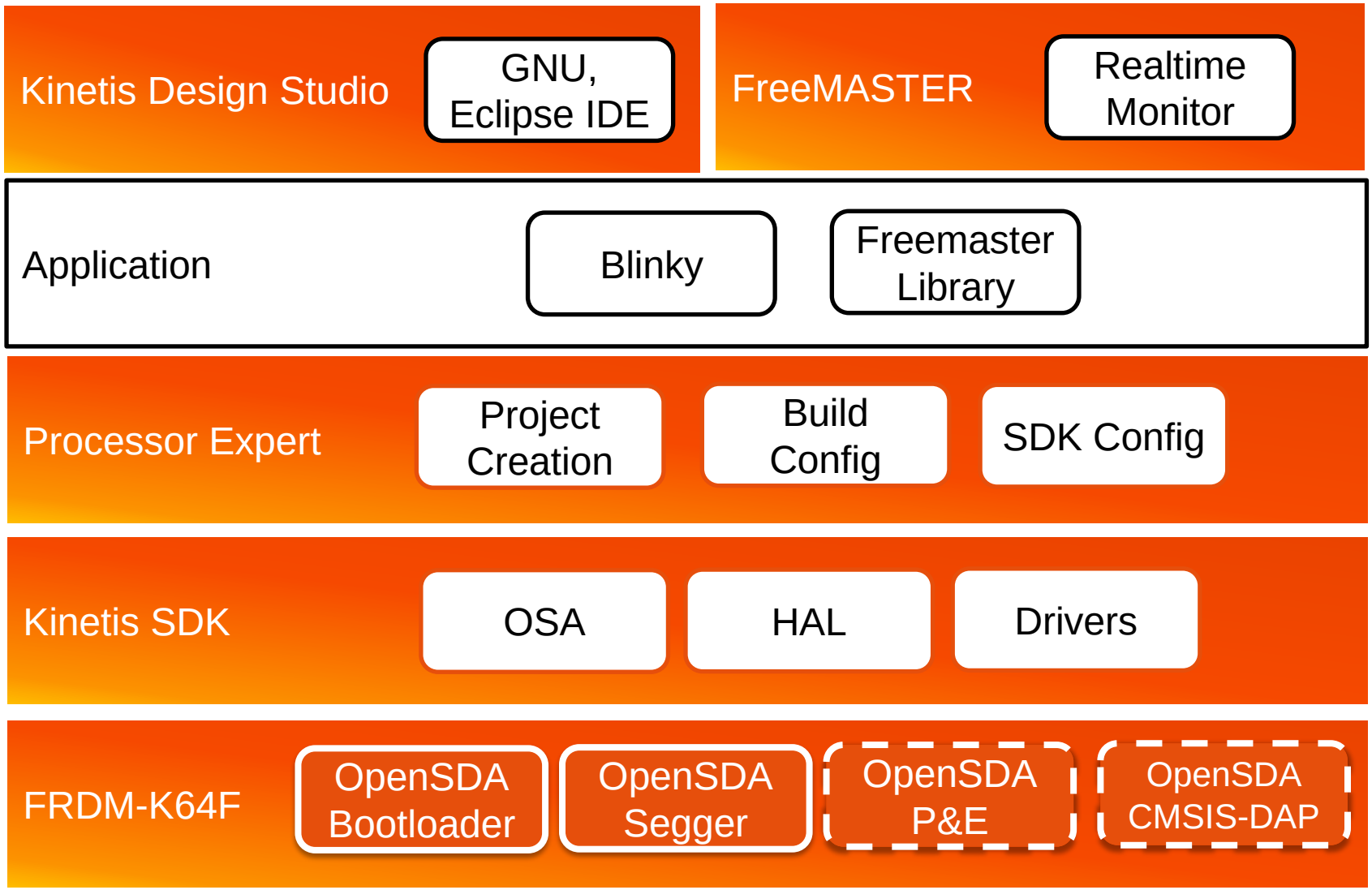
FreeMASTER	• User-friendly Real-Time debug monitor • Data Visualization Tool • Testing and prototyping
Kinetis Design Studio	• Eclipse with GNU (gcc, gdb) • Robust editing, compiling, debugging • Free of charge, no code size restriction
Processor Expert	• Peripheral Initialization and Configuration • Peripheral and Device knowledge base • Makes using complex silicon simple
Kinetis SDK	• Hardware abstraction • Comprehensive drivers • Open source license
FRDM-K64F	• Powerful board with sensors • Onboard debug interface(s) • Low cost, US$ 35 (other boards ~US$15)

Lab Session Map



Lab Setup

Lab Session Map: Lab Setup

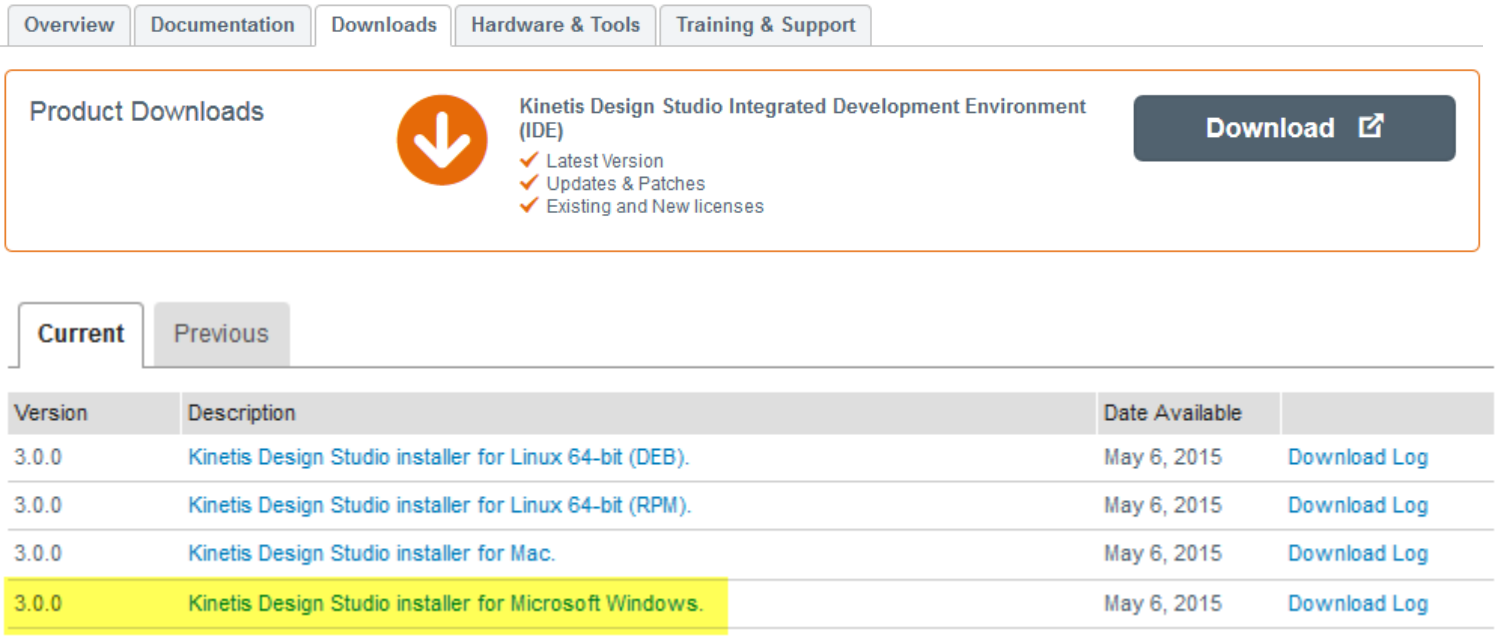


Required Tools for Lab Session

- Kinetis Design Studio V3.0.0
  - Download and install from http://www.freescale.com/kds
- Kinetis SDK V1.2.0
  - Download and install from http://www.freescale.com/ksdk
- Install SDK v1.2.0 support into KDS v3.0.0
  - Install archive in Eclipse:
  - C:\Freescale\KSDK_1.2.0\tools\eclipse_update\KSDK_1.2.0_Eclipse_Update.zip
- FreeMASTER V1.4.4, Communication Driver v1.9
  - http://www.freescale.com/freemaster
- FRDM-K64F Board with Segger OpenSDA firmware
  - https://www.segger.com/opensda.html

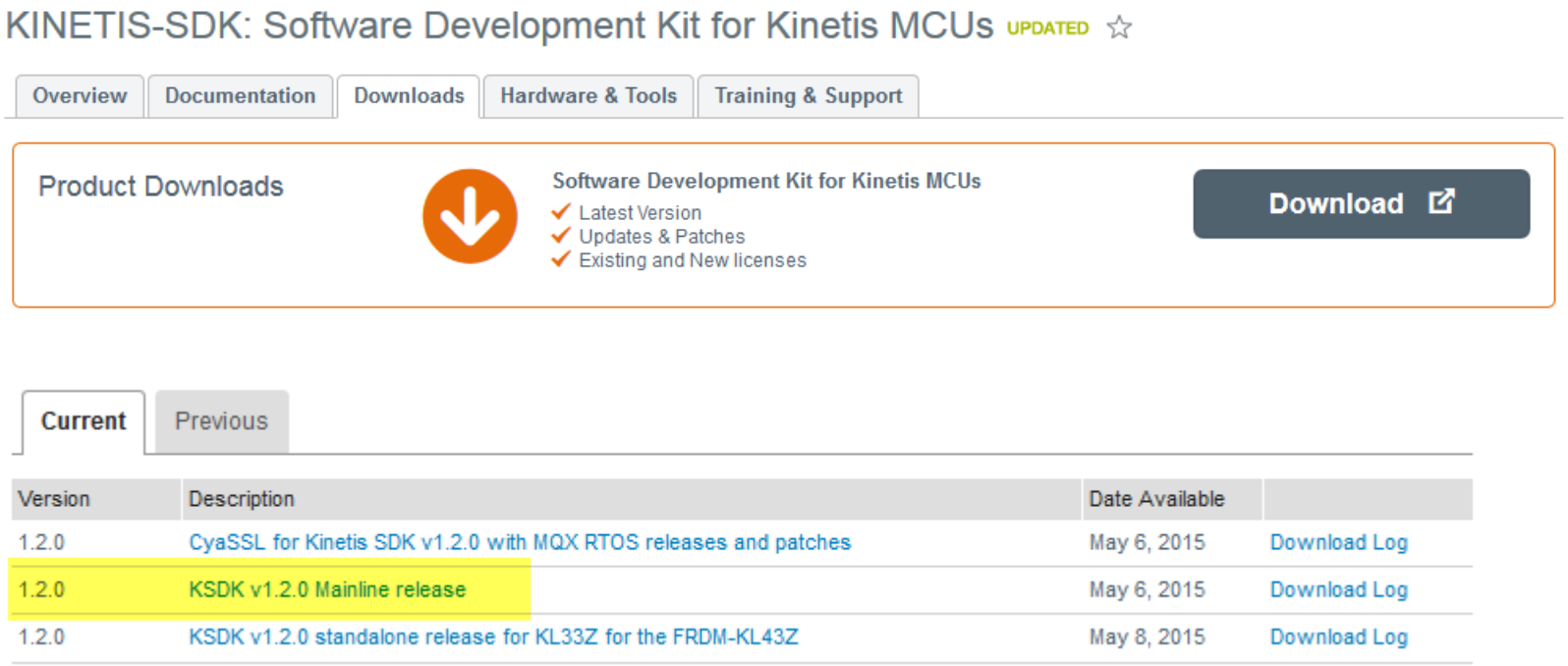
Kinetis Design Studio V3.0.0 Installation

- Download and install from http://www.freescale.com/kds
  - Kinetis Design Studio installer for Microsoft Windows (v3.0.0)
- Install with the default settings



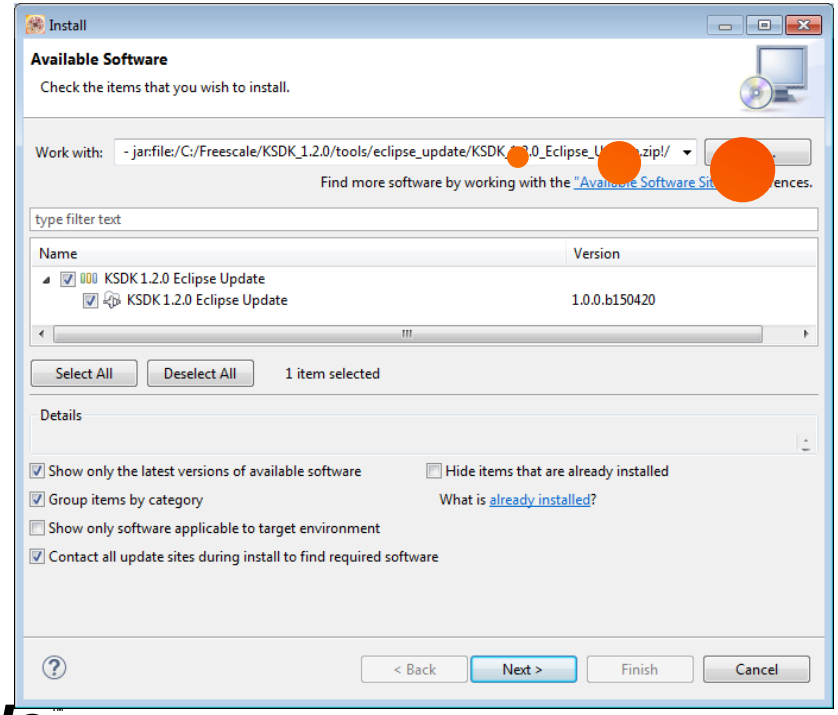
Kinetis SDK V1.2.0 Installation

- Download and Install from http://www.freescale.com/ksdk
  - KSDK v1.2.0 Mainline release (v1.2.0)
- Install with the default settings



Adding Kinetis SDK to KDS

- Launch Eclipse/Kinetis Design Studio
- Menu ***Help > Install New Software***
- Install the following update archive:
  - C:\Freescale\KSDK_1.2.0\tools\eclipse_update\KSDK_1.2.0_Eclipse_Update.zip



Tip:
Drag&Drop
the zip file into
'Work with'

FreeMaster Installation

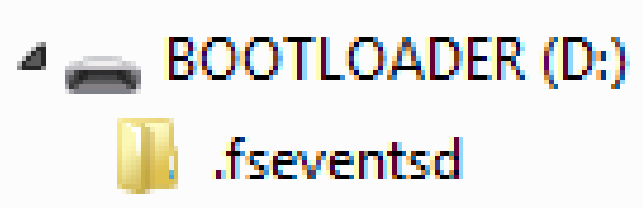
- Download and Install from http://www.freescale.com/freemaster
  - FreeMASTER Software
  - FreeMASTER Communication Driver
- Use installation defaults

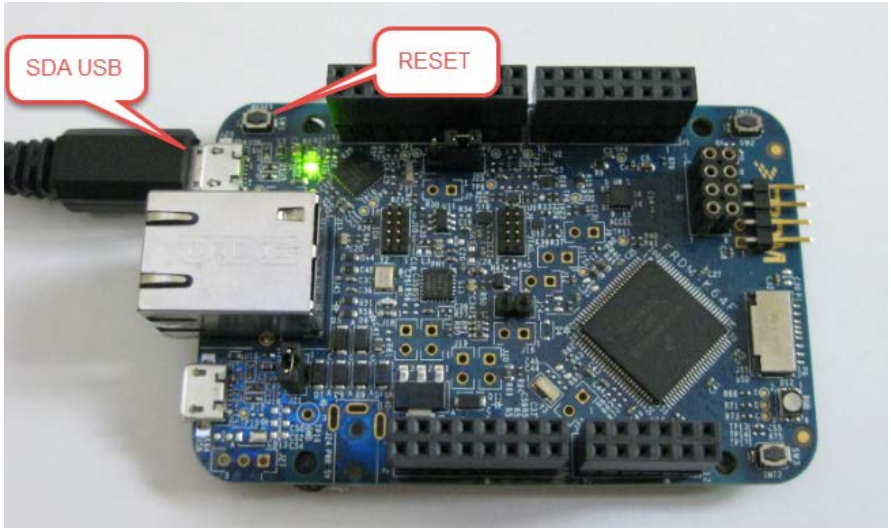
FRDM-K64F OpenSDA Firmware Files

- Firmware files
  - CMSIS-DAP (factory default, not recommended)
  - P&E, included in KDS layout
  - **Segger, download from web (recommended)**
- Use **V2.0** files for FRDM-K64F (V2.1 are for K22F!)
- P&E Firmware file
  - C:\Freescale\KDS_3.0.0\pemicro\opensda
  - **DEBUG-FRDM-K64F_Pemicro_v108a_for_OpenSDA_v2.0.bin**
- Segger Firmware file (recommended)
  - https://segger.com/opensda.html, scroll to bottom of page
  - OpenSDA V2 (E.g. FRDM-K64F boards): Download JLink_OpenSDA_V2_2015-04-23.zip
  - **Unpack** zip file: **JLink_OpenSDA_V2_2015-04-23.bin**

Updating FRDM-K64F OpenSDA Firmware

- Power Board with 'SDA USB' while RESET button pressed
  - Green LED near connector blinks slowly
  - Board enumerates as BOOTLOADER
  - Copy firmware file (**JLink_OpenSDA_V2_2015-04-23.bin**) to BOOTLOADER drive
  - Green LED blinks fast
- Repower board with SDA USB port normally
  - USB driver might enumerate
  - Green LED is on

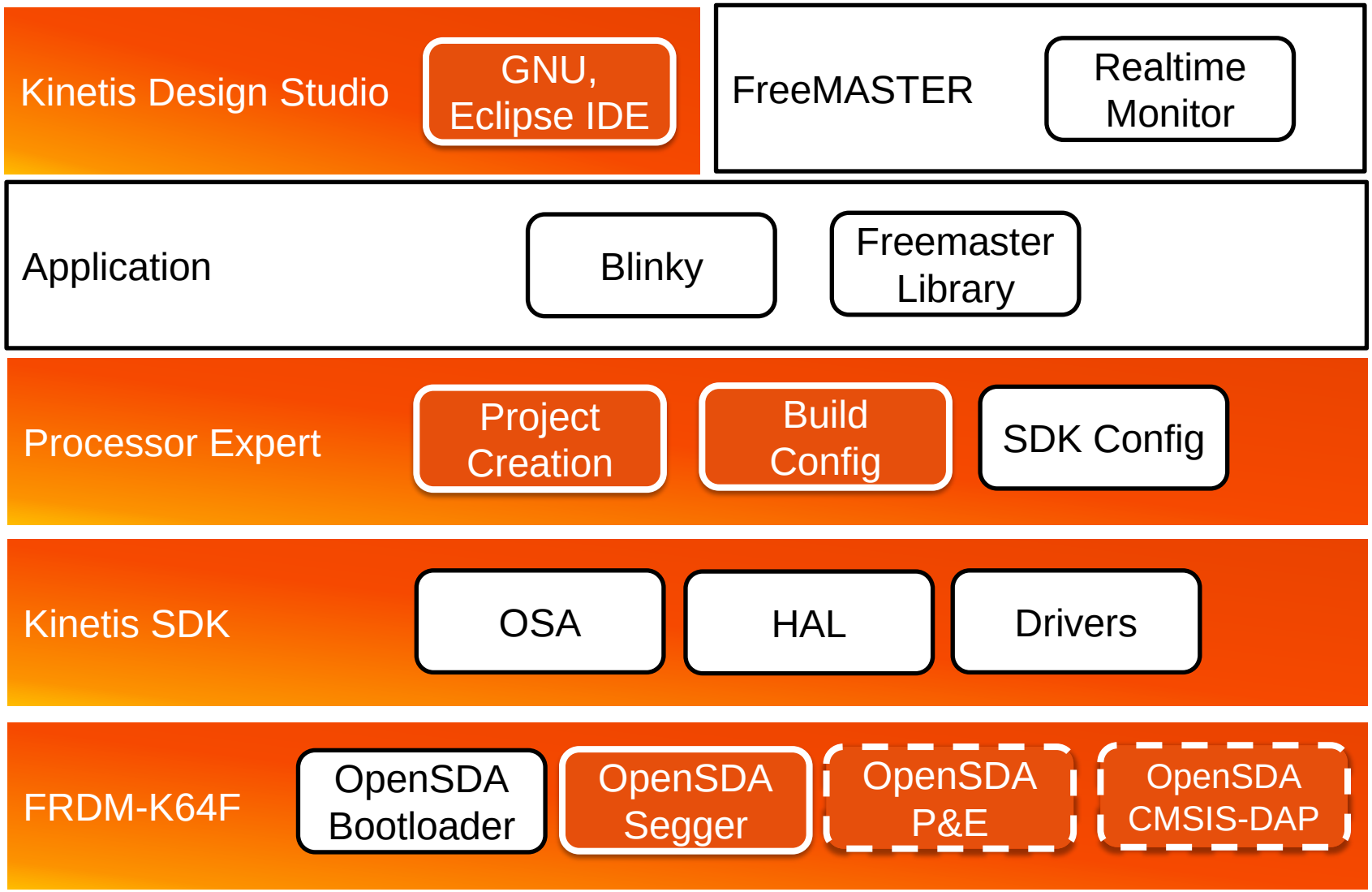




Lab: Kinetis Design Studio: Create, Build, Debug

Public

Lab Session Map: KDS Create, Build, Debug



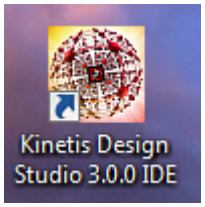
Outline

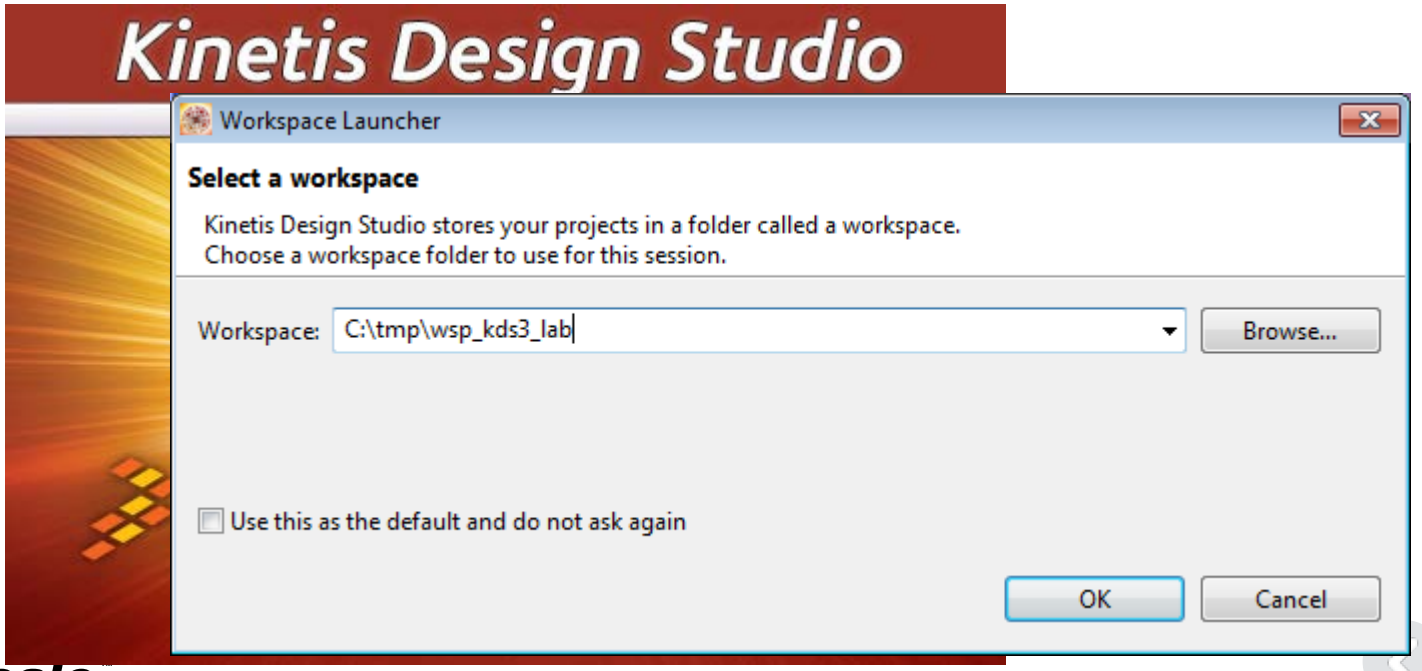
- Goal: Using KDS to create, build and debug

- Steps
  - Launching Kinetis Design Studio
  - Select Workspace, goto workbench
  - Create new Kinetis project
    - Kinetis SDK
    - Processor Expert
  - Compile/Build
  - Debug

Launch KDS: Workspace

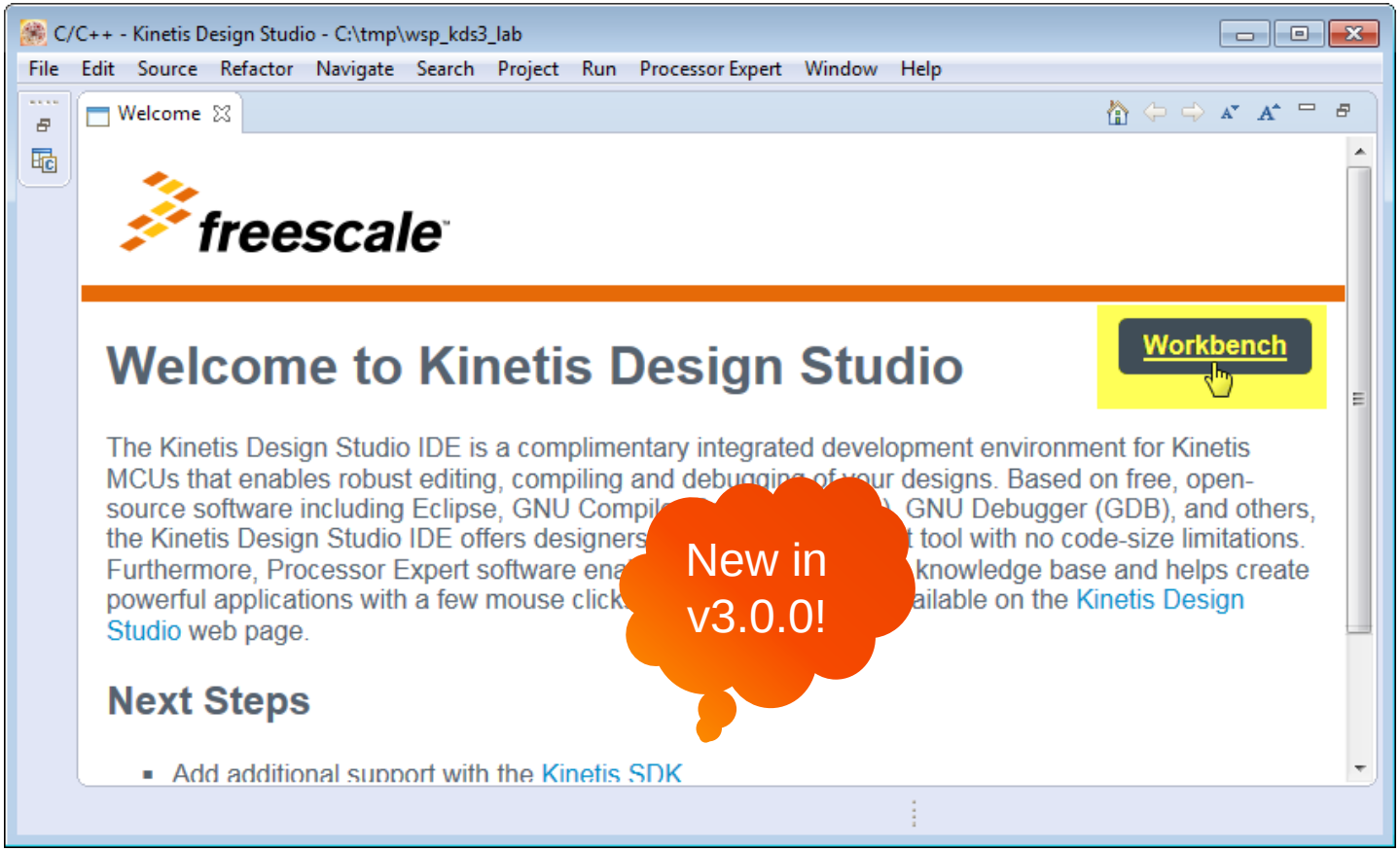
- Start from the shortcut on Desktop
  - C:\Freescale\KDS_3.0.0\eclipse\kinetis-design-studio.exe
- Choose (new) workspace folder
  - E.g. in your user account
  - Tip: workspace contains settings (usually projects too). Do *not* mix workspaces between IDE's!



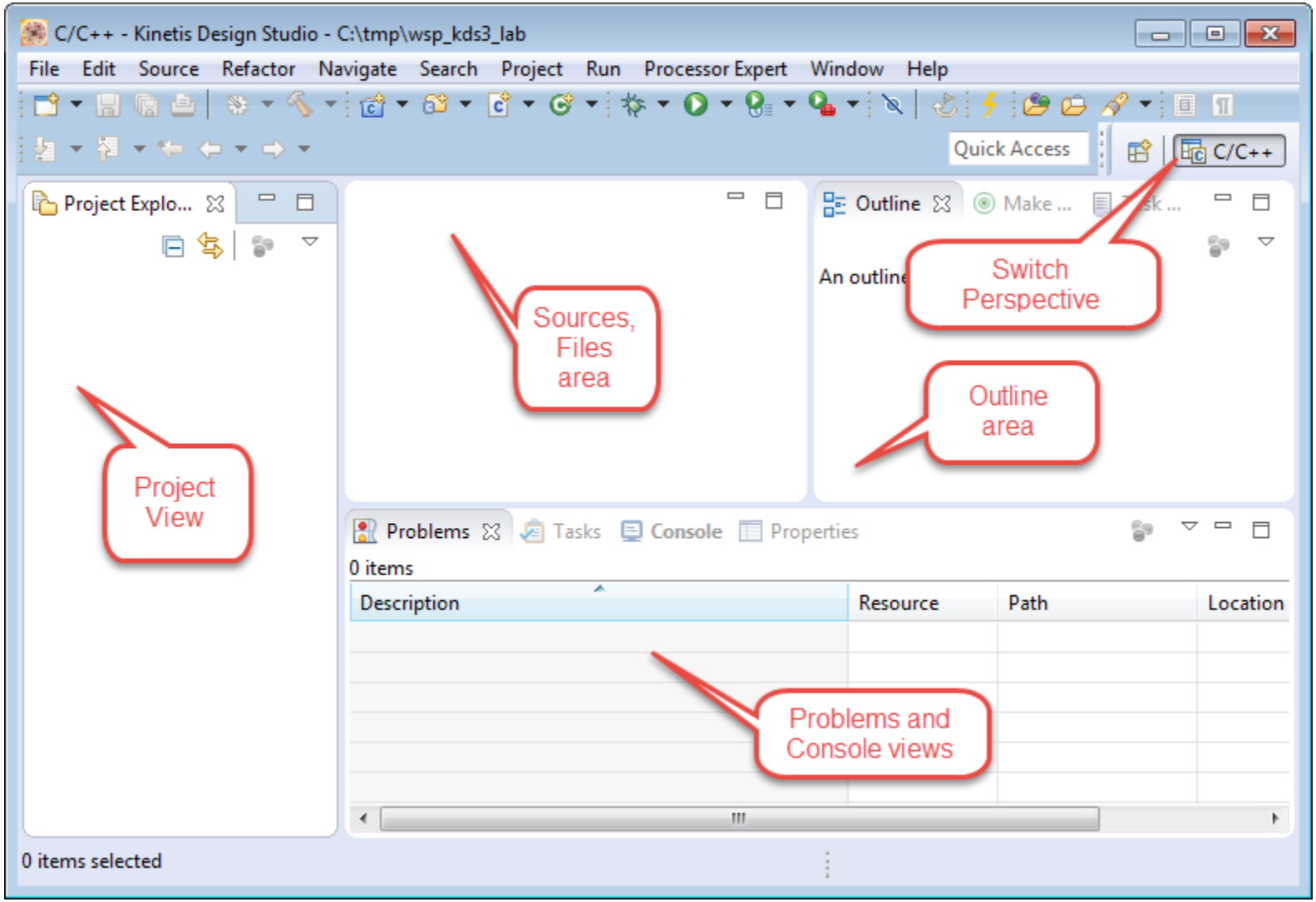


Welcome

- Welcome view: links to web and further information
- Click on **Workbench** to switch to C/C++ Perspective

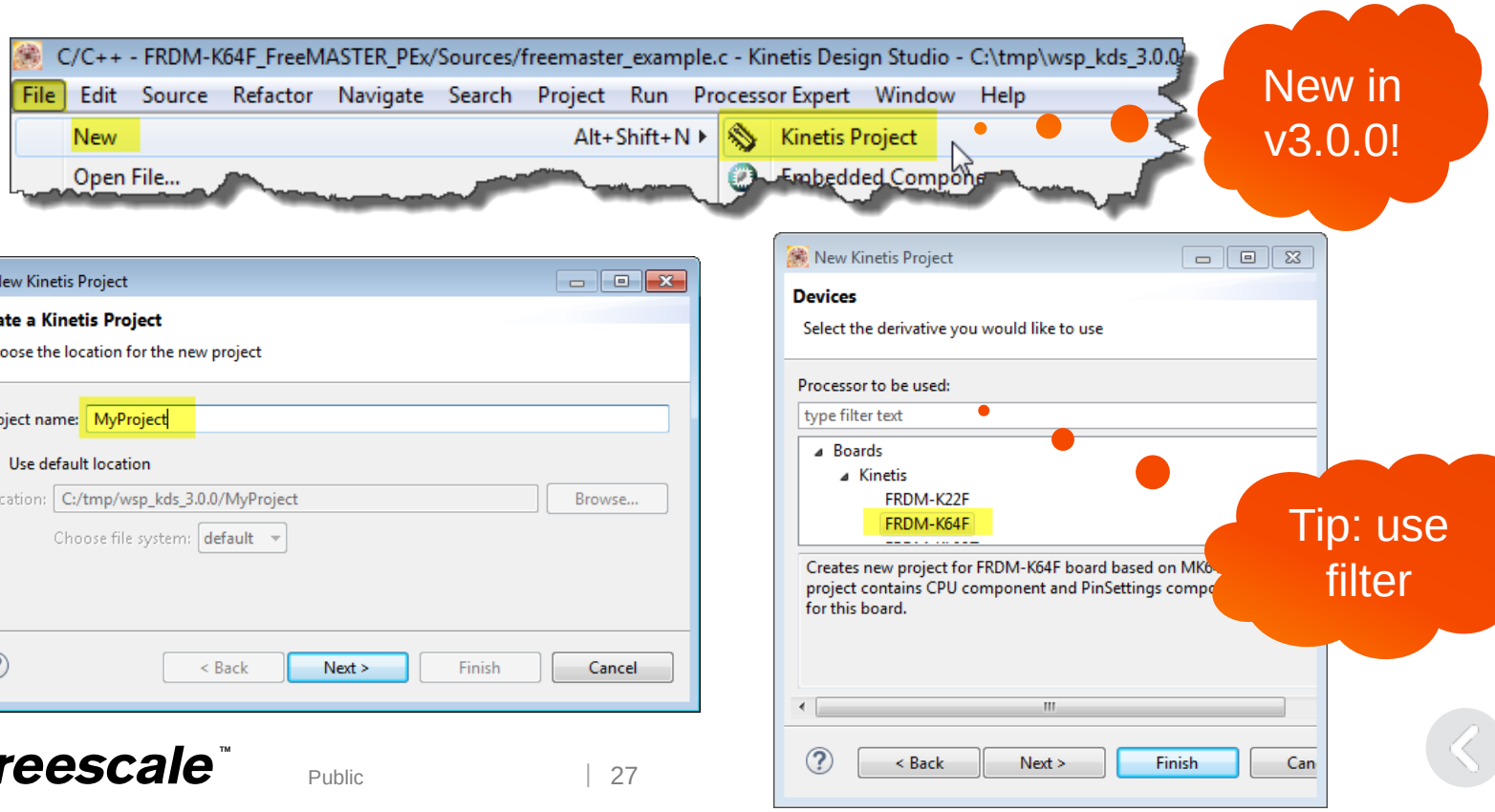


Workbench View



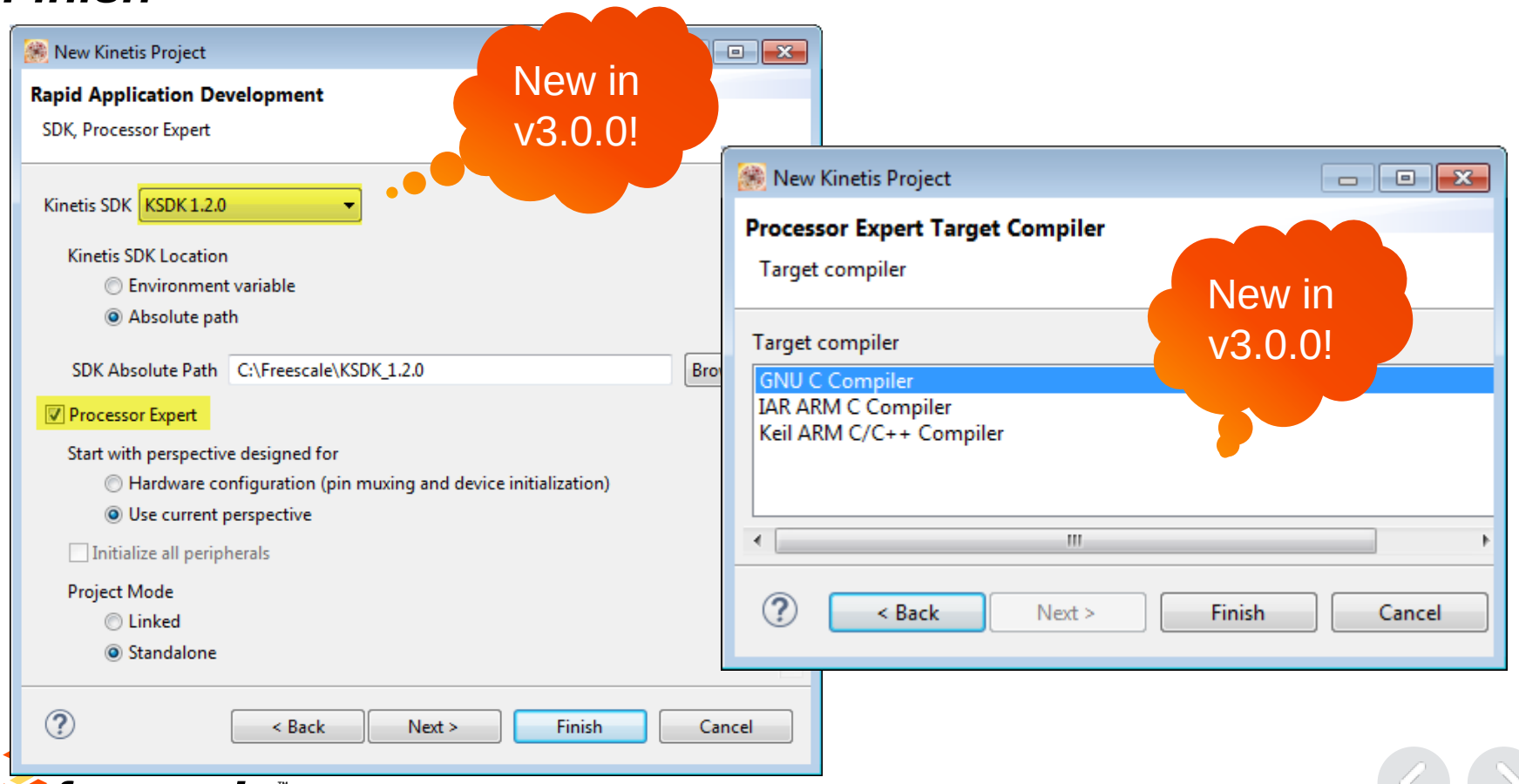
Creating Project

- Menu ***File > New > Kinetis Project***
- Next, specify a ***Project name***
- Next, select Board (***Boards > Kinetis > FRDM-K64F***)

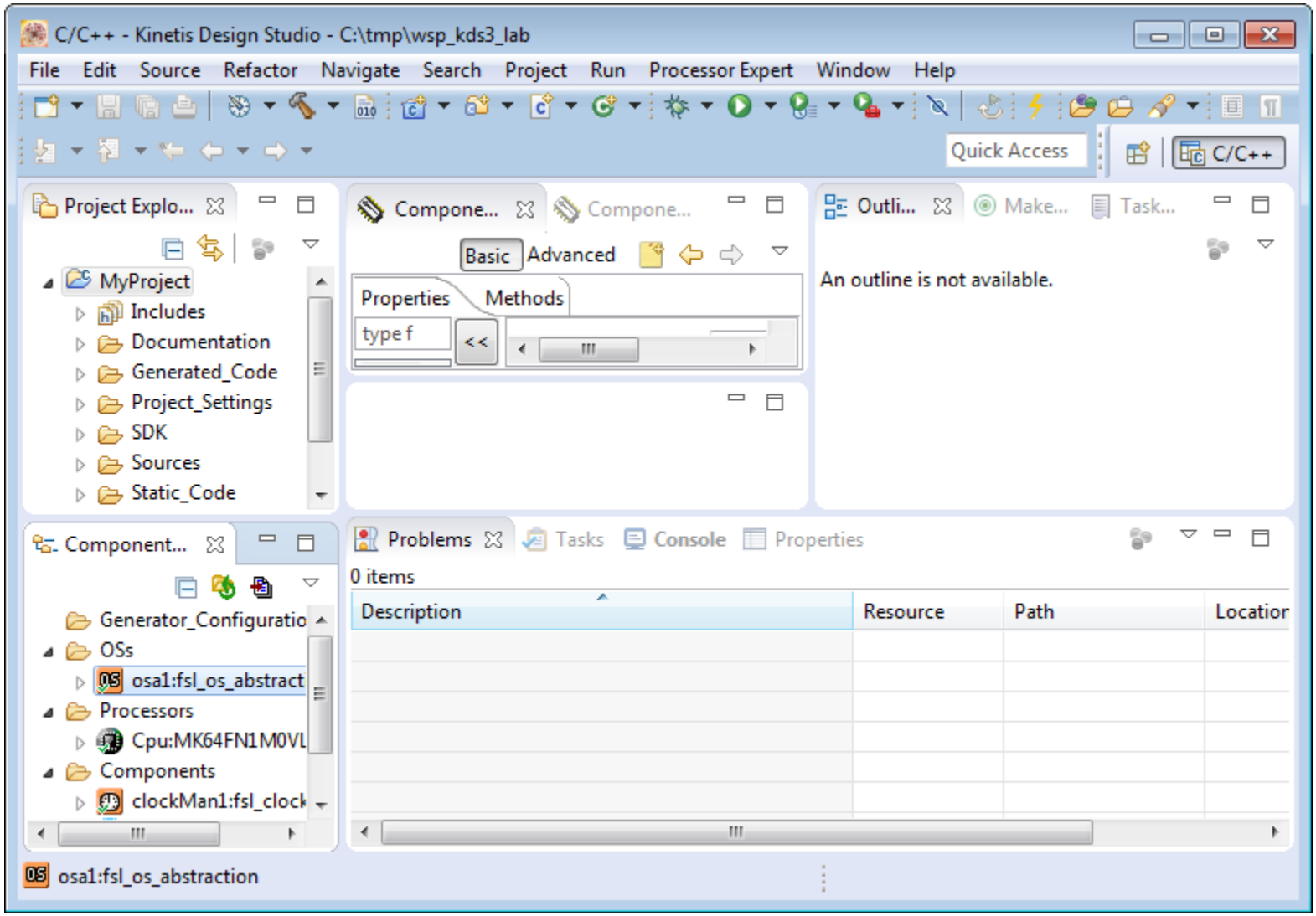


Project Creation

- Next, Select ***Kinetis SDK 1.2.0*** and Enable ***Processor Expert***
- Next, Use ***GNU C Compiler***
- ***Finish***

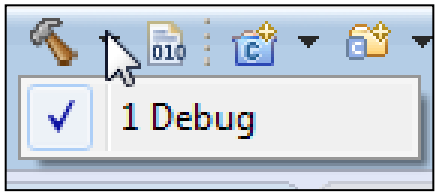


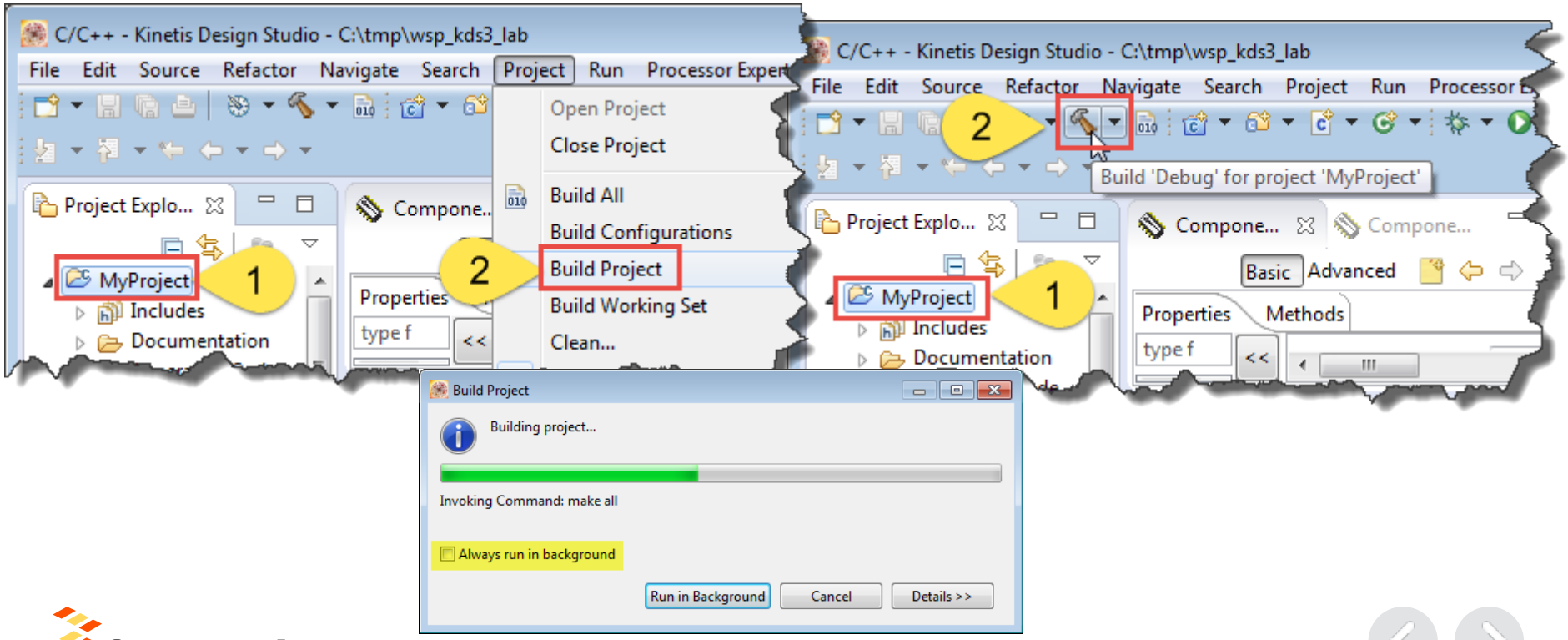
Project Created



Building Project

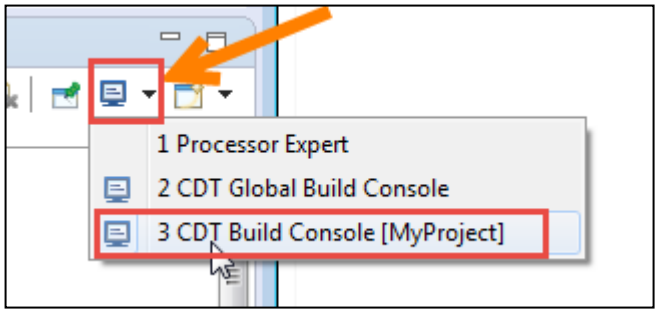
- Have Project **root folder selected**
- menu ***Project > Build Project***
  - Or: use **'Hammer'** icon
- Tip: **Run in Background**

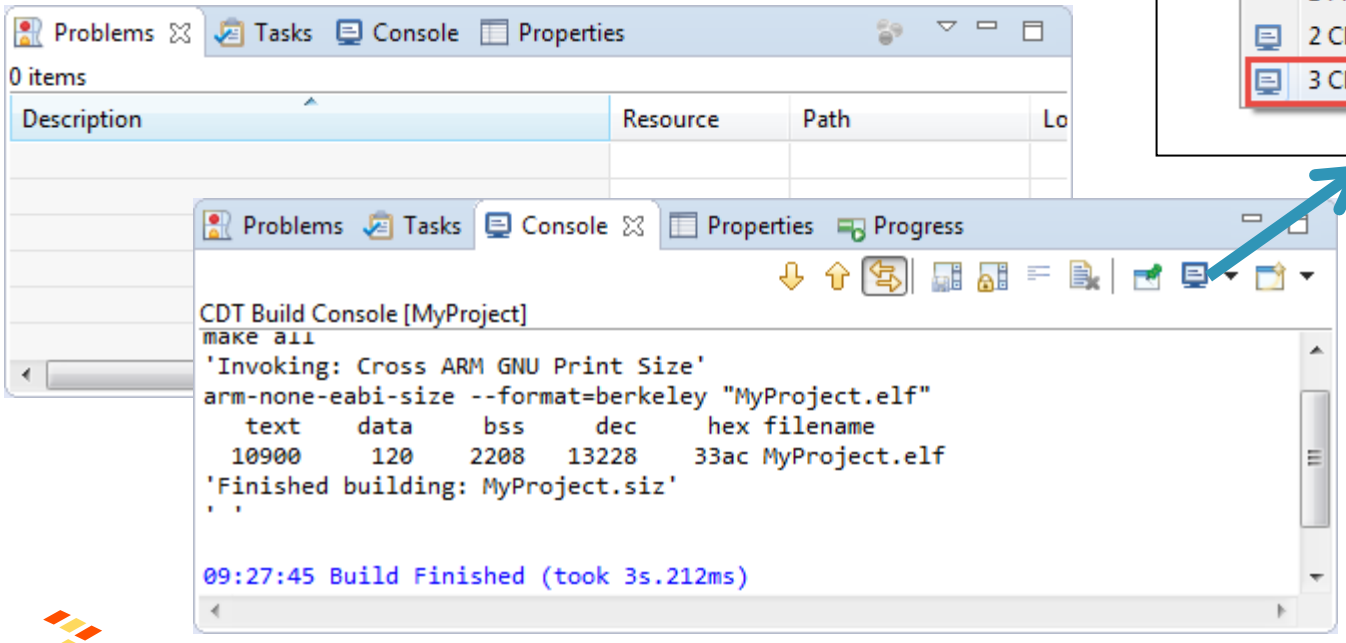




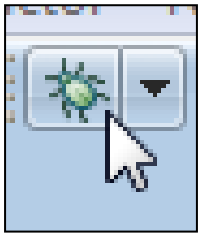
Building

- Warnings/Errors reported in **Problems** view
  - Should not report any errors (hopefully ☺)
  - Eclipse parses tool output from console
- Tool Output is in **Console** view
  - Switch Console with 'triangle' menu

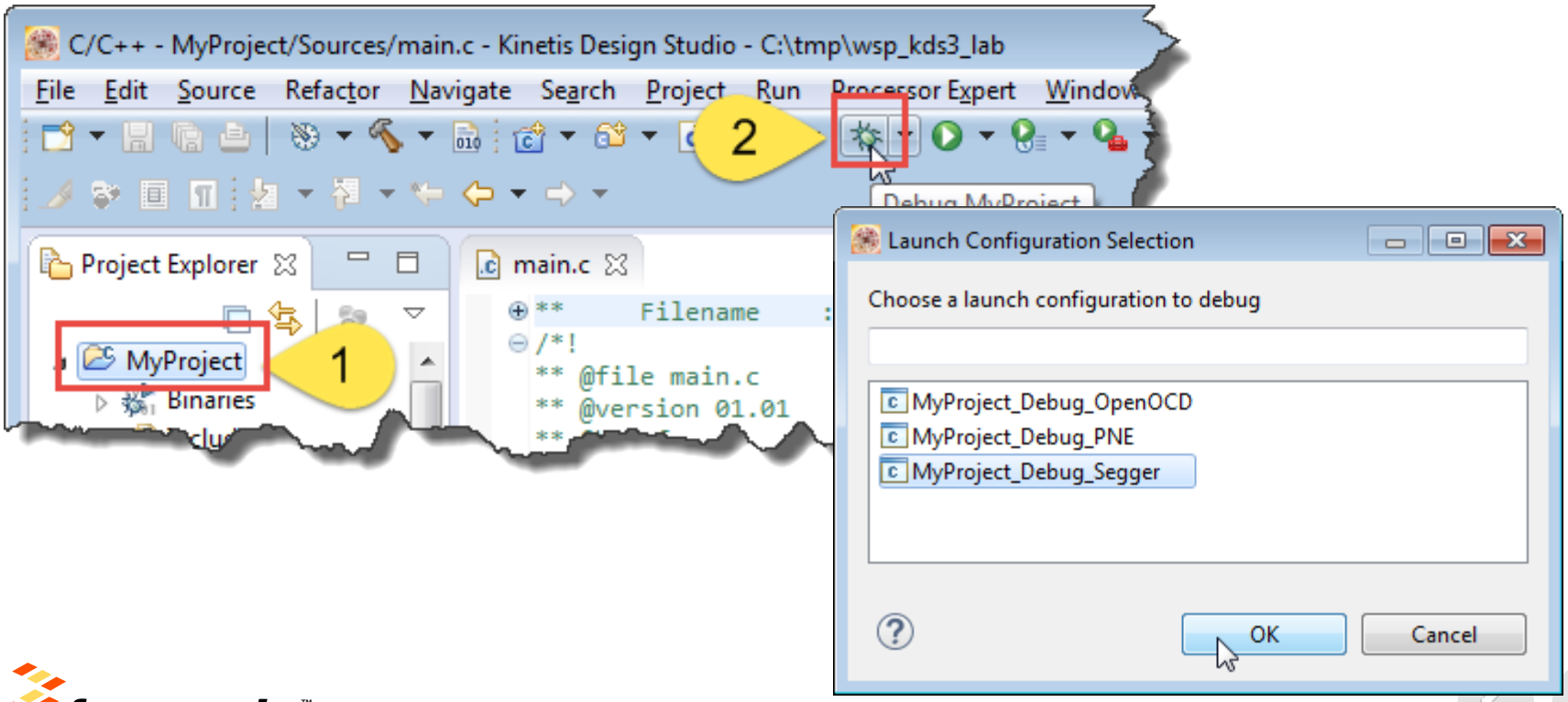




Debug

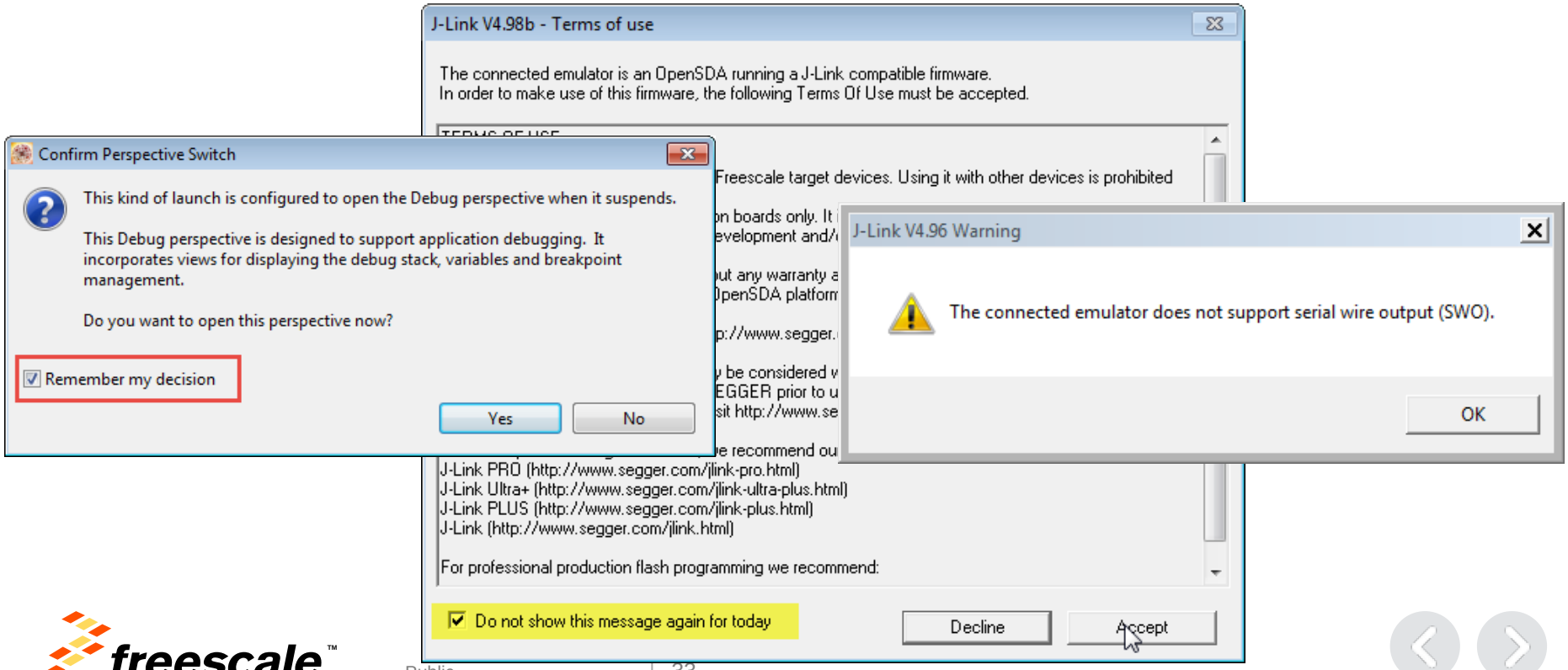


- Have **project selected**
- Use '**Debug**' icon
- If no recent launch: dialog to select configuration

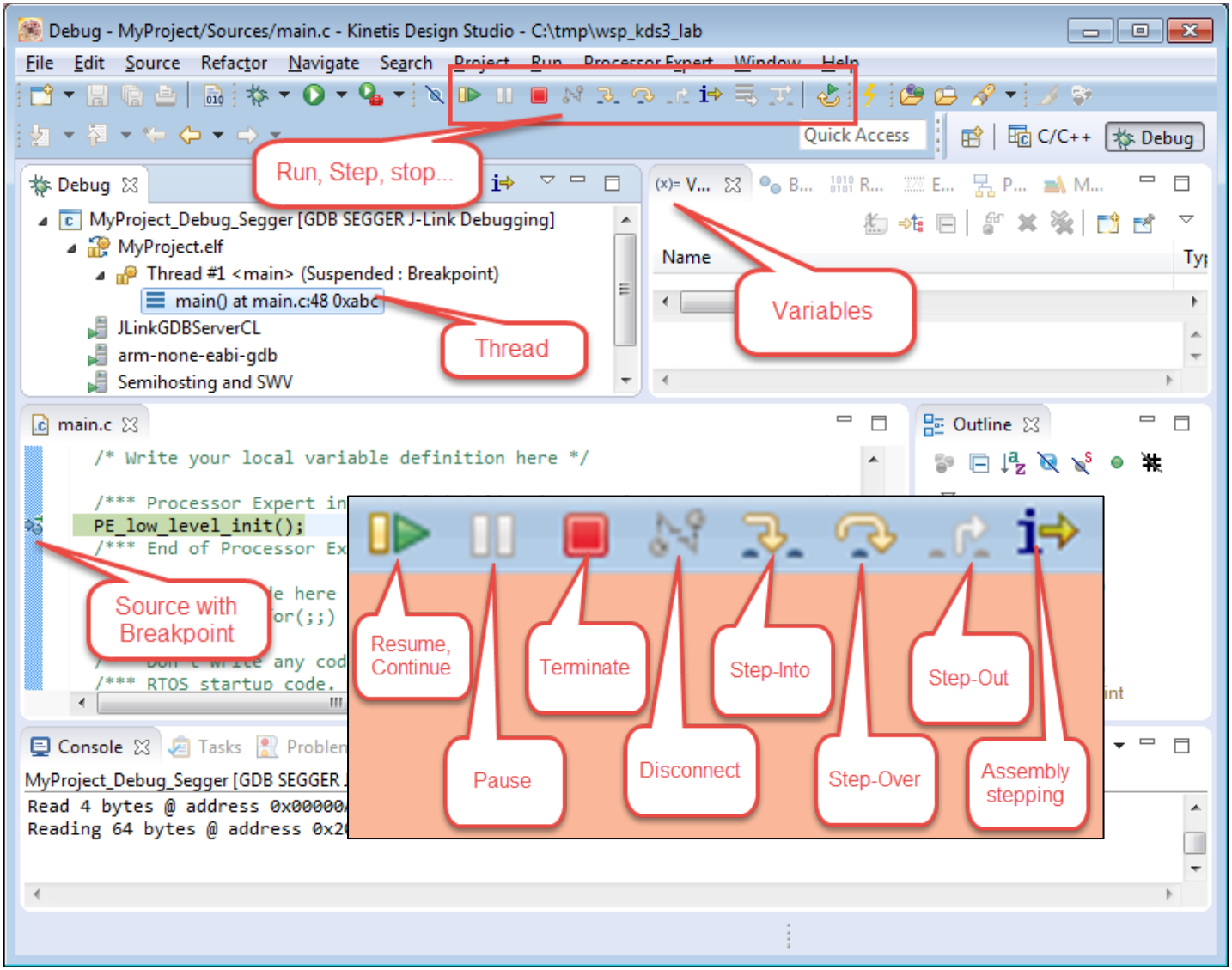


Debug Perspective

- If asked to switch to Debug perspective
  - Enable 'Remember my decision'
- If asked to agree to Segger 'Terms of use': **Accept**
- If asked about not supporting SWO: **OK** (will fix that one later)

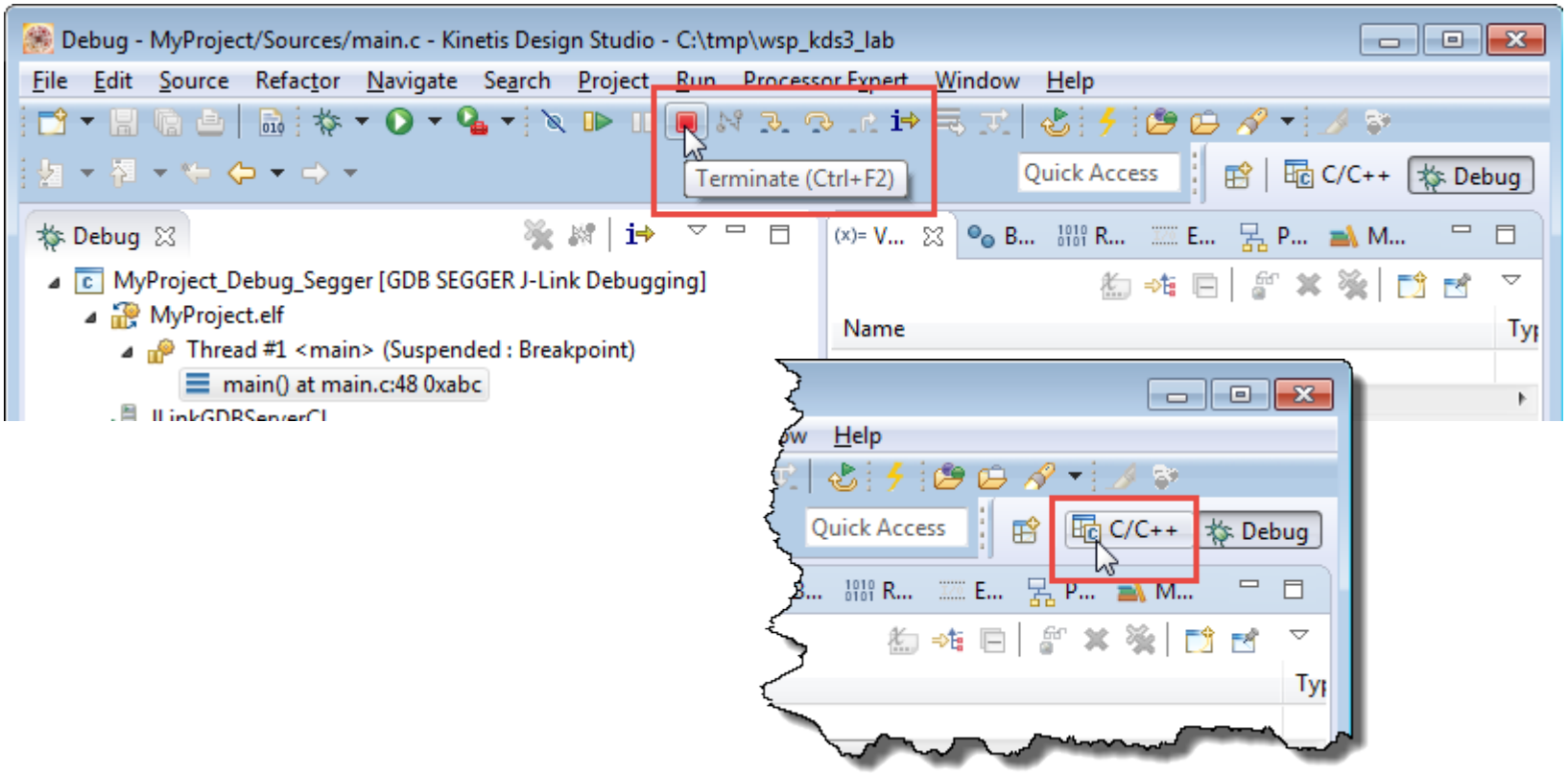


Debug Perspective



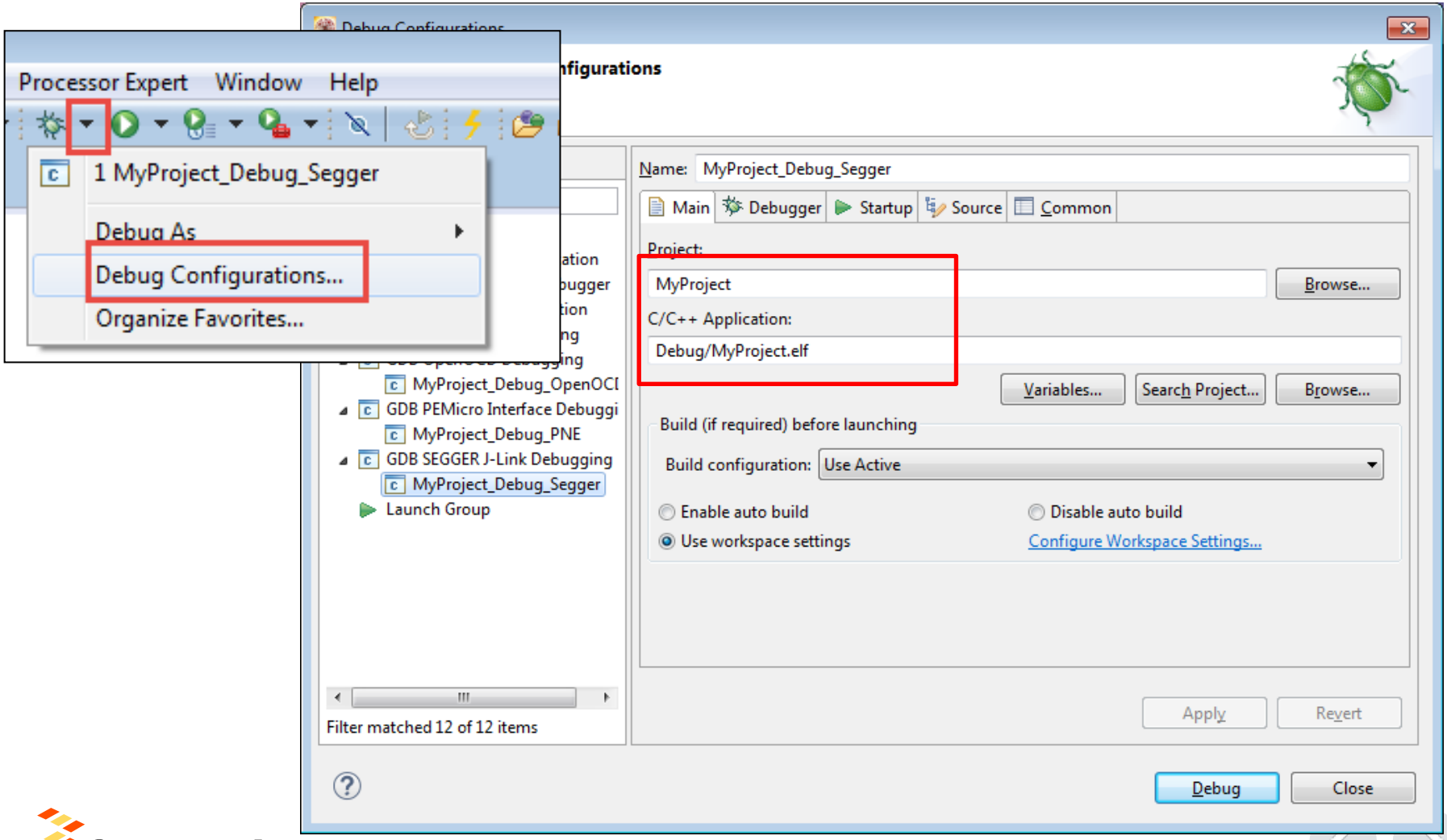
Terminating Debug Session

- Terminate with 'Stop' Icon
  - Do NOT start another debug session if one is already running!
- Switch back to C/C++ perspective



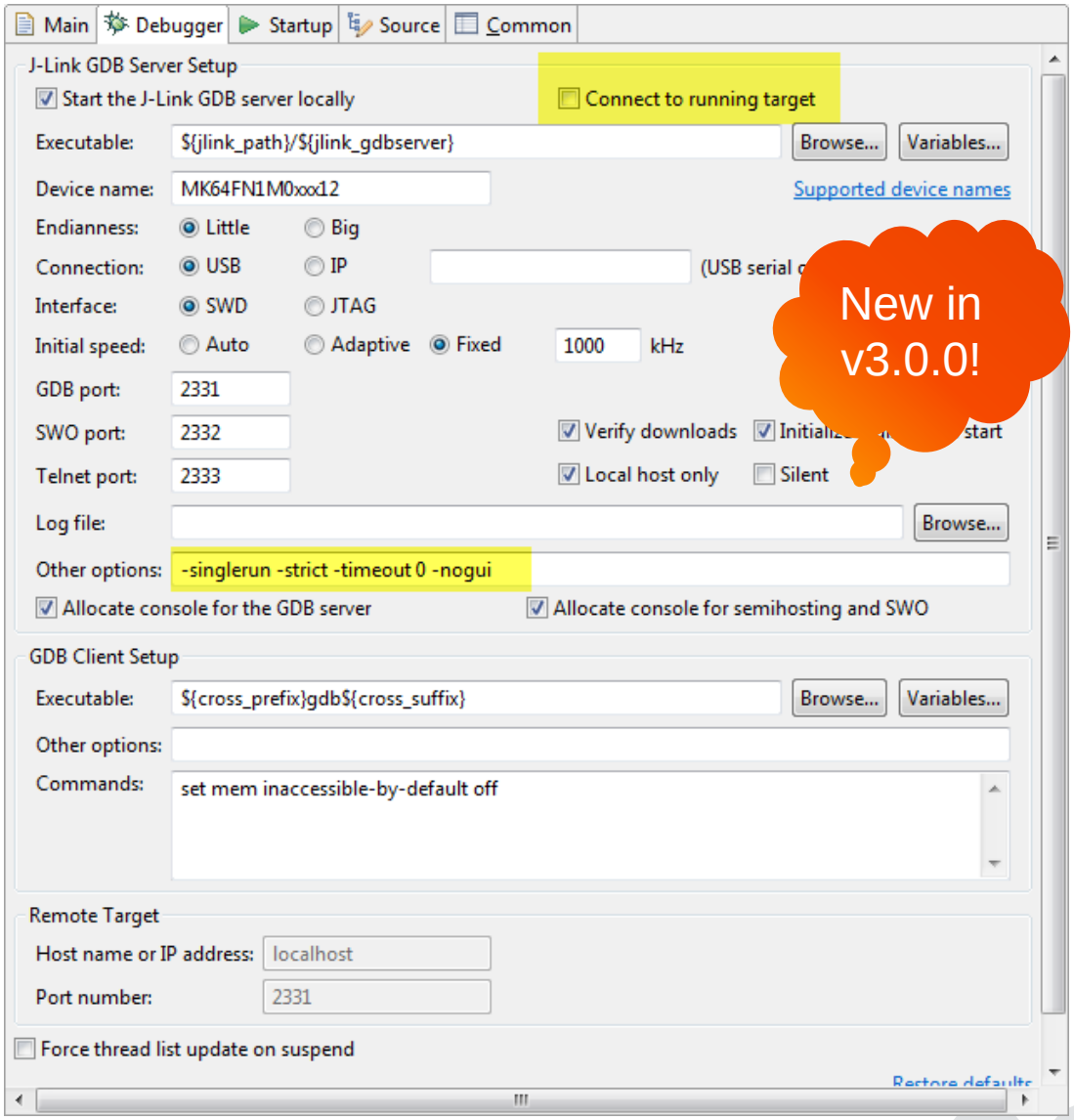
Editing Debug Configuration

- Toolbar menu, or Menu ***Run > Debug Configurations…***



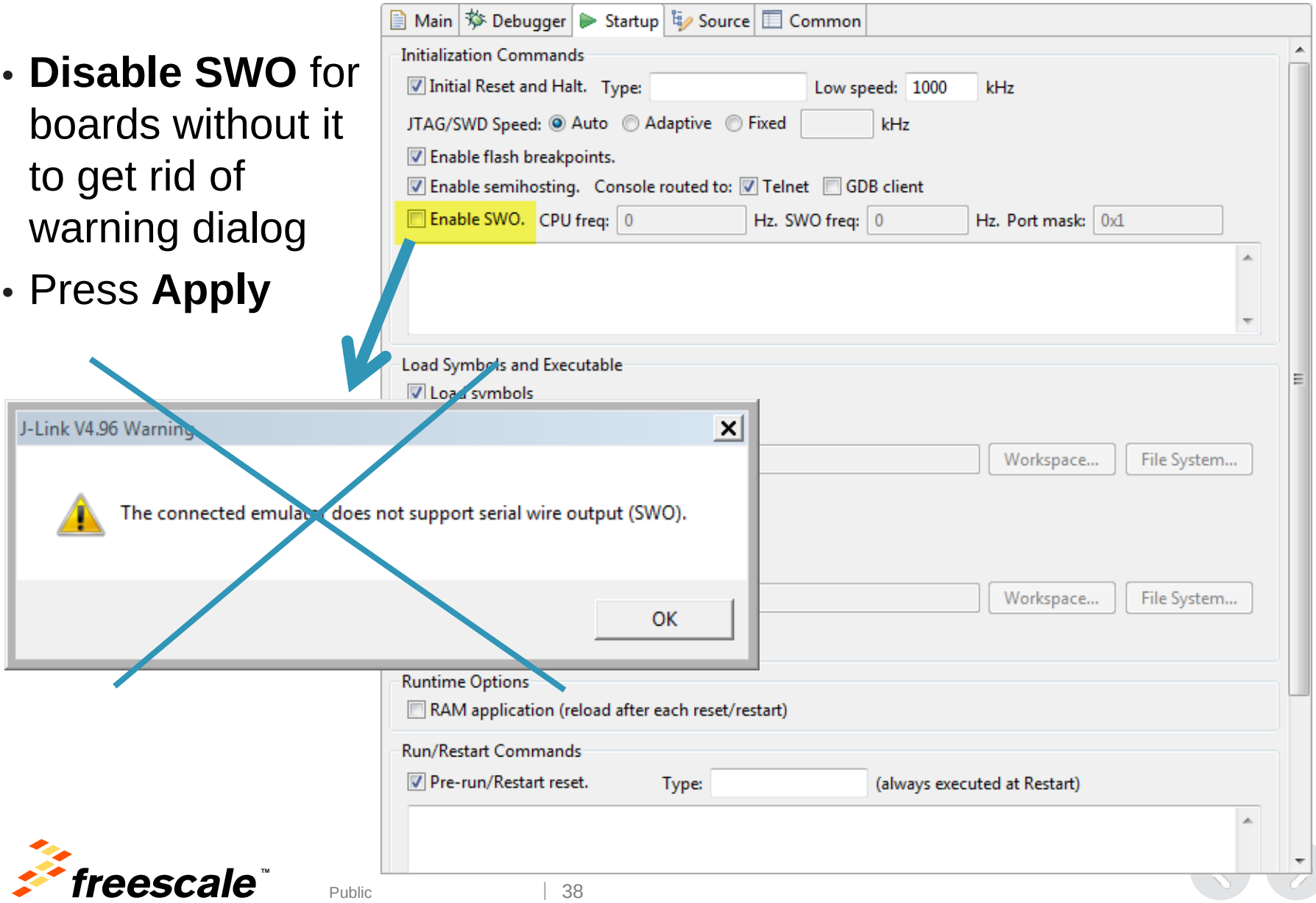
Segger Debug Configuration

- Connect to running target
- Detect already running debug session
- Timeout configuration
- Disable GUI (e.g. during Segger firmware update)



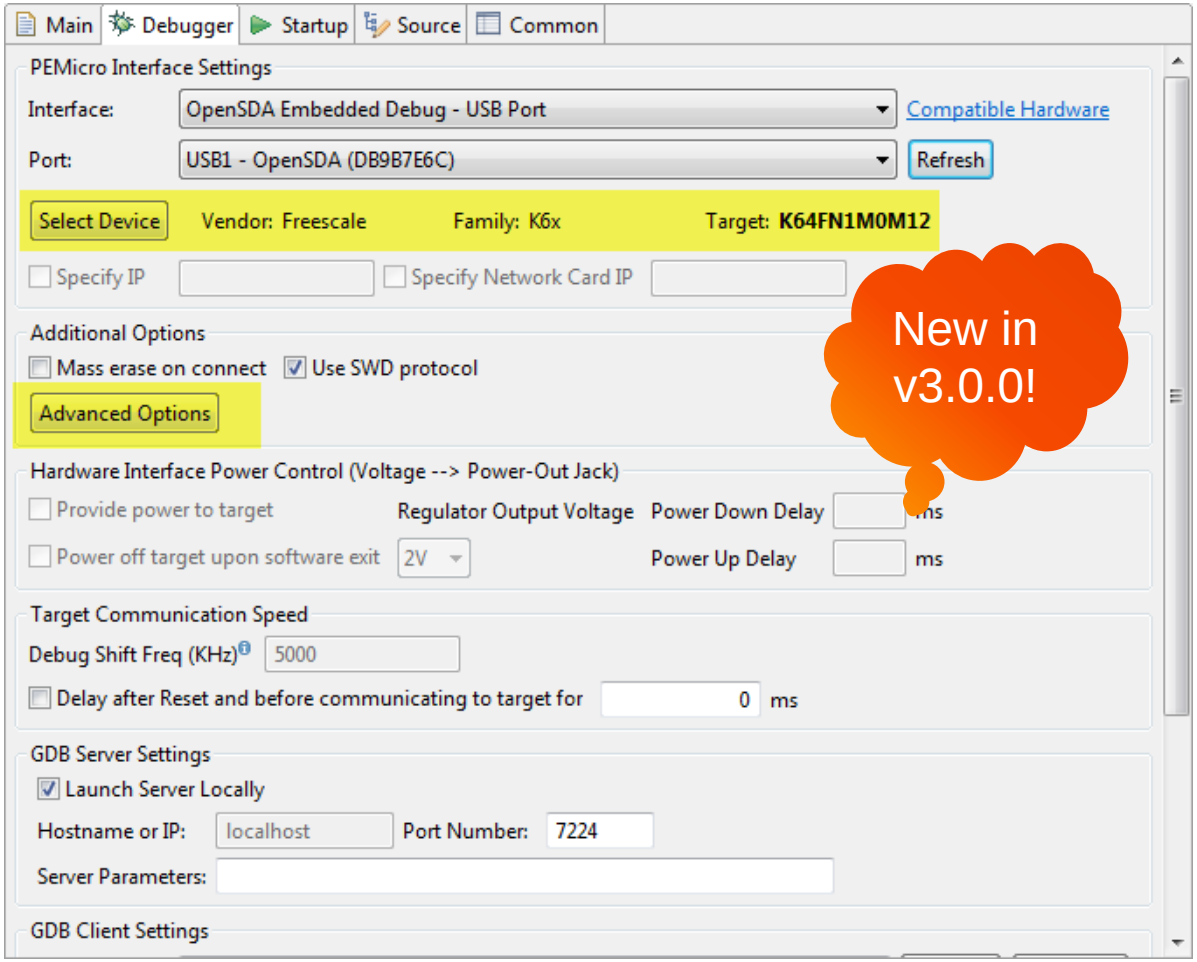
Segger Debug Configuration: SWO

- **Disable SWO** for boards without it to get rid of warning dialog
- Press **Apply**



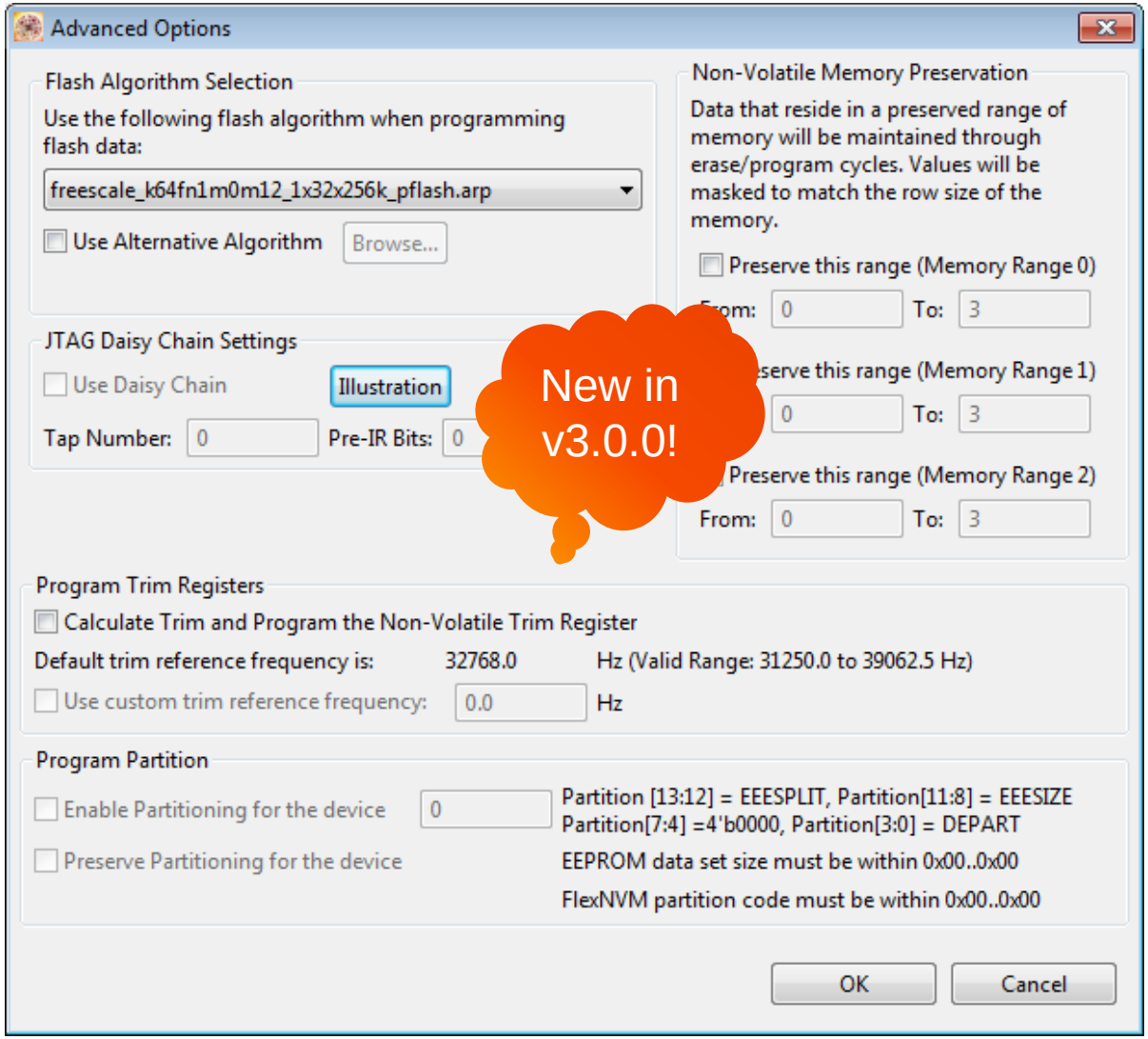
P&E Debug Configuration

- Easier device selection in v3.0.0
- **Advanced flash programming option**



P&E Advanced Programming Options

- Flash Algo Selection
- JTAG Daisy Chaining
- Memory Preservation
- Trim Register programming
- Program partition

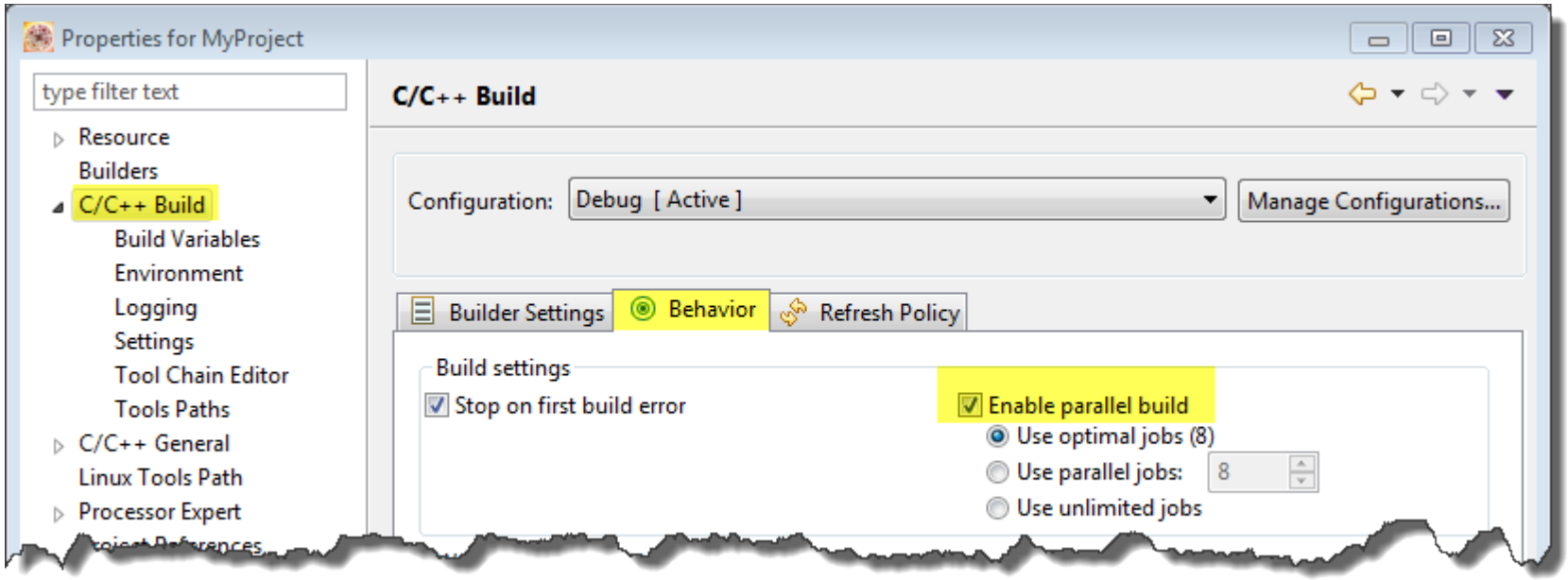


Few Eclipse Tips ☺

(more in the backup slide section)

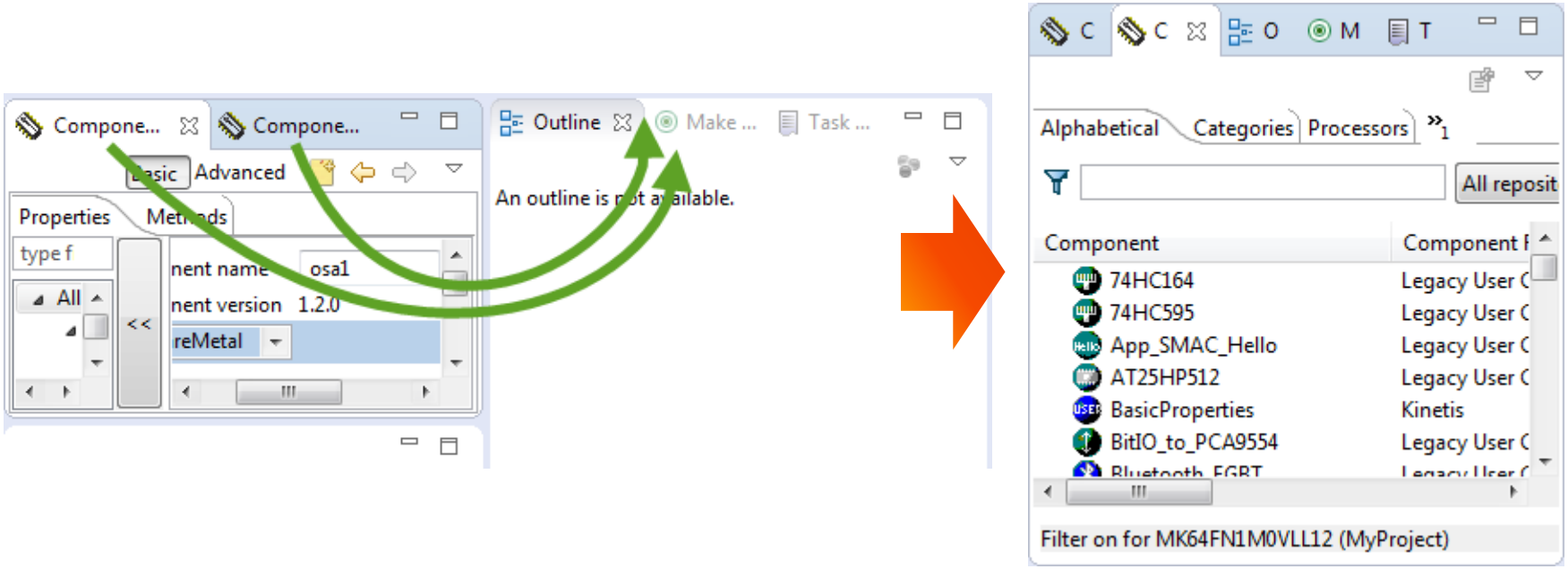
Enable Parallel Build

- Reduces Build time by factor 2-4 (depends on number of cores)
- Project > Properties > C/C++ Build
- Behavior tab
- Enable parallel build



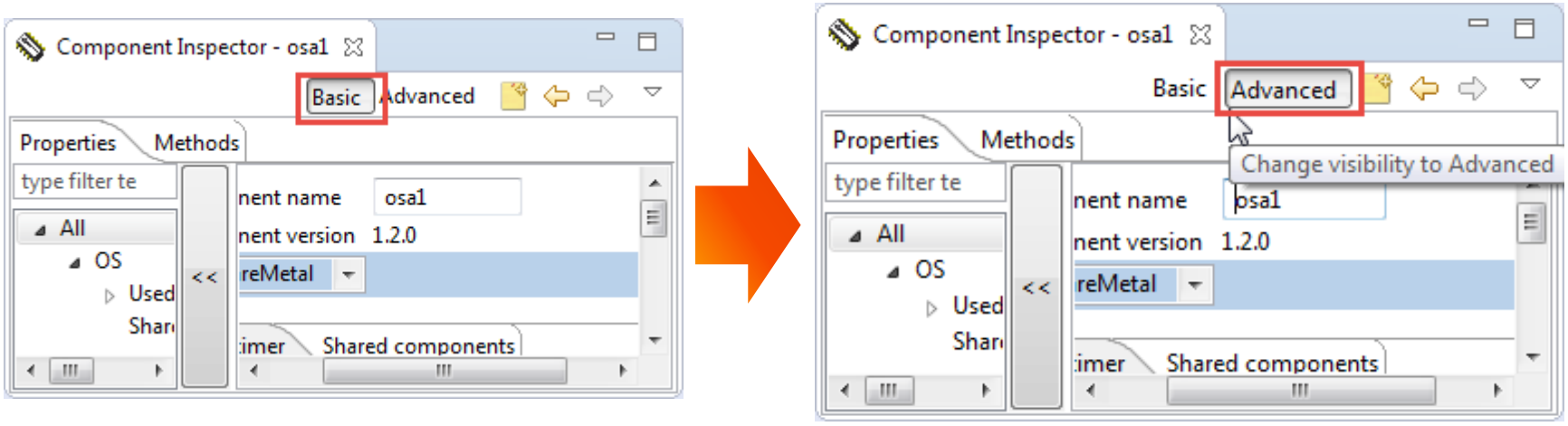
Tip: Processor Expert View Location

- Make room for other views (sources, …)
- Move Processor Expert Views to 'Outline' area
- Drag 'Tabs' to new place



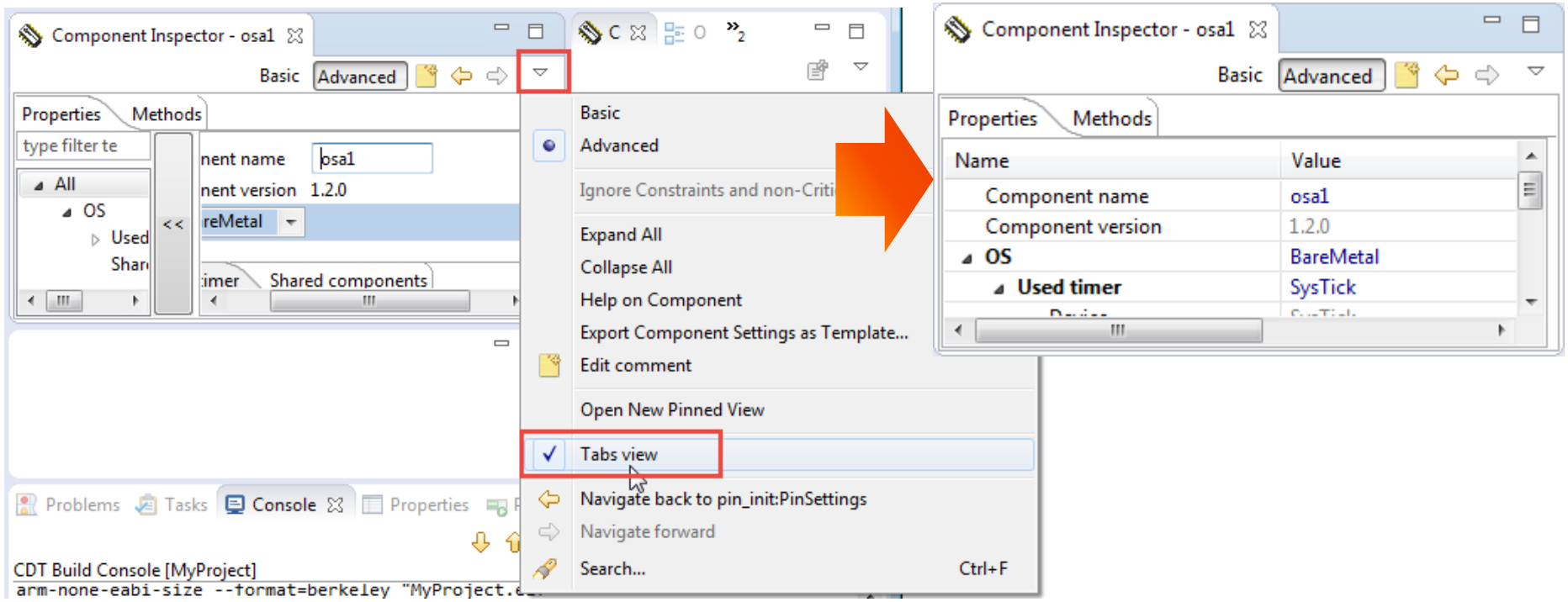
Tip: Processor Expert Advanced Mode

- Switch to '**Advanced**' mode to see all properties



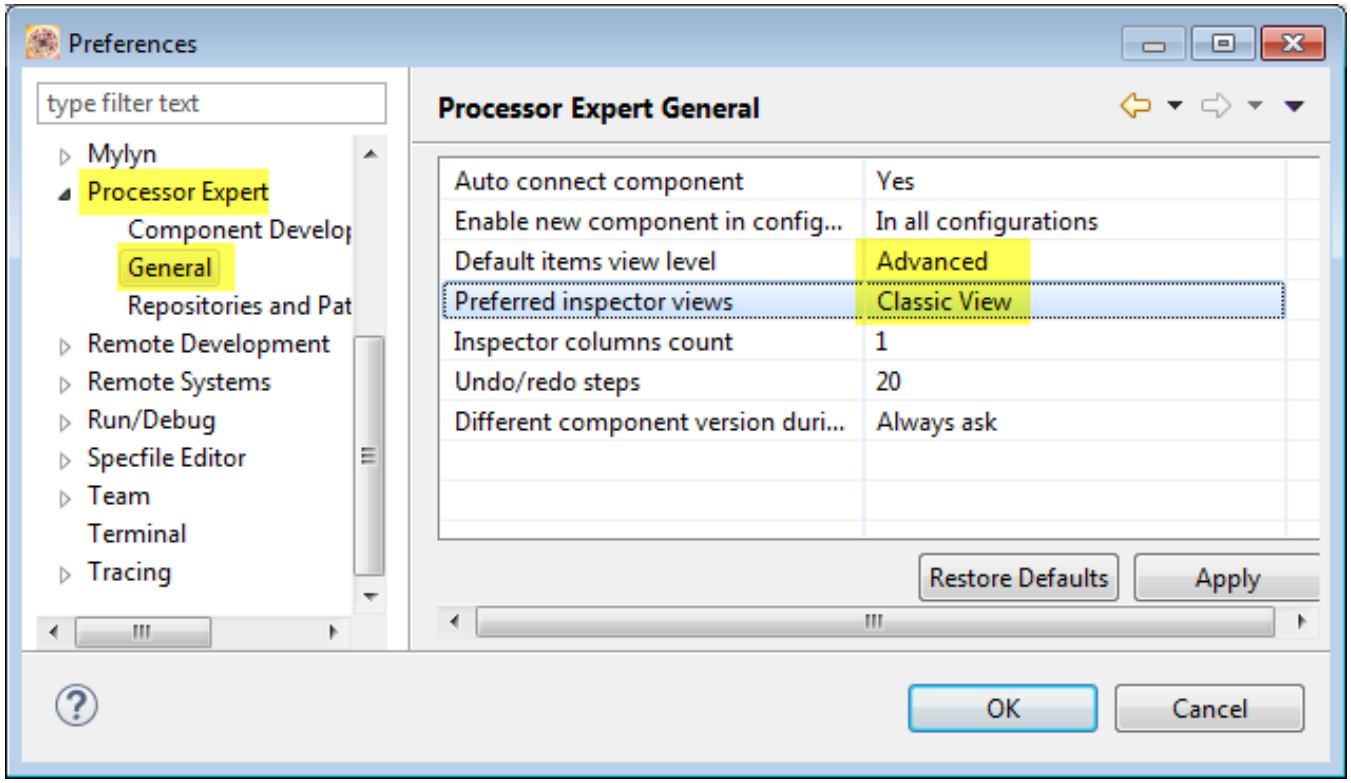
Tip: Processor Expert Classic View

- Disable **Tabs view**
- Needs less space
- Better readability and navigation



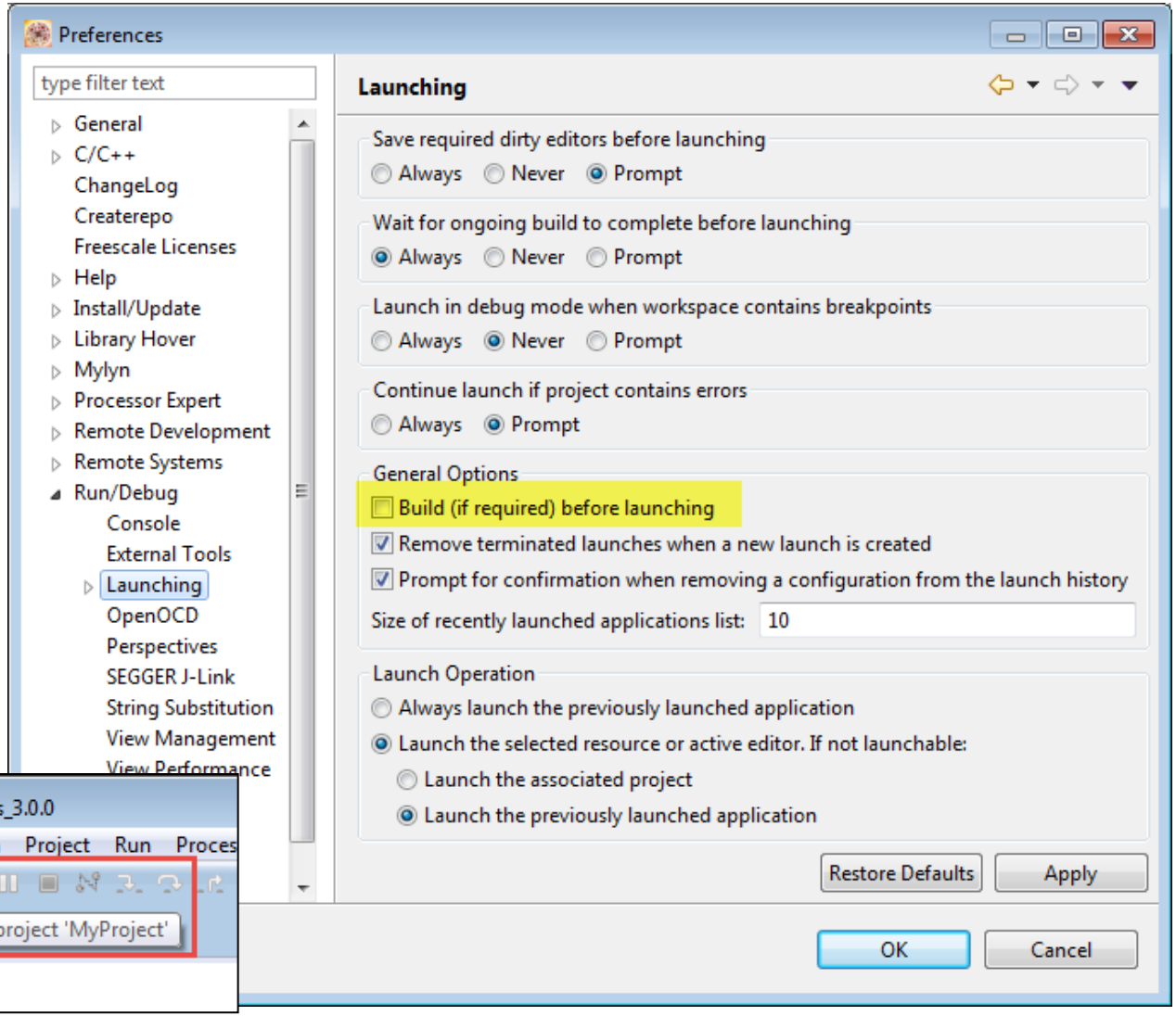
Tip: Making Processor Expert Settings persistent

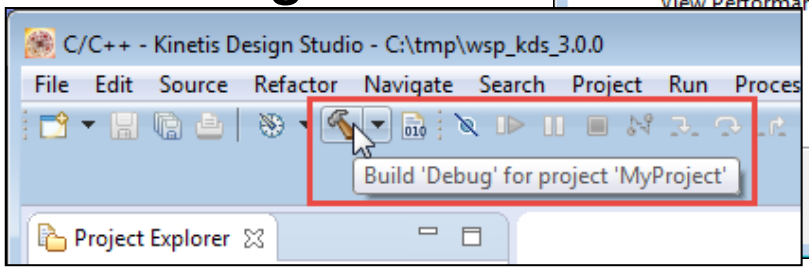
- Menu ***Window > Preferences > Processor Expert > General***
- Settings stored in **Workspace**



Tip: Do not Build Before Debug

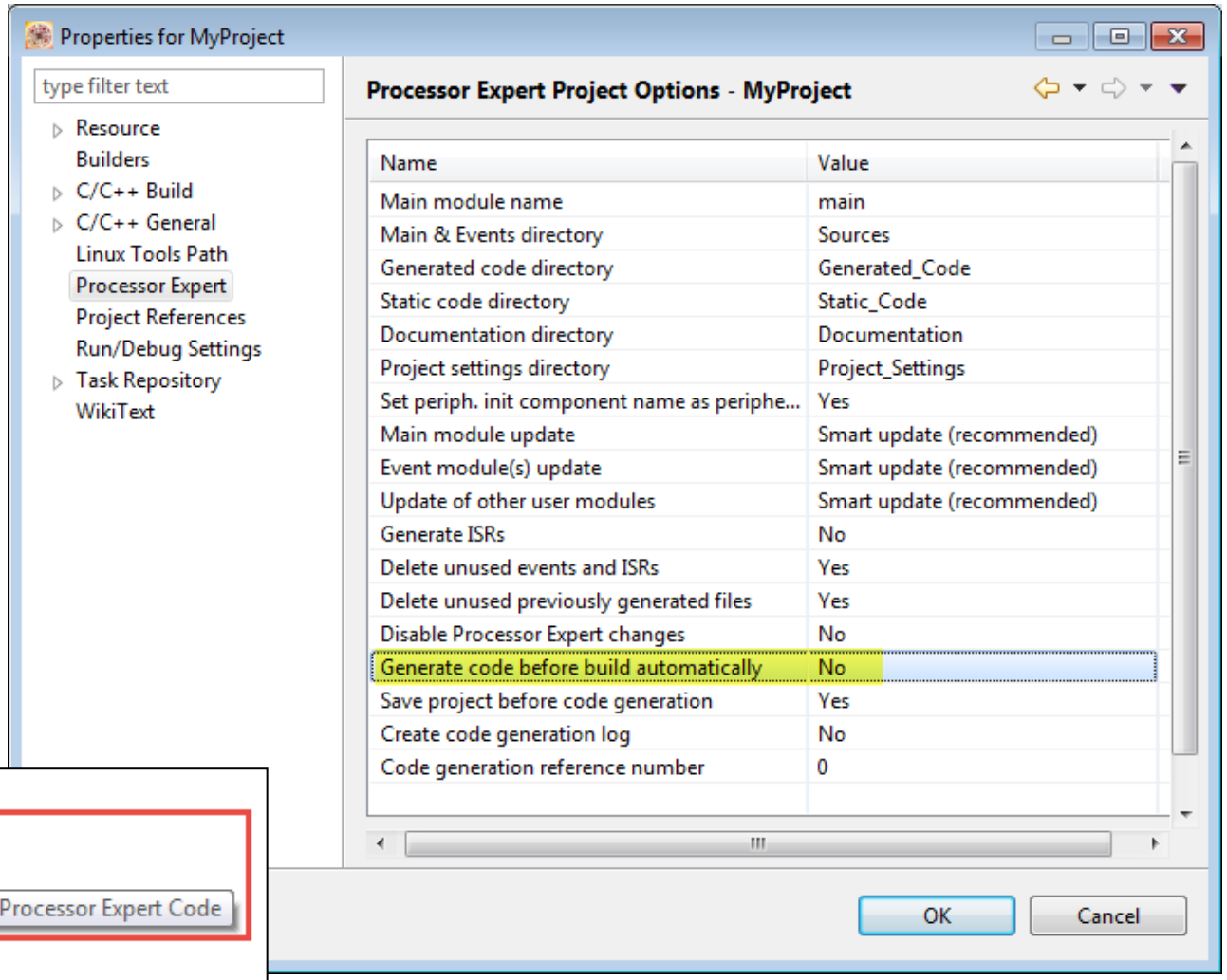
- Workspace setting
- If doing a build anyway before debug
- Menu ***Window > Preferences > Run/Debug > Launching***
- Disable **Build (if required) before launching**

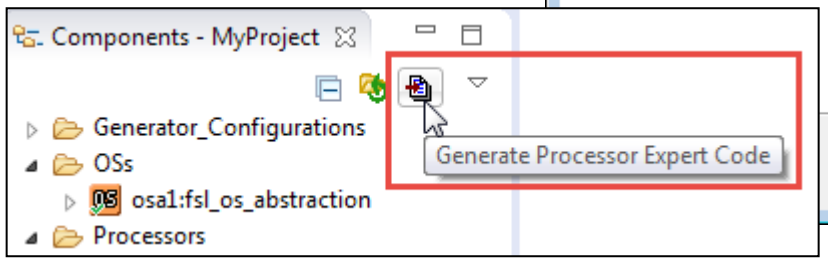




Tip: Do not generate code before build automatically

- **Project > Properties**
- If user generates code after PEx changes
- Can disable **'Generate code before build automatically'**

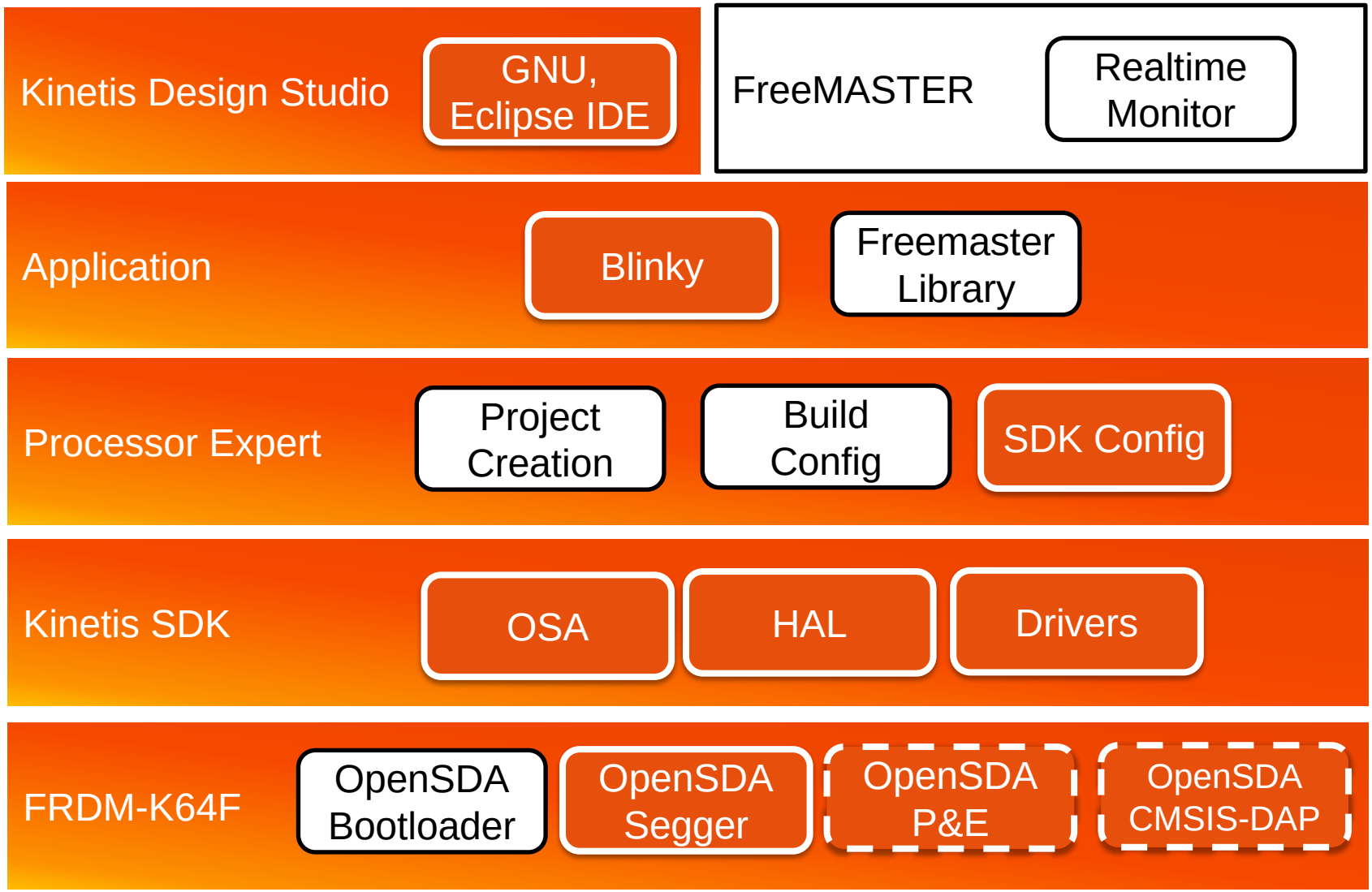




Blinky LED

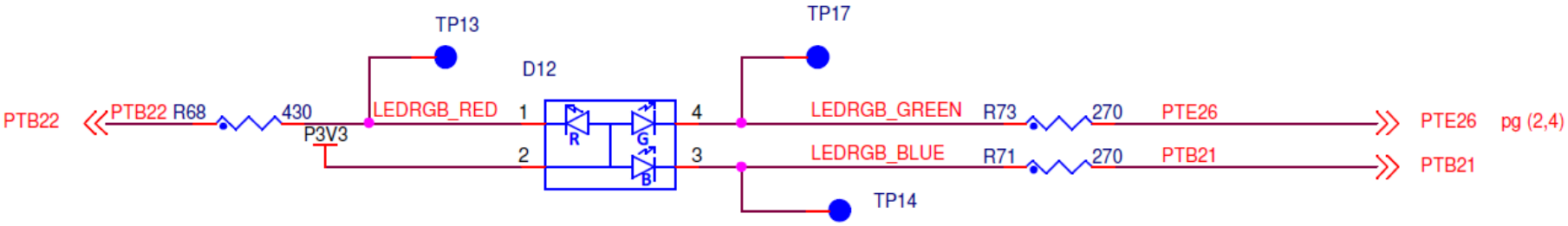
Public

Lab Session Map: Blinky



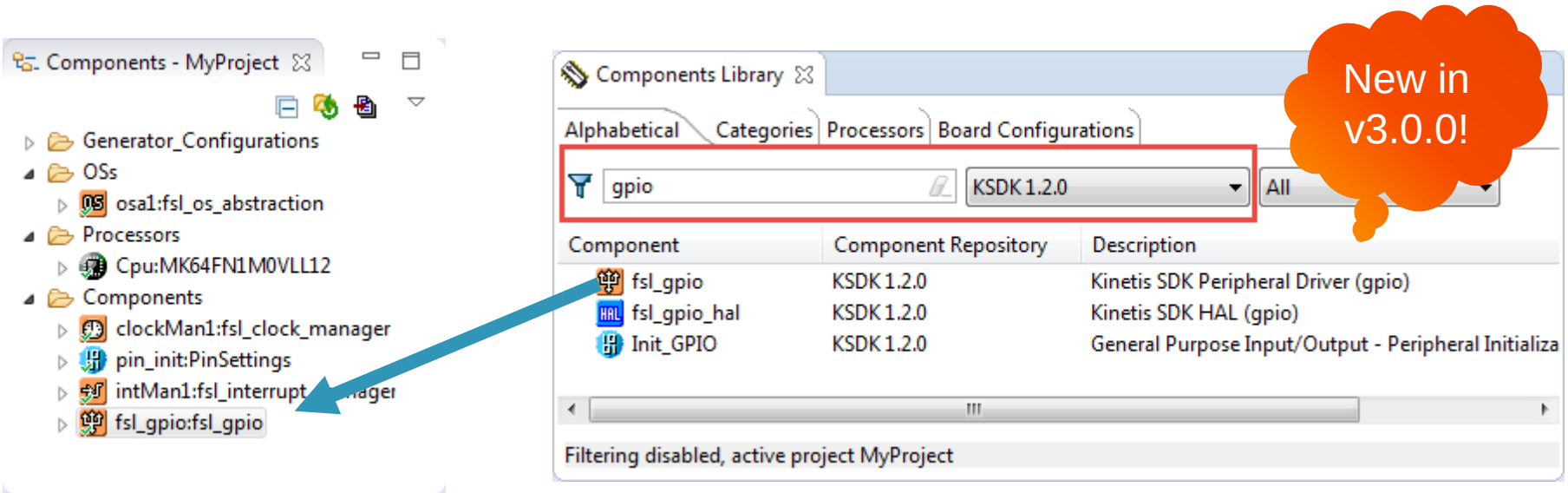
Outline

- Blink the RGB LED on the FRDM-K64F board
  - Red: **PTB22**
  - Green: **PTE26**
  - Blue: **PTB21**
- Drive pins with SDK **fsl_gpio** component



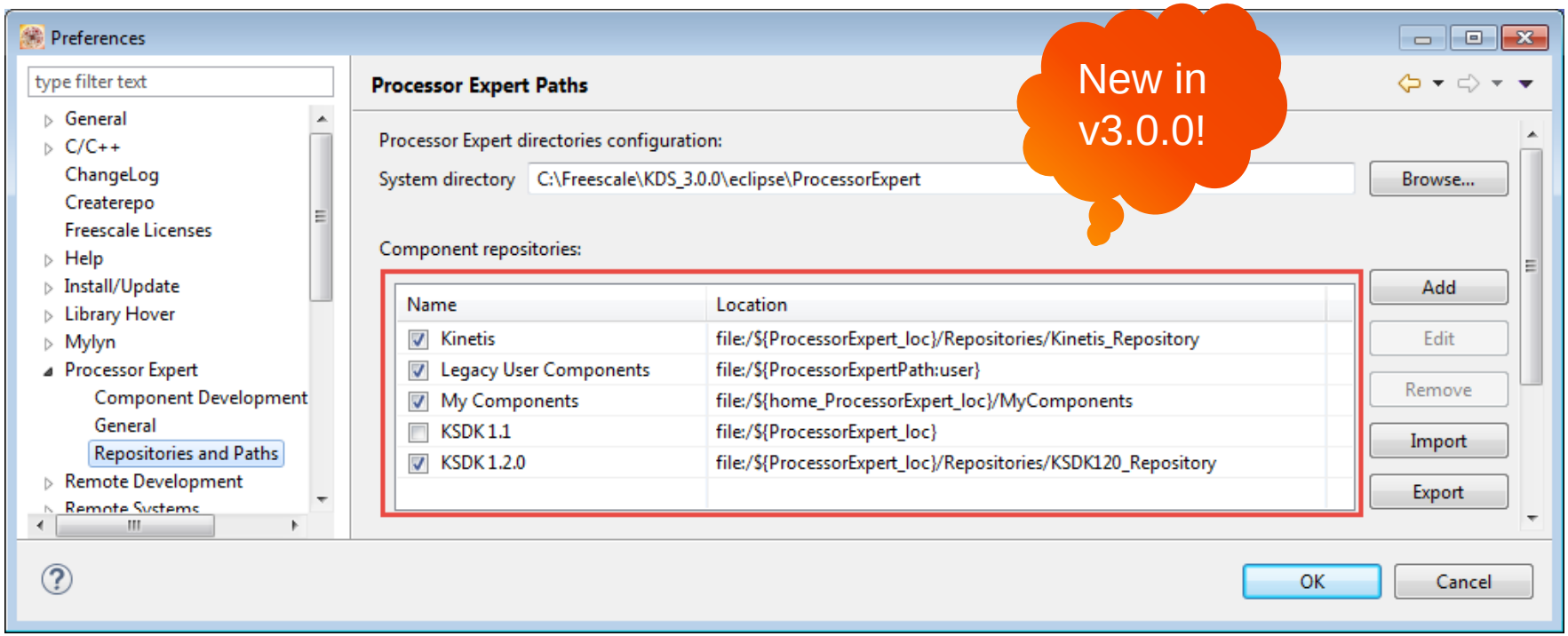
Components Library: add fsl_gpio

- Filter: **gpio**
- Repository selection: **KSDK 1.2.0**
- Double click on **fsl_gpio** to add it to the project
  - Or: drag&drop it to 'Components' view



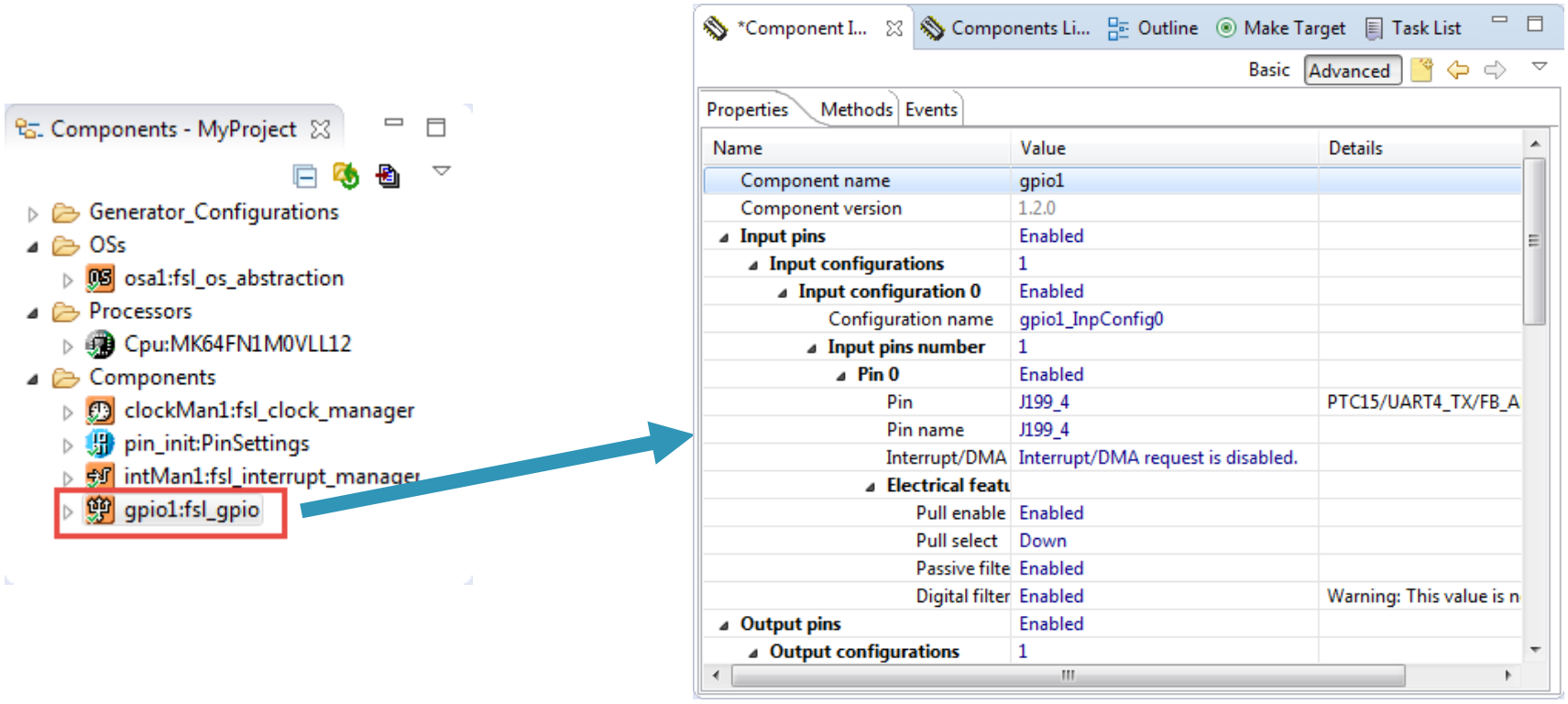
Processor Expert Repositories

- Ability to have multiple repositories with components
- Easy to separate between different versions (e.g. SDK)
- Menu ***Window > Preferences > Processor Expert > Repositories and Paths***



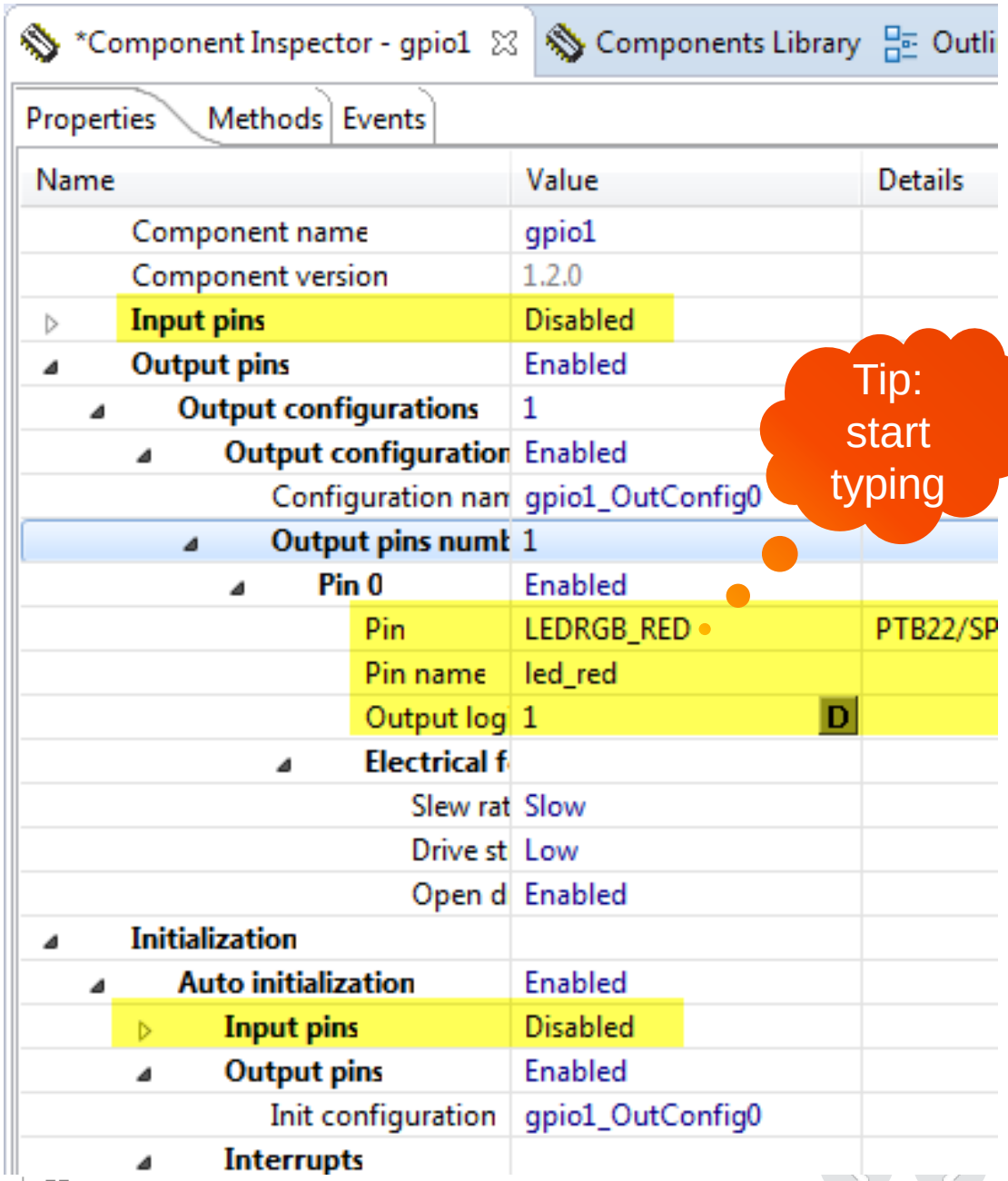
Component Inspector

- Double Click on **gpio1** in Components view to open **Inspector**



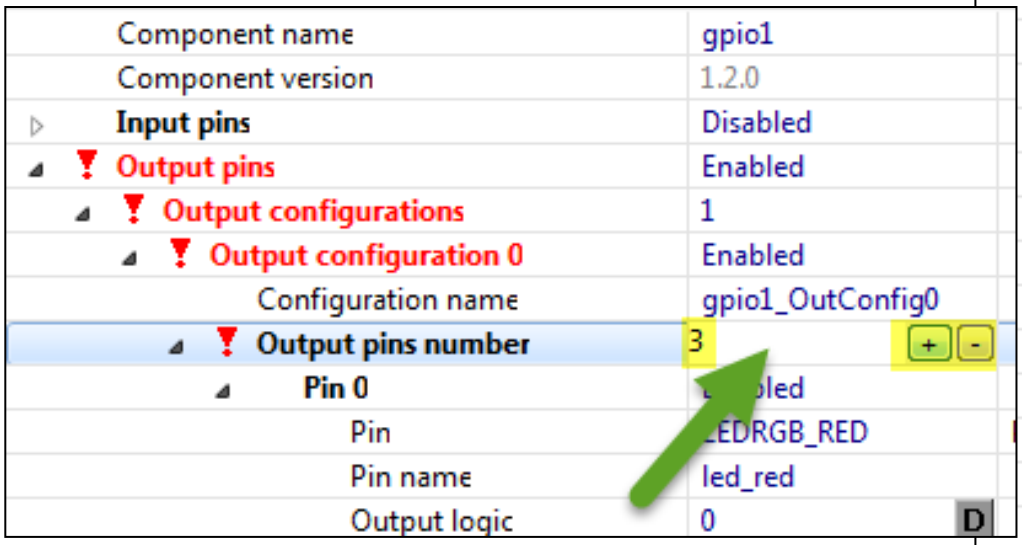
Red LED Configuration

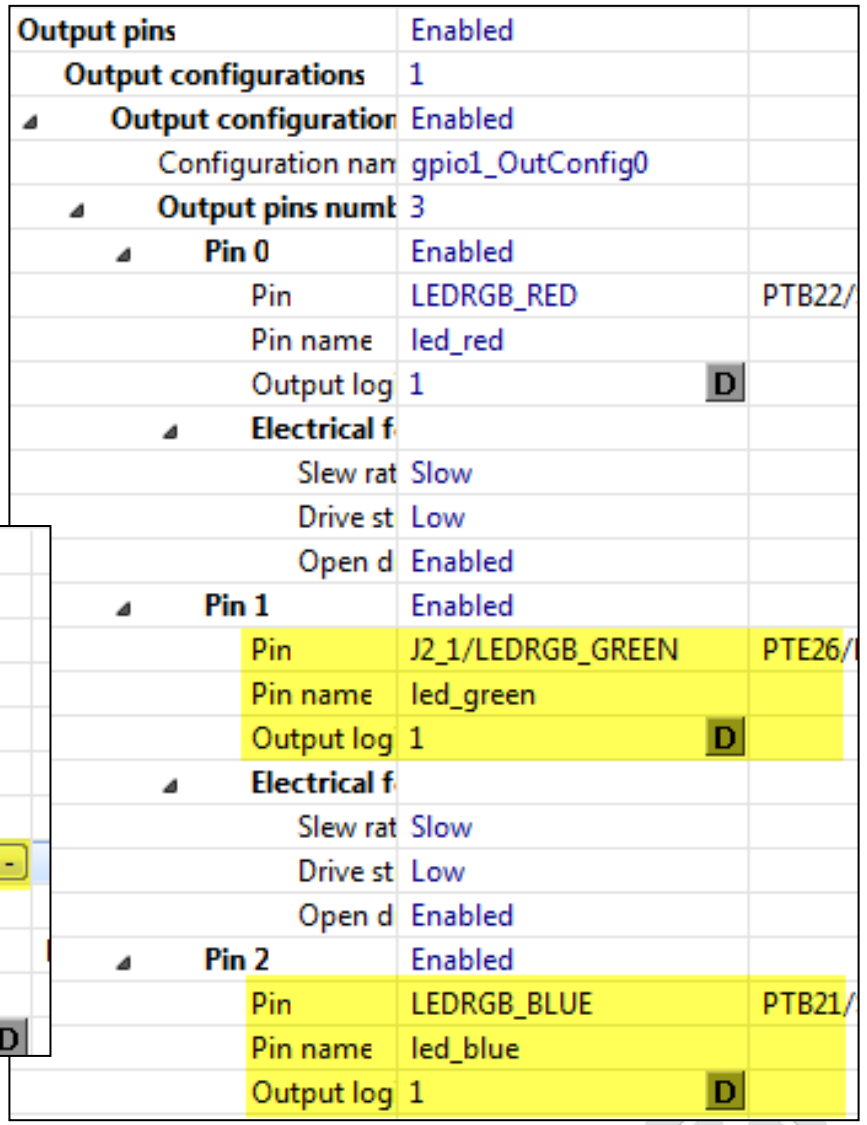
- **Disable Input pins** (we are only using output)
- Set **Output Pin0** Pin to **PTB22**
- Specify Pin Name **led_red**
- Initial **Output logic** to **1** (LED is LOW active)
- In the initialization section, **disable the Input pins**



Adding more: Green and Blue LED

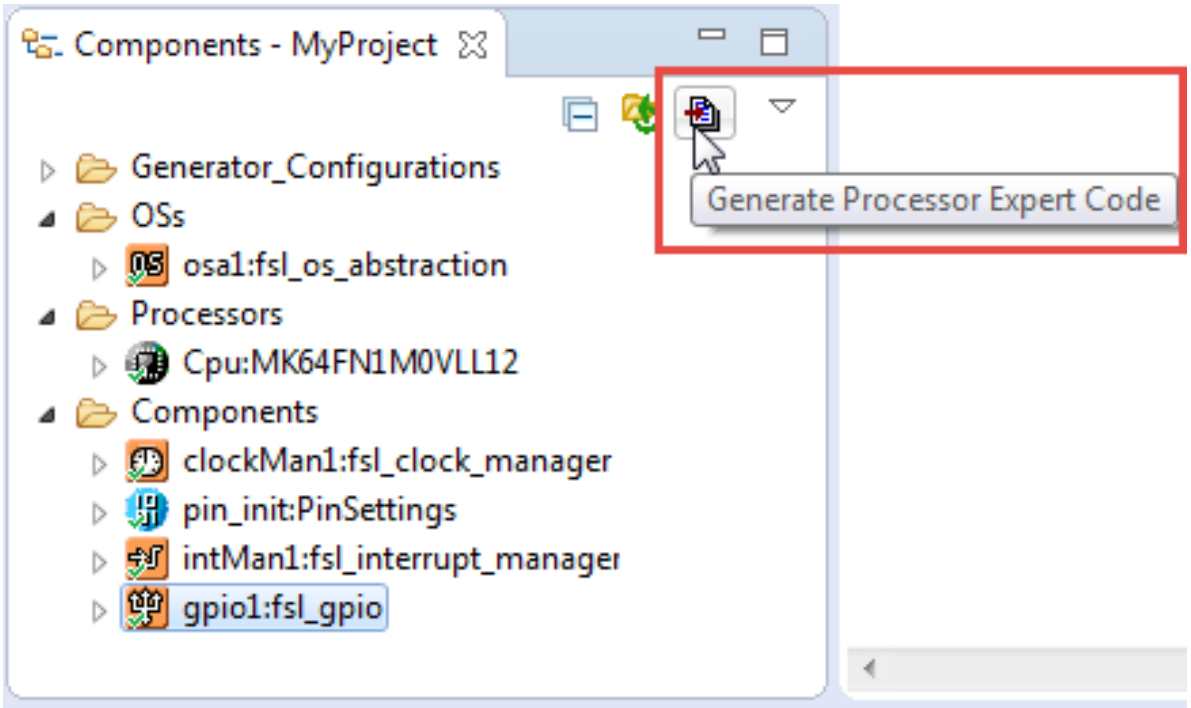
- Click into field for **'Output pins number'**
- Use '+' button to increase it to a total of **3**
- led_green: **PTE26**
- led_blue: **PTB21**

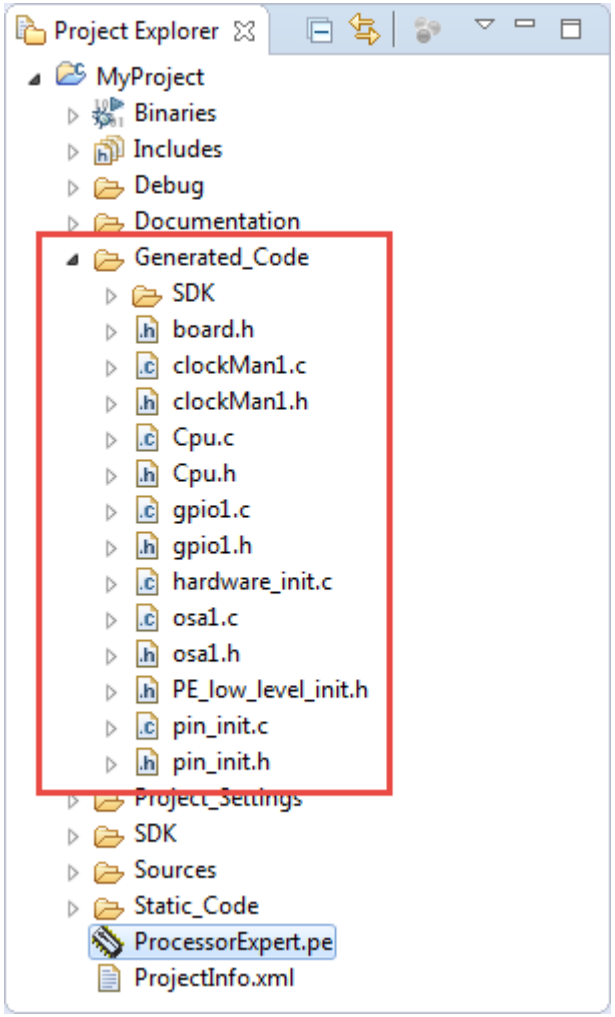




Generating Code

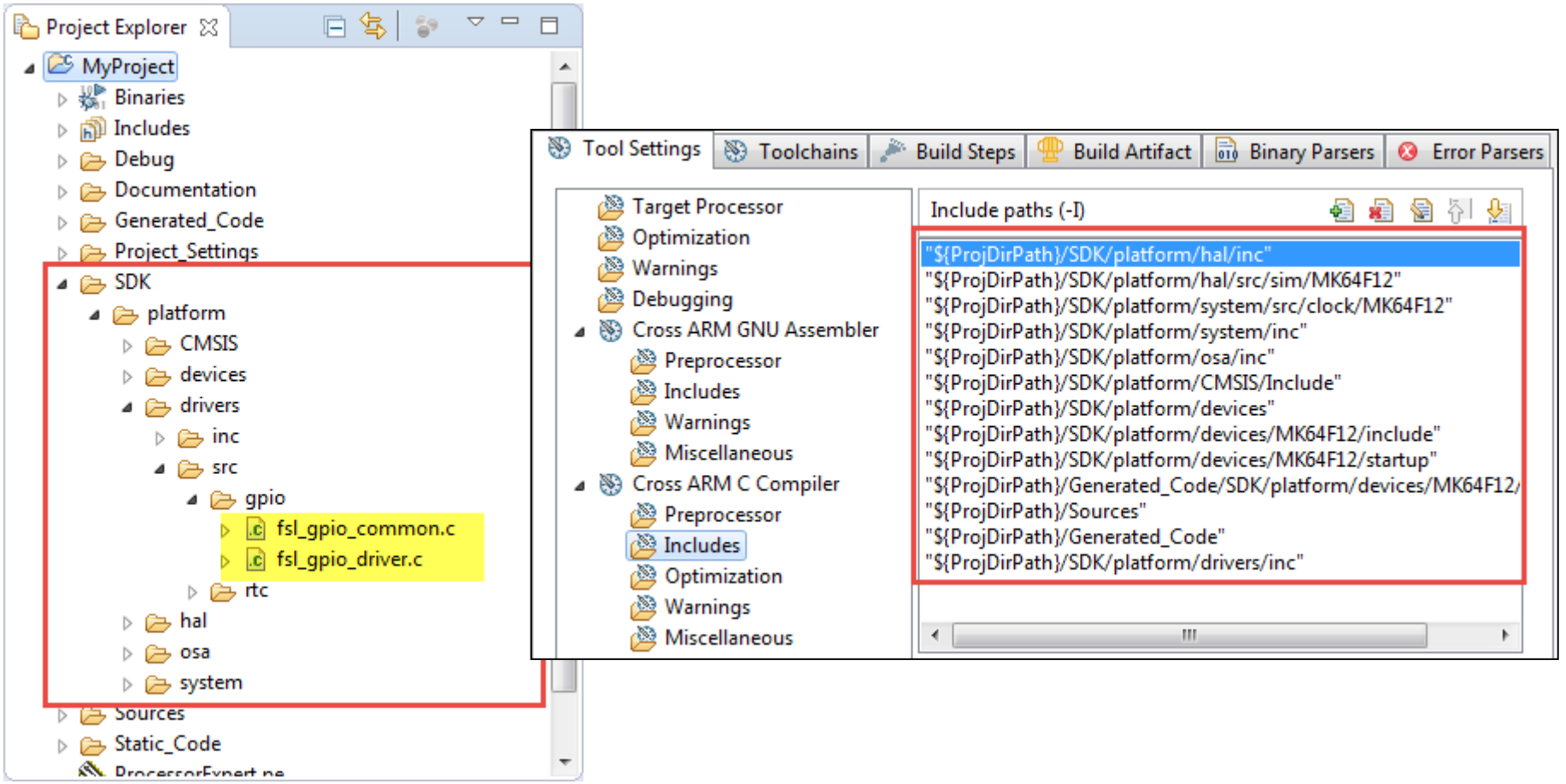
- 'Generate Processor Expert Code' button
- Code in 'Generated_Code' Project Folder





Processor Expert makes usage of SDK easy…

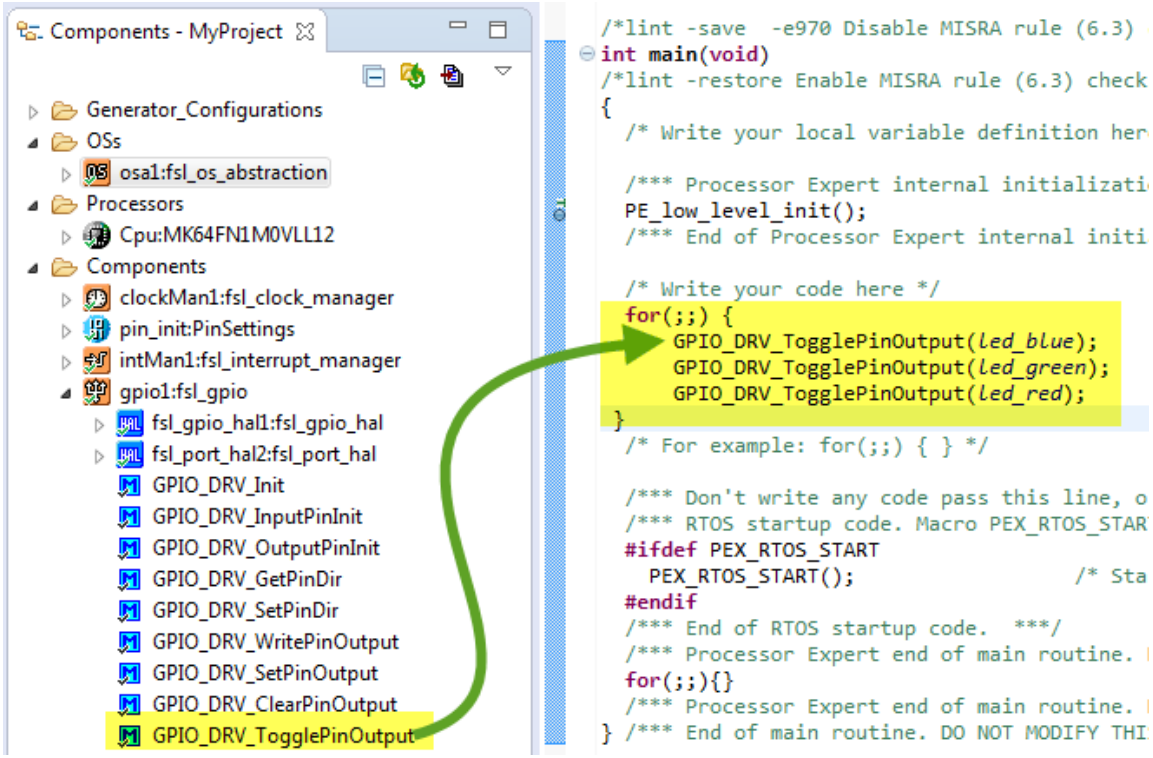
- Adds SDK files and compiler settings to project ☺



Adding Code

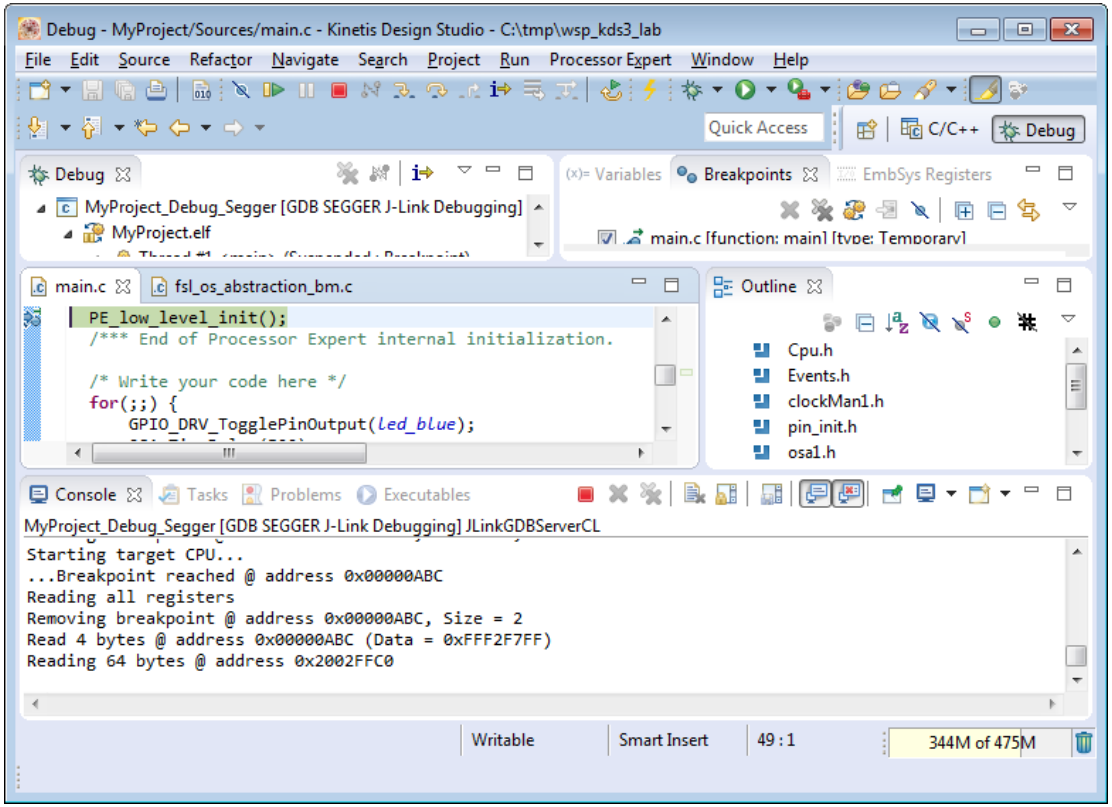
- Open **main.c**
- Add endless loop (**for(;;) .**)
- Add **code** to toggle output pins
  - Tip: Drag&Drop function name

```
for(;;) {
  GPIO_DRV_TogglePinOutput(led_blue);
  GPIO_DRV_TogglePinOutput(led_green);
  GPIO_DRV_TogglePinOutput(led_red);
}
```



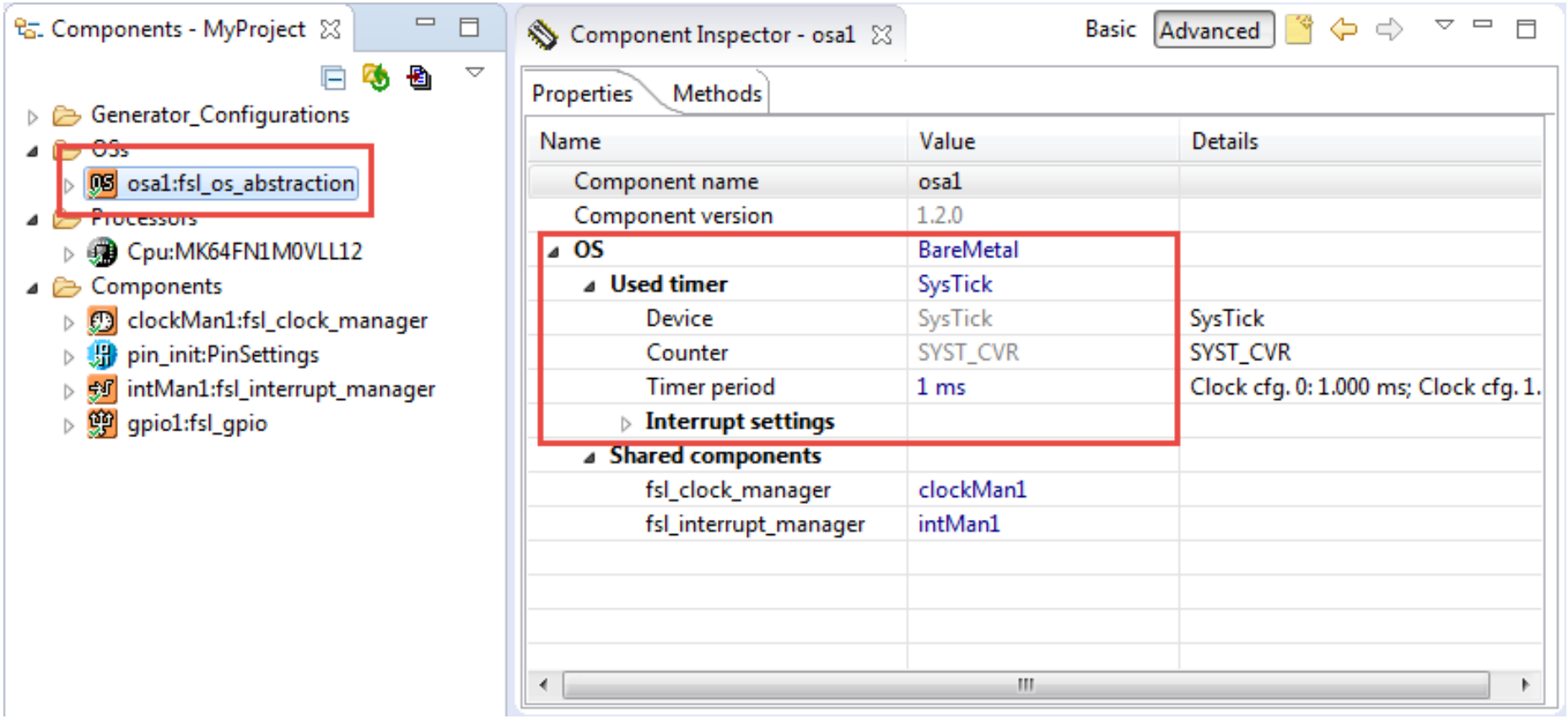
Testing it out…

- Build
- Download and debug
- Step through the code
- LED's shall toggle
- Run the app
  - LED is toggling fast
  - ➔Need a delay



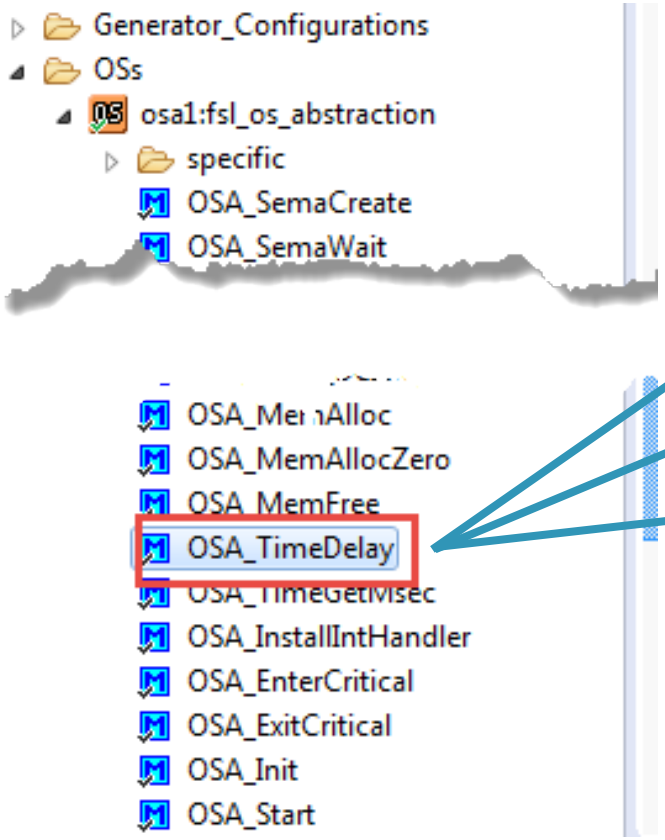
OSA: OS Abstraction

- OSA comes in bare metal with (optional) tick timer
- Can be used for delays



Adding OSA_TimeDelay()

- Delays for the specified number of milliseconds
- Add code, build and debug…




```
for(;;) {
  GPIO_DRV_TogglePinOutput(led_blue);
  OSA_TimeDelay(500);
  GPIO_DRV_TogglePinOutput(led_green);
  OSA_TimeDelay(500);
  GPIO_DRV_TogglePinOutput(led_red);
  OSA_TimeDelay(500);
}
```

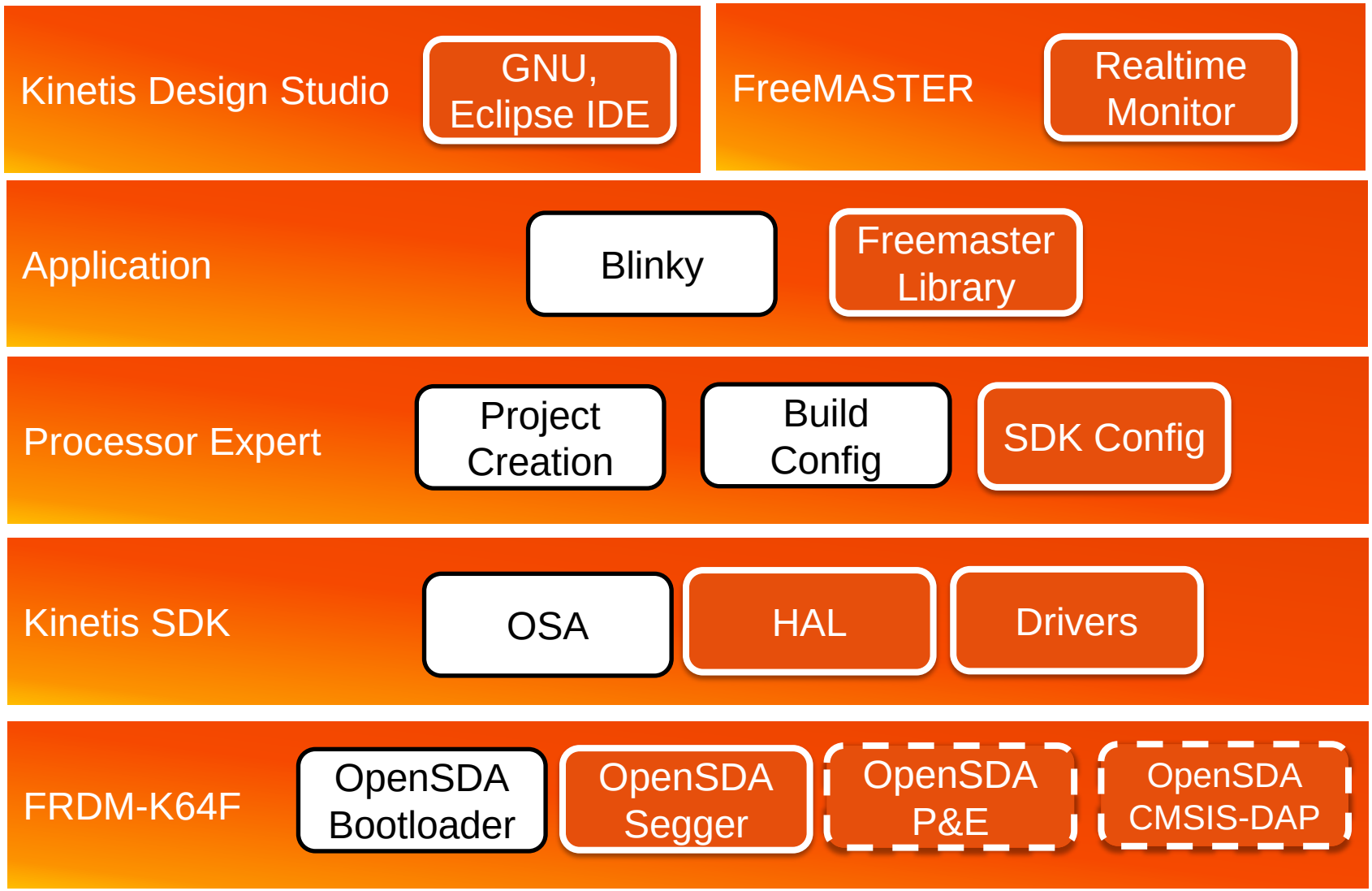
Summary

- Adding SDK for GPIO (used for LED's)
  - HAL
  - Driver
- Processor Expert makes usage of the Kinetis SDK really easy
  - Component Repositories
  - Configuration/Initialization of driver
  - Graphical User interface
  - Adds SDK include paths to compiler settings
  - Adds SDK source files to project
  - Drag&Drop of function/names

FreeMaster

Public

Lab Session Map: FreeMaster



Sample LED Control Code

- Simple demonstration code for LED
- **Global *whichLED*** variable

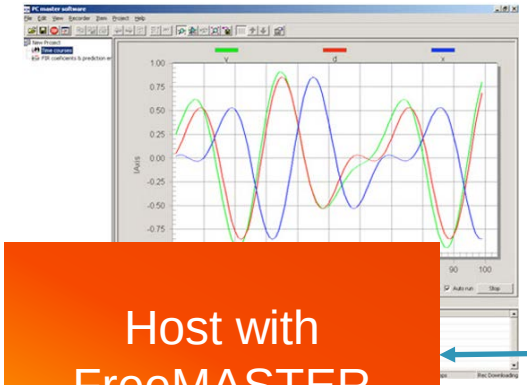
```
int whichLED=1; /* default red */
```

- Code inside **main()**
  - Bits: 1:red, 2:green, 4:blue

```
for(;;) {
  if (whichLED&1) {
    GPIO_DRV_ClearPinOutput(led_red); /* red ON */
  } else {
    GPIO_DRV_SetPinOutput(led_red); /* red OFF */
  }
  if (whichLED&2) {
    GPIO_DRV_ClearPinOutput(led_green); /* green ON */
  } else {
    GPIO_DRV_SetPinOutput(led_green); /* green OFF */
  }
  if (whichLED&4) {
    GPIO_DRV_ClearPinOutput(led_blue); /* blue ON */
  } else {
    GPIO_DRV_SetPinOutput(led_blue); /* blue OFF */
  }
}
```

Benefits of FreeMASTER

- Goal: inspect/change application variables (LED)
- User friendly real-time debug monitor
- Visualization of target states: memory, variables, …
- Read/Write Date while target is running
- Monitoring and debugging
- Prototyping/Demonstration platform



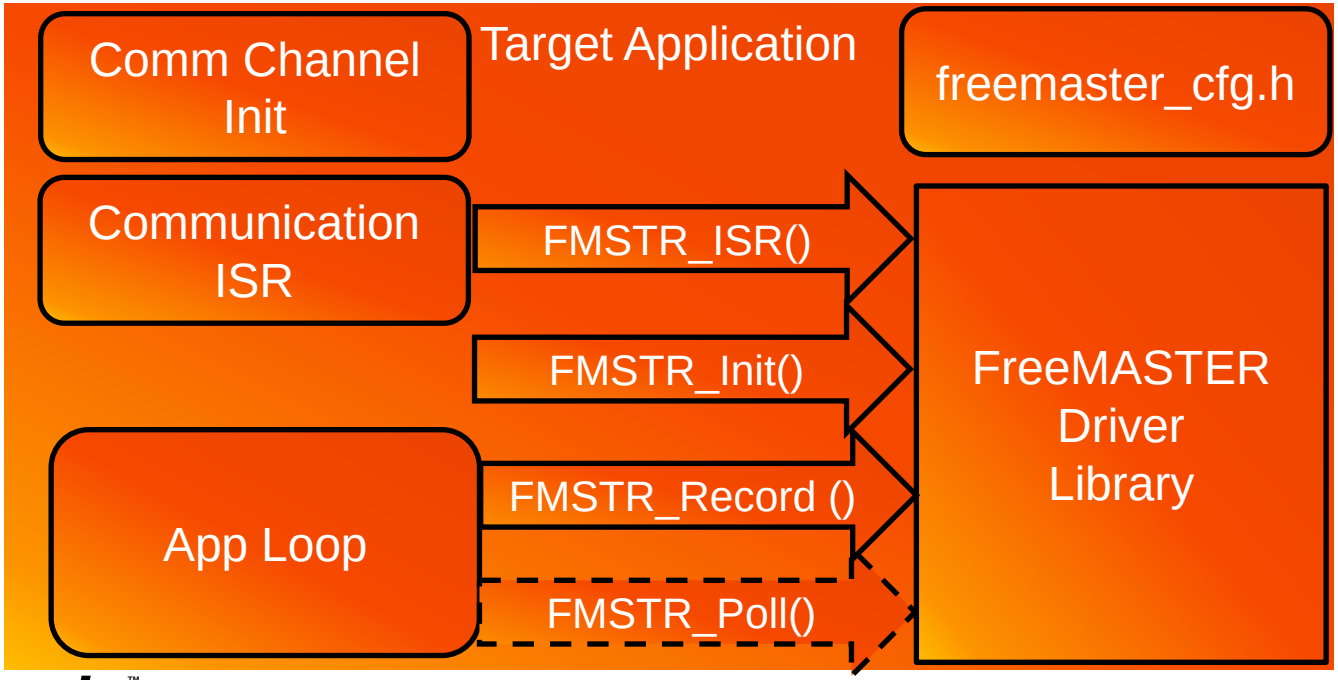
Host with FreeMASTER

Target Application

FreeMASTER Communication Library

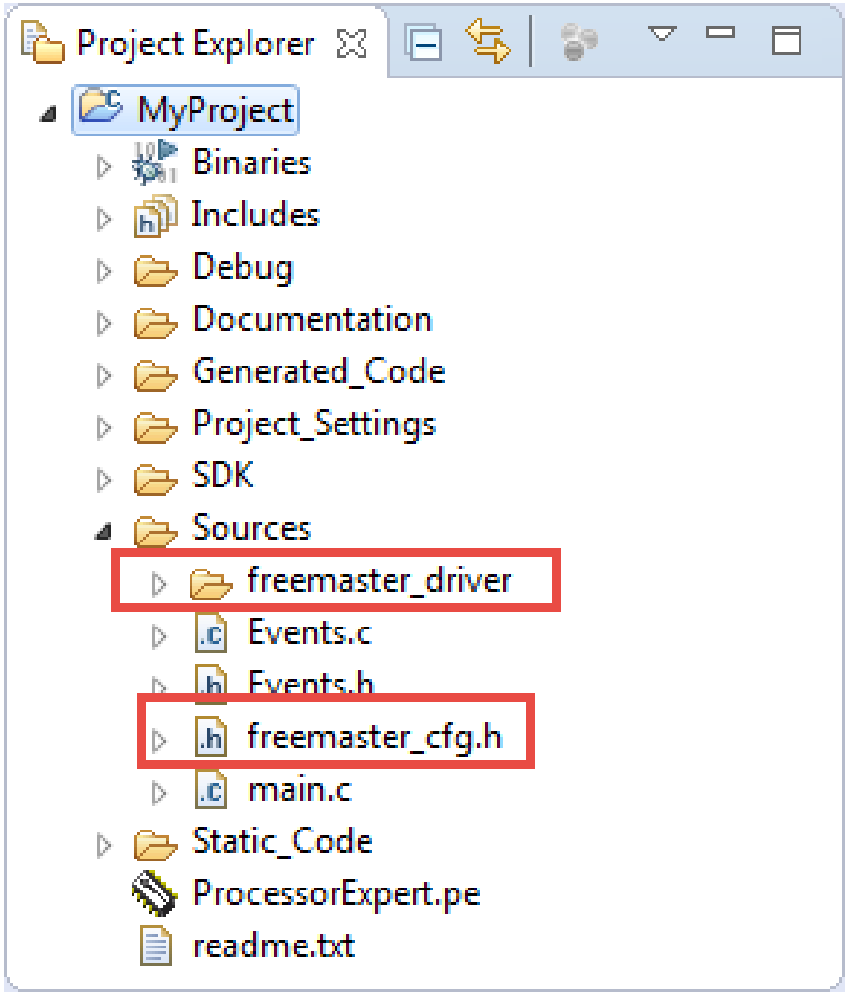
Adding FreeMASTER to Application

- **Initialization** of **communication channel** (**UART**, CAN, …)
- **freemaster_cfg.h** for configuration
- Either **ISR** or **Polling**
  - Polling in Application loop (only needed if not using Interrupts)
  - Recording in Application loop (records variable changes, etc)
- **Initialization** of **Driver Library**



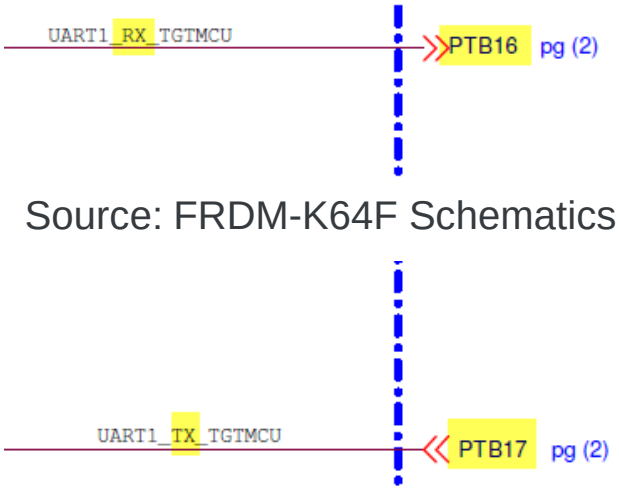
Adding FreeMASTER Communication Drivers

- Copy **freemaster_driver** folder to **Sources** folder
  - Communication Driver Library
- Add **freemaster_cfg.h** file into Sources folder
  - Configures features of FreeMASTER
- Need to add compiler include path
  - Next slide



FreeMASTER UART Init

- UART selection: **UART0**
  - Rx: **PTB16**
  - Tx: **PTB17**
- **115200 baud, 8N1**
- Use **Interrupts**



Source: FRDM-K64F Schematics

uartCom1:fsl_uart_hal

Or:

UART0:Init_UART

Select UART
Clock Rx/Tx port
Configure Muxing for Rx/Tx
Enable peripheral clock
Set Baudrate
Set Bitcount
Set Parity
Set Stop Bit
Implement UART ISR
Enable UART Interrupt

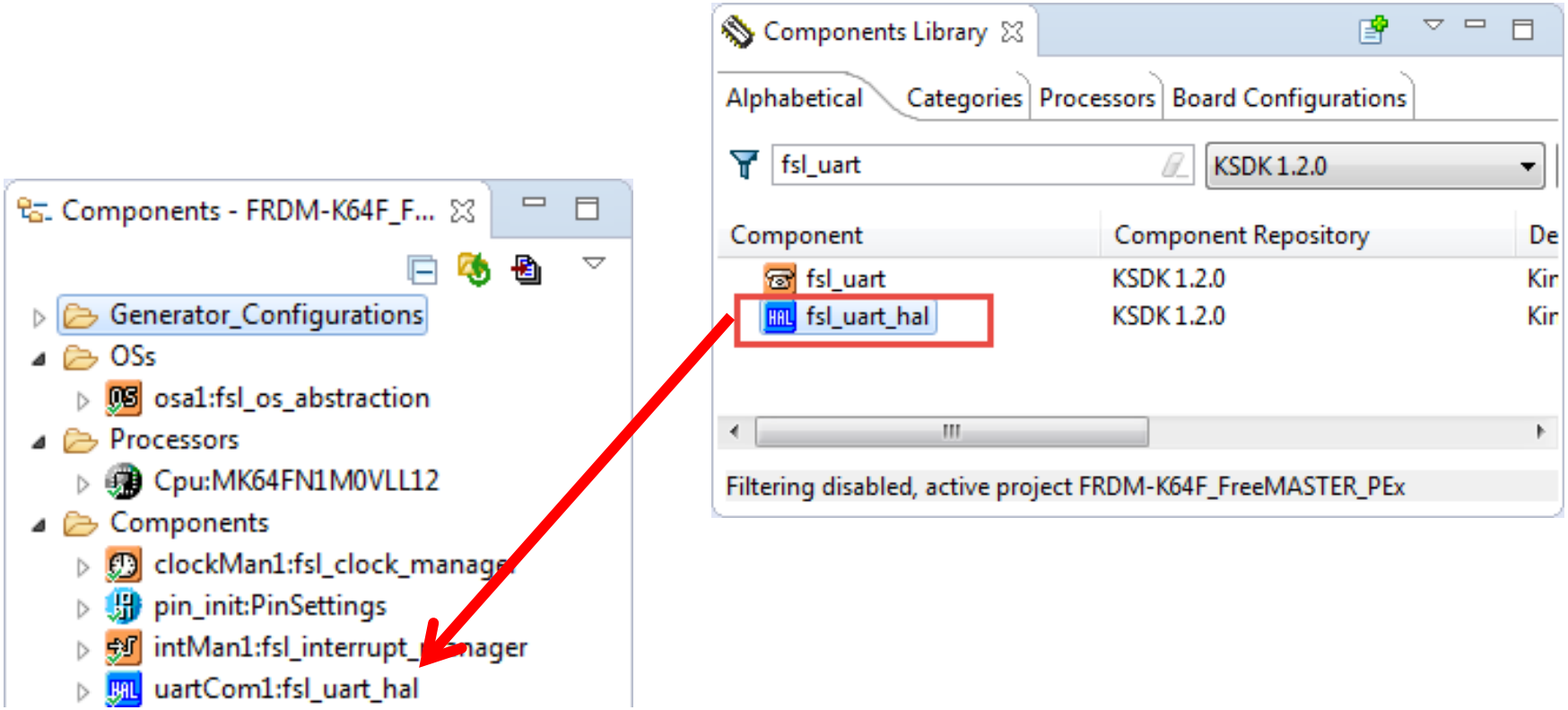
UART Init: UART HAL Version

uartCom1:fsl_uart_hal

```
static void init_freemaster_uart(void) {
  const uint32_t instance =  0; /* 0: UART0, 1: UART1, ... */

  SIM_HAL_EnableClock(SIM,kSimClockGatePortB); /* Enable clock to PORTB */
  PORT_HAL_SetMuxMode(PORTB,16u,kPortMuxAlt3); /* Mux PORTB_PCR16 */
  PORT_HAL_SetMuxMode(PORTB,17u,kPortMuxAlt3); /* Mux PORTB_PCR17 */
  CLOCK_SYS_EnableUartClock(instance); /* Un-gate UART module clock */
  UART_HAL_Init((UART_Type*)UART0_BASE); /* UART low-level initialization */
  UART_HAL_SetBaudRate((UART_Type*)UART0_BASE,
             CLOCK_SYS_GetUartFreq(instance), 115200);
  UART_HAL_SetBitCountPerChar((UART_Type*)UART0_BASE, kUart8BitsPerChar);
  UART_HAL_SetParityMode((UART_Type*)UART0_BASE, kUartParityDisabled);
  UART_HAL_SetStopBitCount((UART_Type*)UART0_BASE, kUartOneStopBit);
  INT_SYS_EnableIRQ(UART0_RX_TX_IRQn); /* enable UART IRQ */
  FMSTR_SetSciBaseAddress((FMSTR_ADDR)UART0_BASE); /* set base address */
}
void UART0_RX_TX_IRQHandler(void) {
  FMSTR_Isr();
}
```

Solution with fsl_uart_hal

- Add fsl_uart_hal to the project



UART Init: Init_UART Version


```
static void init_freemaster_uart(void) {
  const uint32_t instance =  0; /* 0: UART0, 1: UART1, ... */

  SIM_HAL_EnableClock(SIM,kSimClockGatePortB); /* Enable clock to PORTB */
  PORT_HAL_SetMuxMode(PORTB,16u,kPortMuxAlt3); /* Mux PORTB_PCR16 */
  PORT_HAL_SetMuxMode(PORTB,17u,kPortMuxAlt3); /* Mux PORTB_PCR17 */
  CLOCK_SYS_EnableUartClock(instance); /* Un-gate UART module clock */
  UART_HAL_Init((UART_Type*)UART0_BASE); /* UART low-level initialization */
  UART_HAL_SetBaudRate((UART_Type*)UART0_BASE,
            CLOCK_SYS_GetUartFreq(instance), 115200);
  UART_HAL_SetBitCountPerChar((UART_Type*)UART0_BASE, kUart8BitsPerChar);
  UART_HAL_SetParityMode((UART_Type*)UART0_BASE, kUartParityDisabled);
  UART_HAL_SetStopBitCount((UART_Type*)UART0_BASE, kUartOneStopBit);
  INT_SYS_EnableIRQ(UART0_RX_TX_IRQn); /* enable UART IRQ */
  FMSTR_SetSciBaseAddress((FMSTR_ADDR)UART0_BASE); /* set base address */
}
void UART0_RX_TX_IRQHandler(void) {
  FMSTR_Isr();
}
```

Add to main.c

Add UART RX/TX ISR

- Add ISR to main.c

```
#include "freemaster.h"

void UART0_RX_TX_IRQHandler(void) {
  FMSTR_Isr();
}
```

main.c

```
#include "Cpu.h"
#include "Events.h"
#include "clockMan1.h"
#include "pin_init.h"
#include "osa1.h"
#include "gpio1.h"
#if CPU_INIT_CONFIG
  #include "Init_Config.h"
#endif
/* User includes (#include below this line is not maintained by Processor Expert) */
#include "freemaster.h"

void UART0_RX_TX_IRQHandler(void) {
  FMSTR_Isr();
}
```

Add Init() and Recorder()

*main.c

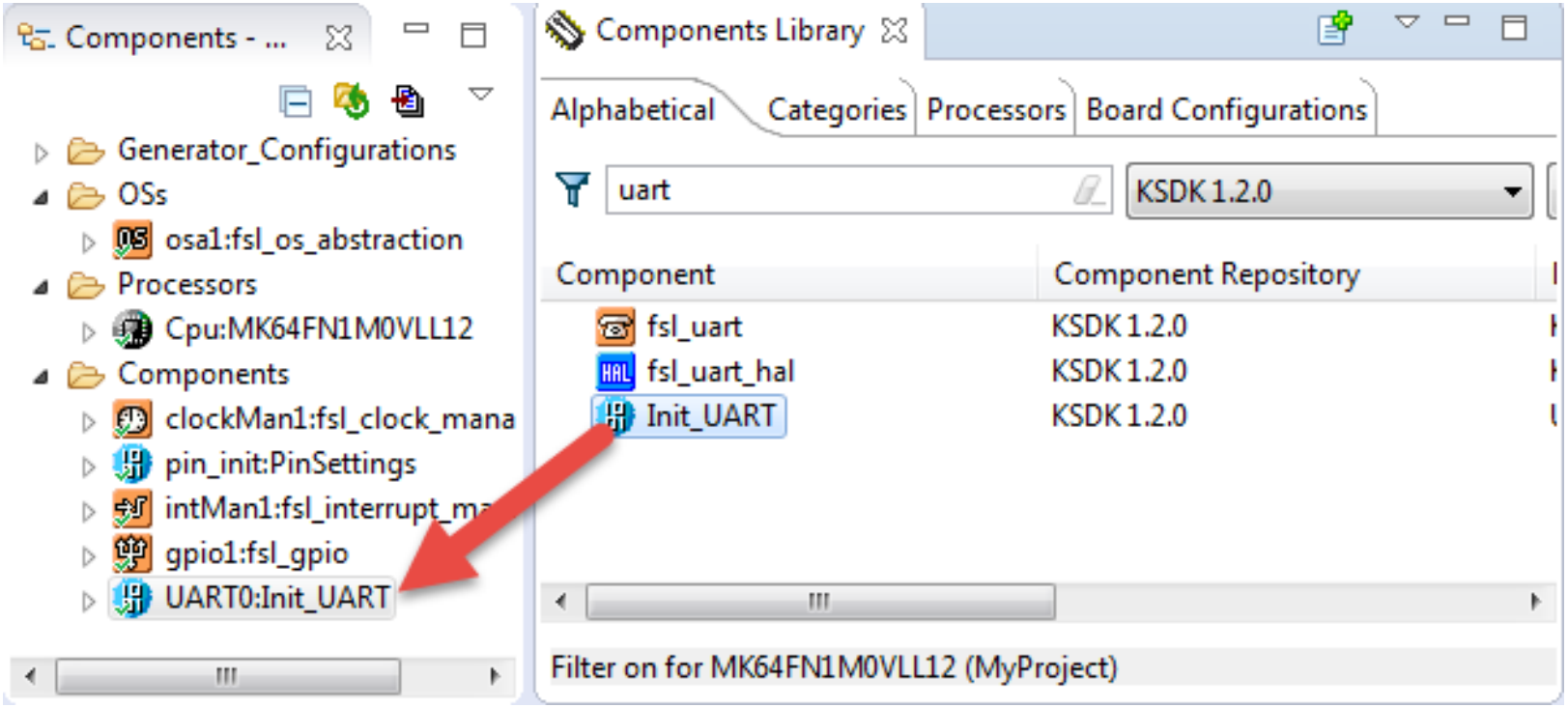
```
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initializati                    ***/

  /* Write your co  here */
  FMSTR_Init();  /* Init FreeMASTER driver */
  for(;;) {
      if (whichLED&1) {
        GPIO_DRV_ClearPinOutput(led_red); /* red ON */
      } else {
        GPIO_DRV_SetPinOutput(led_red); /* red OFF */
      }
      if (whichLED&2) {
        GPIO_DRV_ClearPinOutput(led_green); /* green ON */
      } else {
        GPIO_DRV_SetPinOutput(led_green); /* green OFF */
      }
      if (whichLED&4) {
        GPIO_DRV_ClearPinOutput(led_blue); /* blue ON */
      } else {
        GPIO_DRV_SetPinOutput(led_blue); /* blue OFF */
      }
      FMSTR_Recorder(); /* Record variable changes */
      //FMSTR_Poll(); /* not needed, as using interrupt mode */
  }
  /* For example: for(;;) { } */

  /*** Don't write any code pass this line, or it will be deleted during code ger
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component.
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is
  #endif
```

FMSTR_Init();

FMSTR_Recorder();

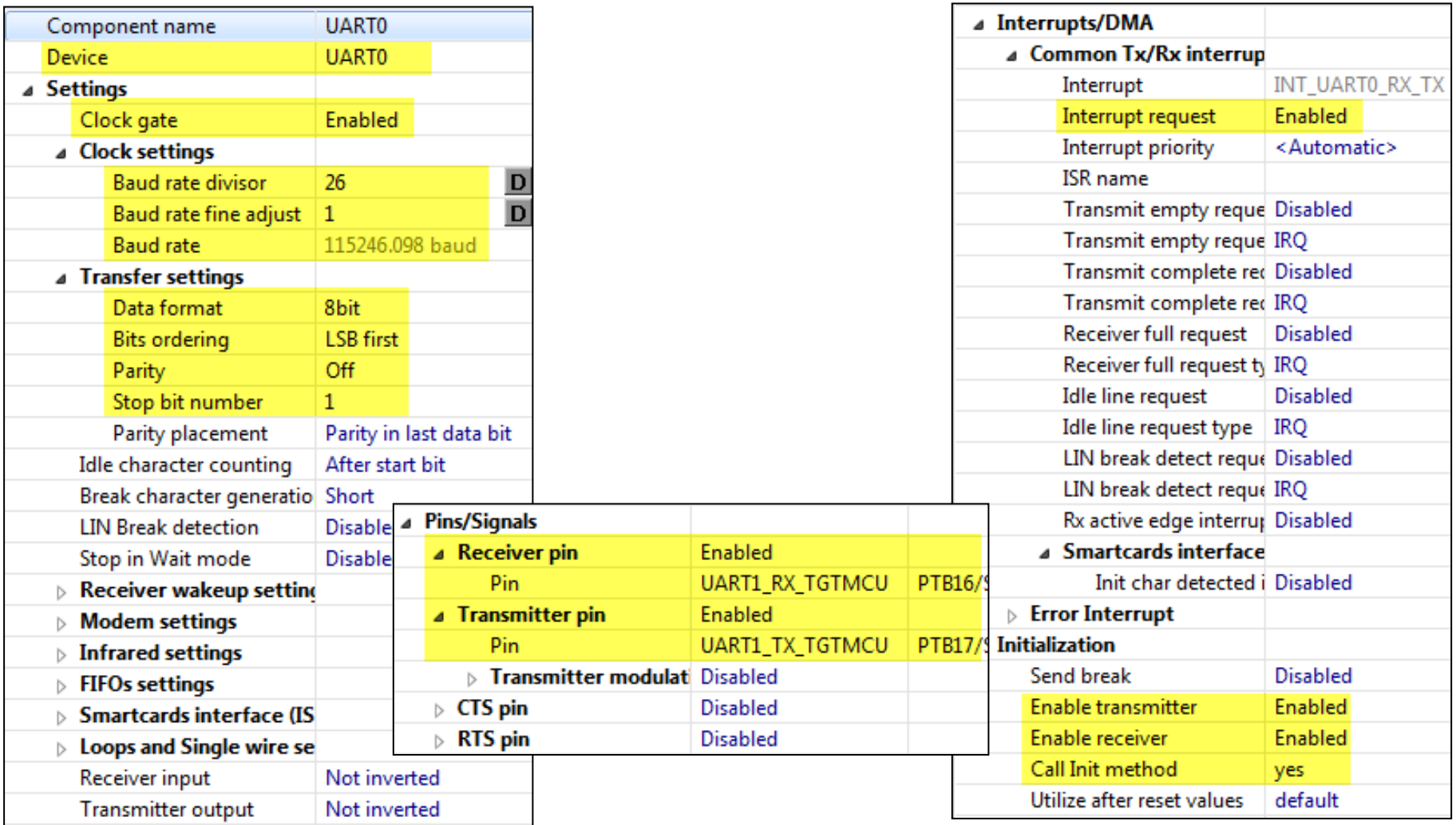
Adding Init_UART

- Add **Init_UART** component to project
  - Double click
  - Or: drag&drop



Configure Init_UART Component

UART0:Init_UART



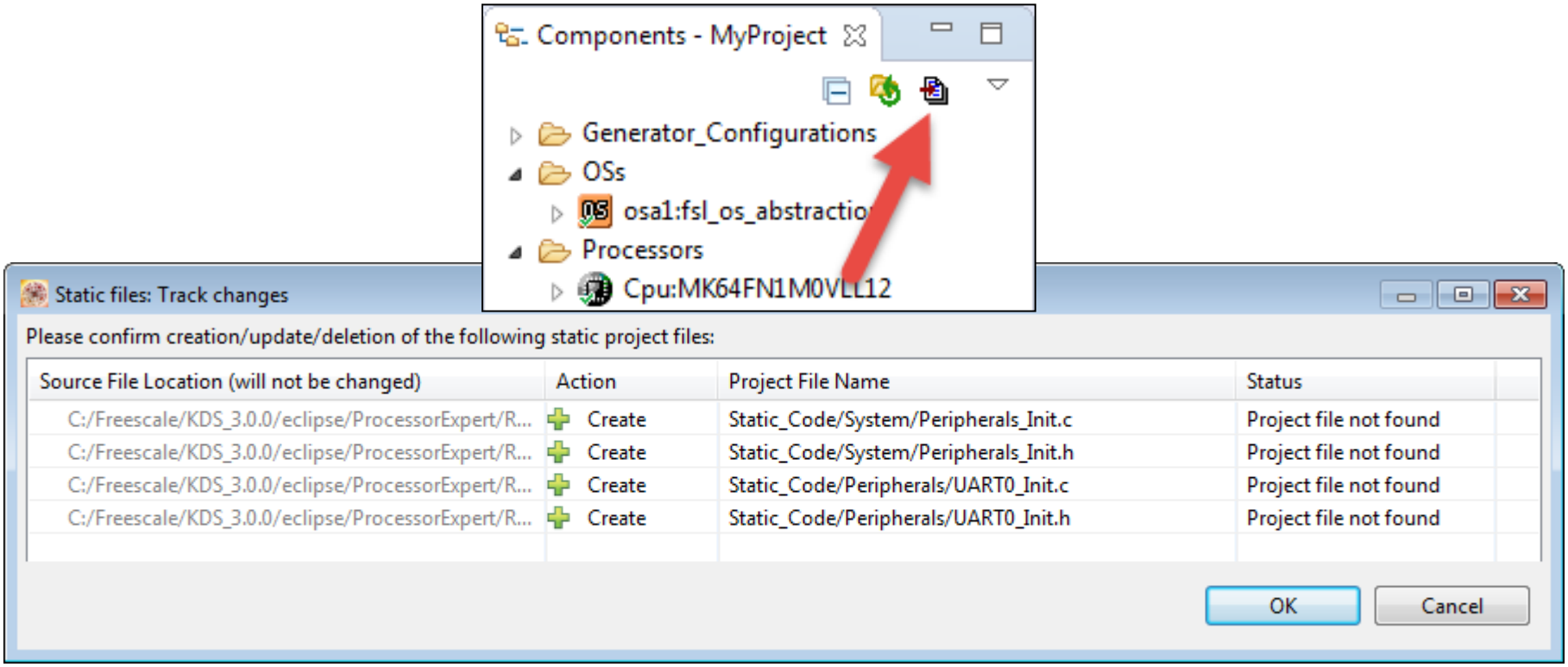
Component name	UART0
Device	UART0
Settings	
Clock gate	Enabled
Clock settings	
Baud rate divisor	26 D
Baud rate fine adjust	1 D
Baud rate	115246.098 baud
Transfer settings	
Data format	8bit
Bits ordering	LSB first
Parity	Off
Stop bit number	1
Parity placement	Parity in last data bit
Idle character counting	After start bit
Break character generatio	Short
LIN Break detection	Disable
Stop in Wait mode	Disable
Receiver wakeup setting	
Modem settings	
Infrared settings	
FIFOs settings	
Smartcards interface (IS	
Loops and Single wire se	
Receiver input	Not inverted
Transmitter output	Not inverted

Pins/Signals		
Receiver pin	Enabled	
Pin	UART1_RX_TGTMCU	PTB16/
Transmitter pin	Enabled	
Pin	UART1_TX_TGTMCU	PTB17/
Transmitter modulat	Disabled	
CTS pin	Disabled	
RTS pin	Disabled	

Interrupts/DMA	
Common Tx/Rx interrup	
Interrupt	INT_UART0_RX_TX
Interrupt request	Enabled
Interrupt priority	<Automatic>
ISR name	
Transmit empty reque	Disabled
Transmit empty reque	IRQ
Transmit complete rec	Disabled
Transmit complete rec	IRQ
Receiver full request	Disabled
Receiver full request ty	IRQ
Idle line request	Disabled
Idle line request type	IRQ
LIN break detect reque	Disabled
LIN break detect reque	IRQ
Rx active edge interrup	Disabled
Smartcards interface	
Init char detected i	Disabled
Error Interrupt	
Initialization	
Send break	Disabled
Enable transmitter	Enabled
Enable receiver	Enabled
Call Init method	yes
Utilize after reset values	default

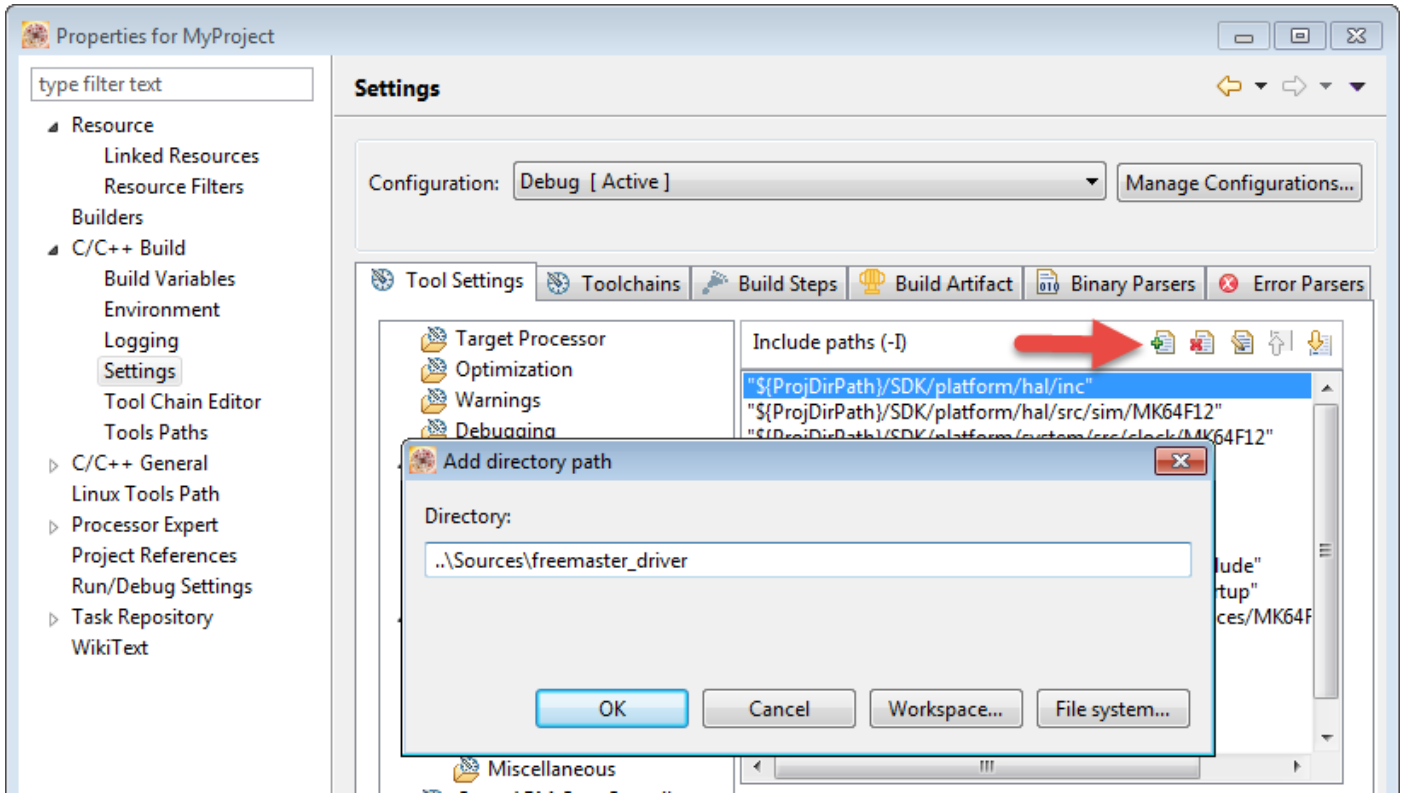
Generating UART Initialization Code

- Generate Code again
- Dialog will inform that additional static source files have been added



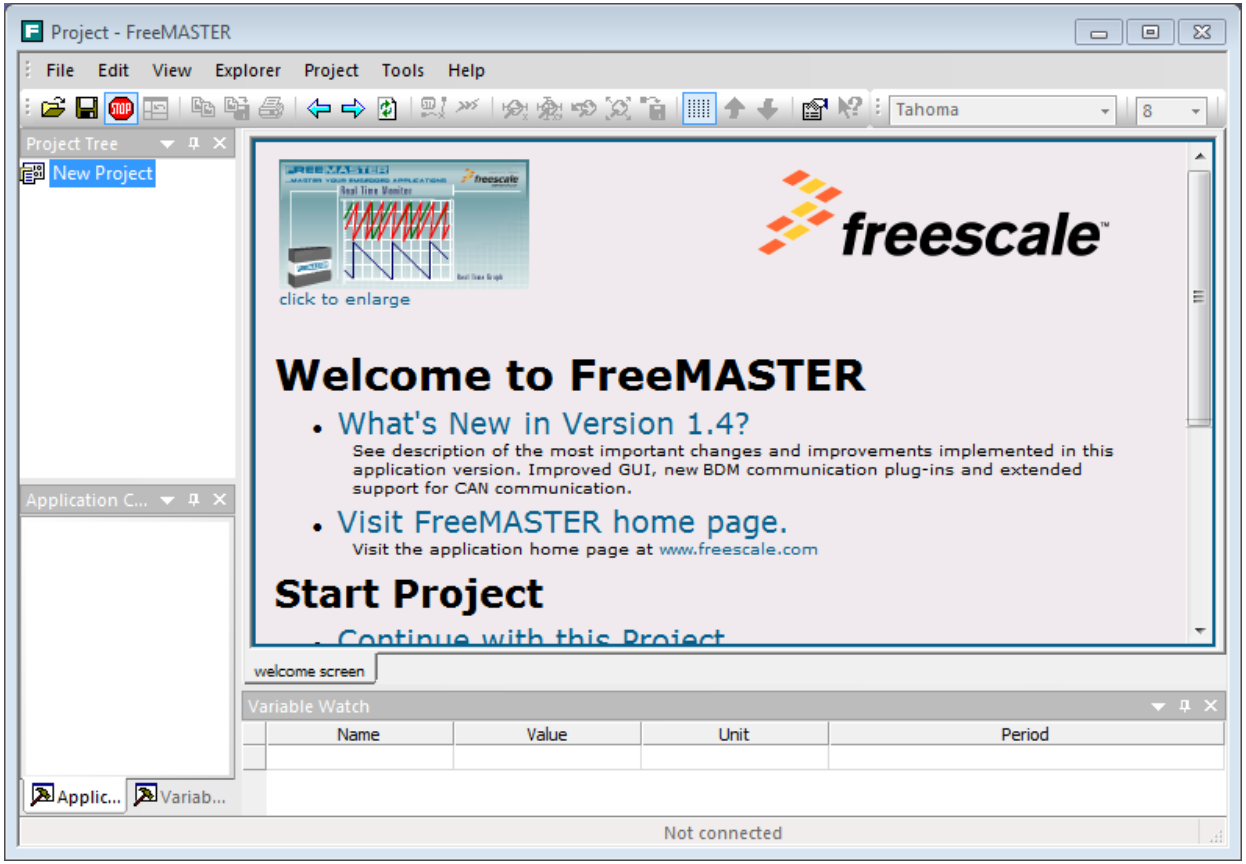
Add Include Path

- Menu ***Project > Properties > C/C++ Build > Settings***
- **Cross ARM C Compiler > Includes**
- Use '**+**' to add **..\Sources\freemaster_driver**



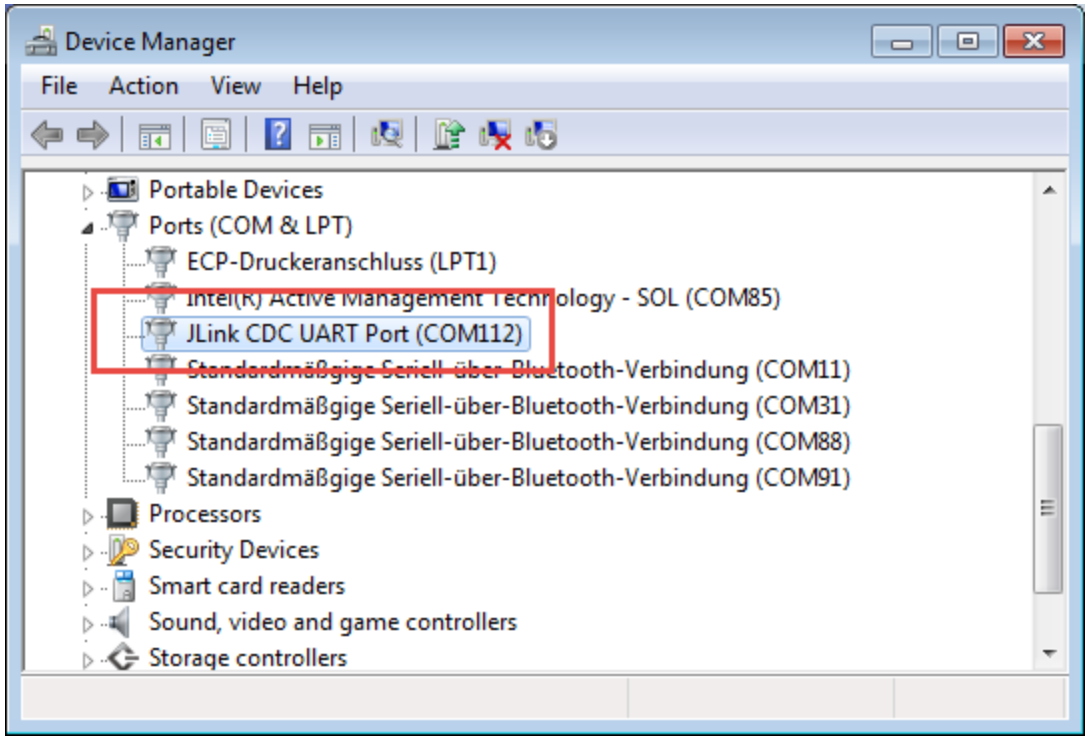
Launch FreeMASTER

- Shortcut on Desktop
- C:\Freescale\FreeMASTER 1.4\pcmaster.exe



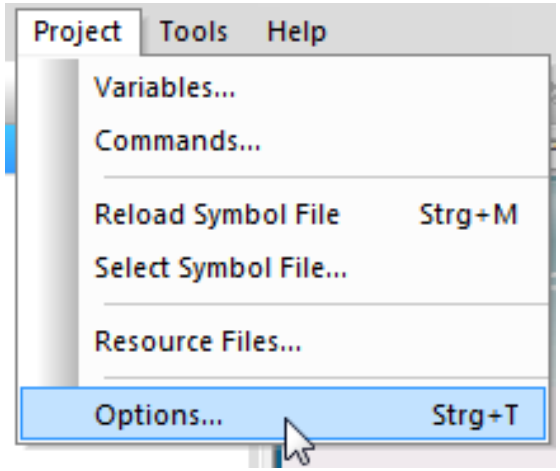
Serial Port

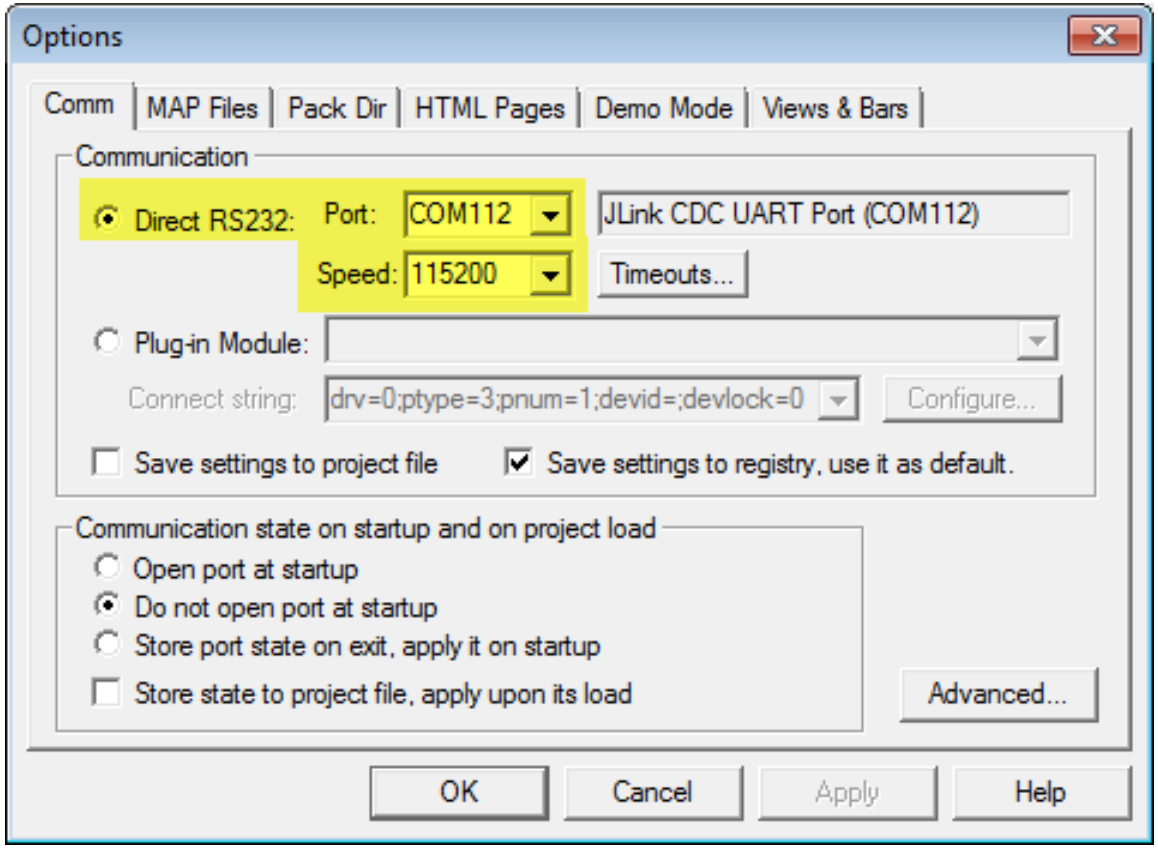
- FreeMASTER host application needs COM port
- Use Windows Device Manager
- Have board plugged in
- Determine the Serial Port



FreeMASTER: Communication Port

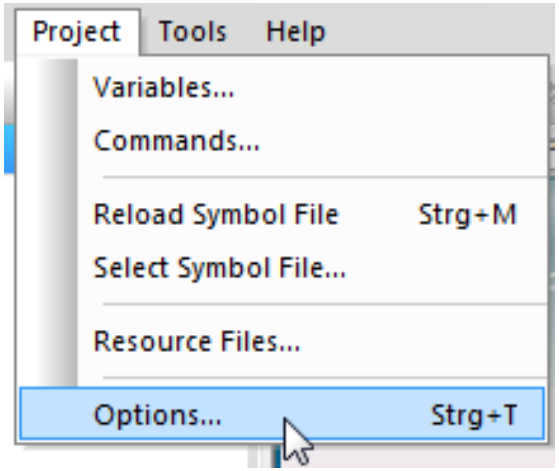
- Menu ***Project > Options…***
- ***Comm***Tab
- Select RS232 port
- 115200 baud

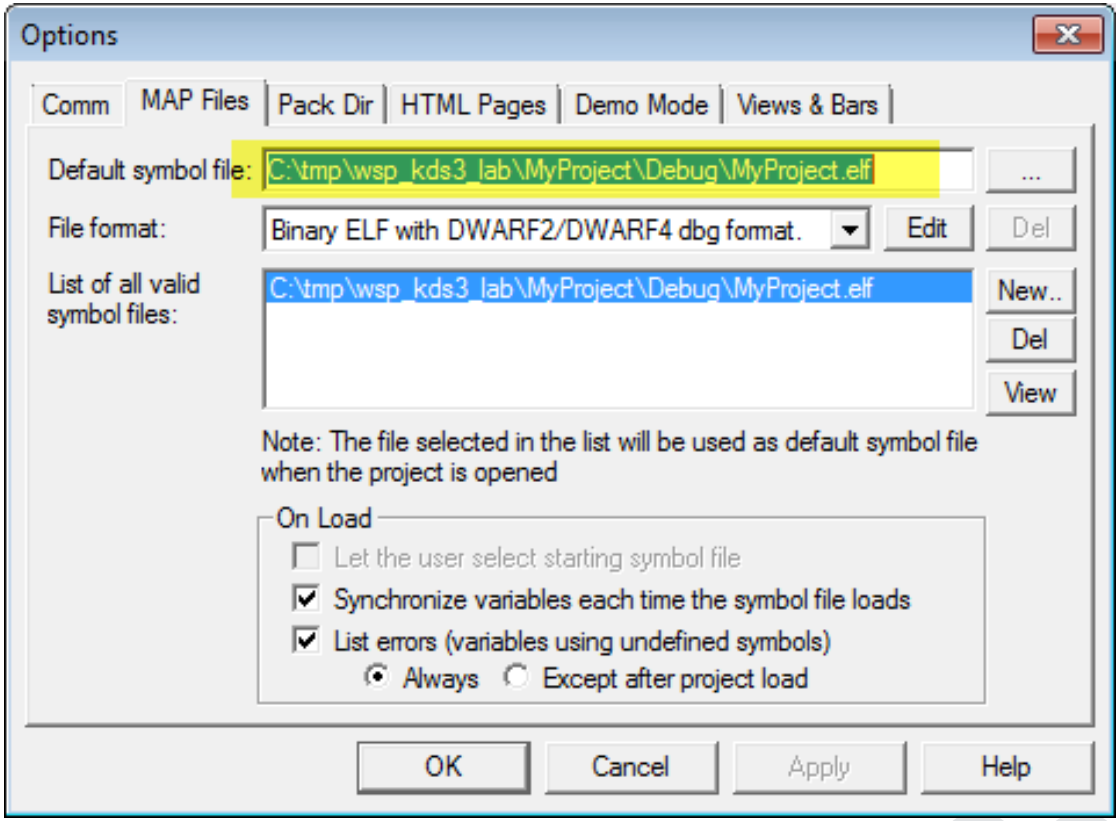




FreeMaster: Map File

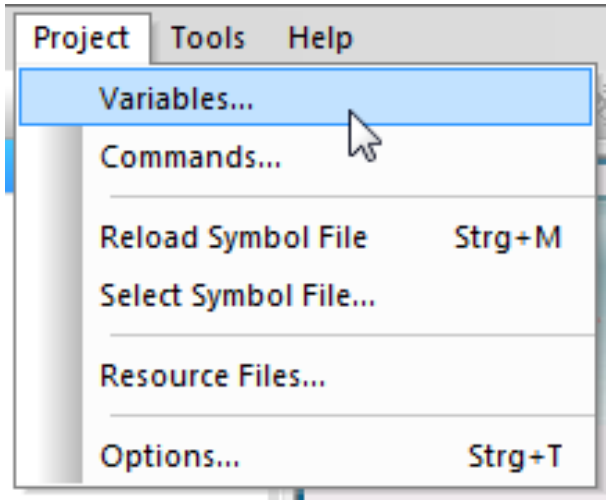
- Menu ***Project > Options…***
- ***MAP Files*** Tab
- Browse for ***symbol*** (ELF) file

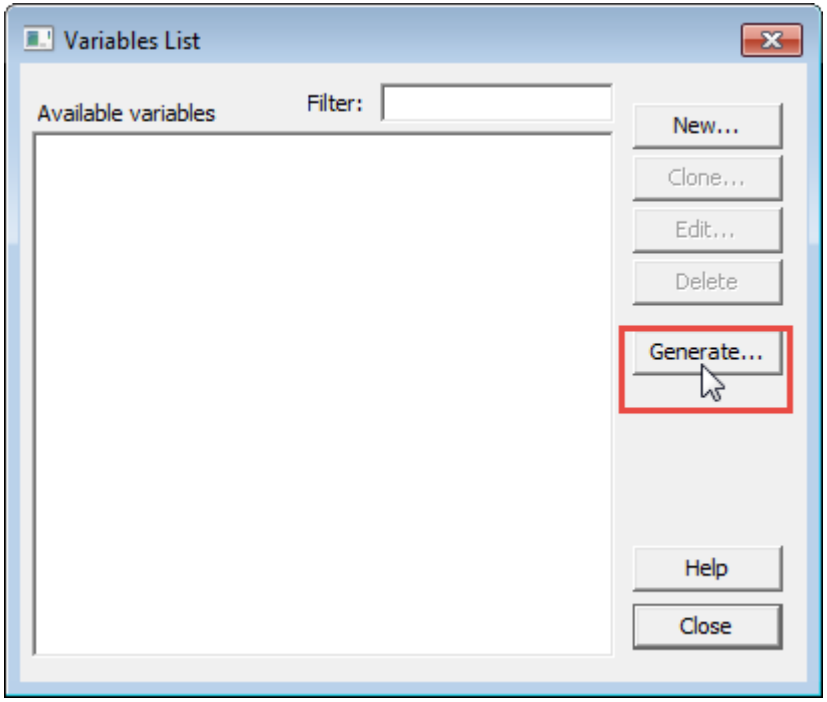




FreeMASTER: Adding Variables

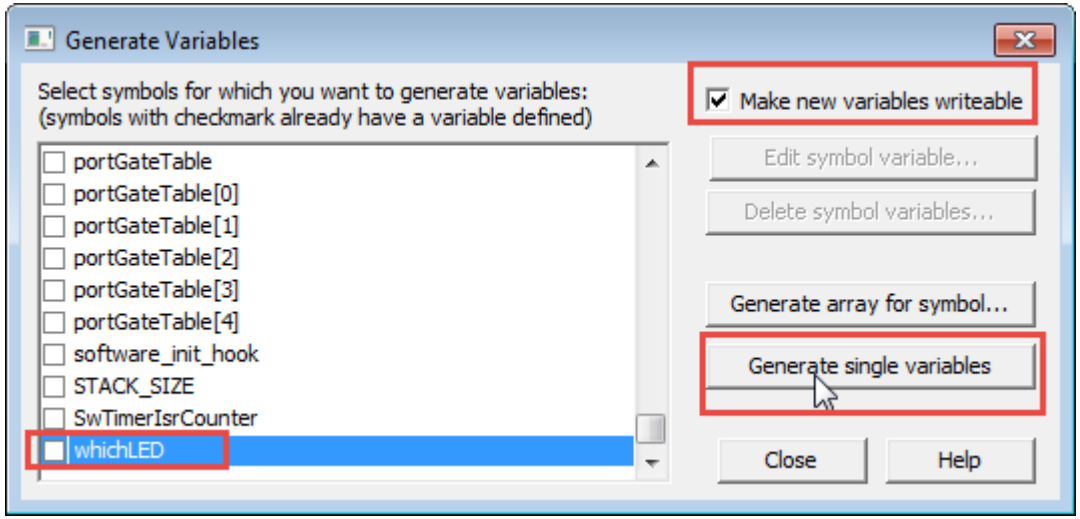
- Menu ***Project > Variables…***
- Use **Generate…** button

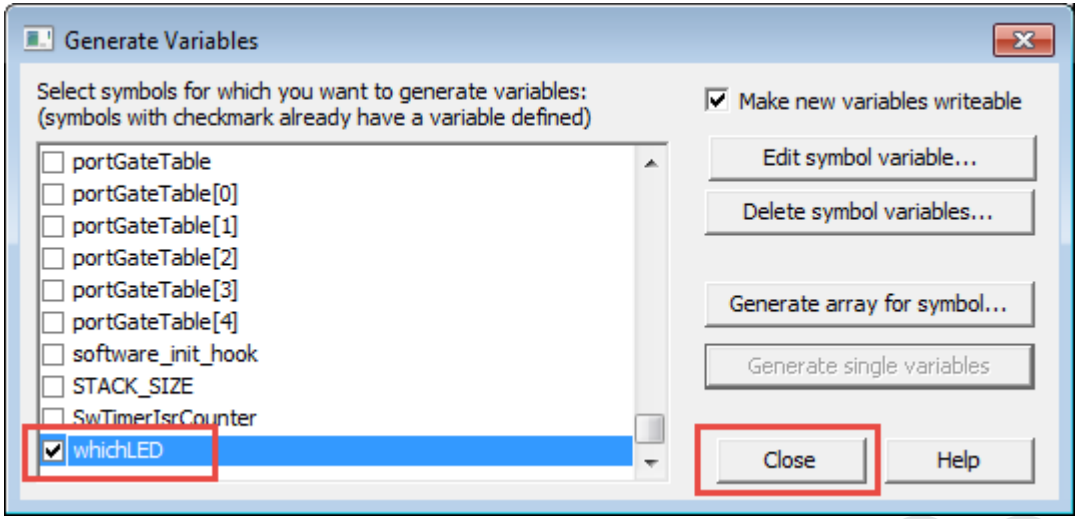




Adding Variable in FreeMASTER

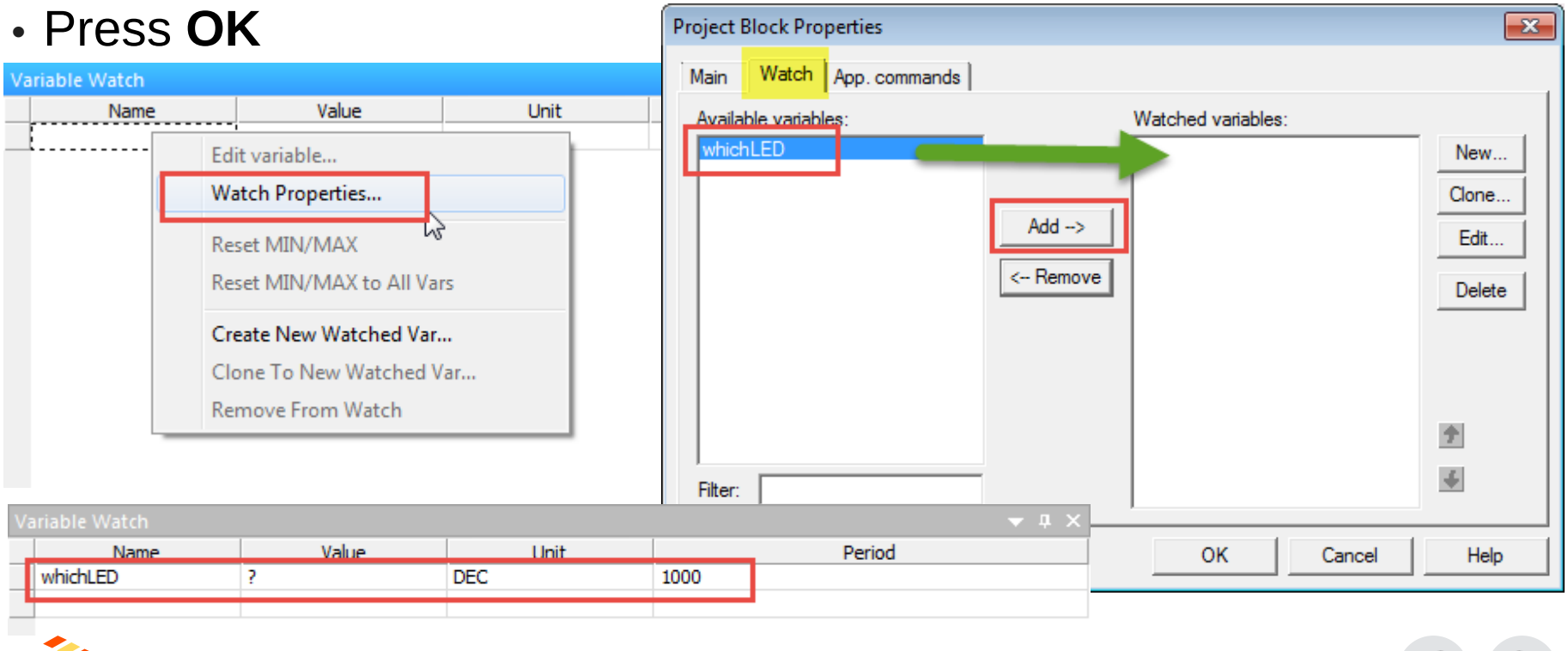
- Select variable
- Enable **Make new variables writeable**
- **Generate single variable**
- Variable gets checked
- **Close** dialogs





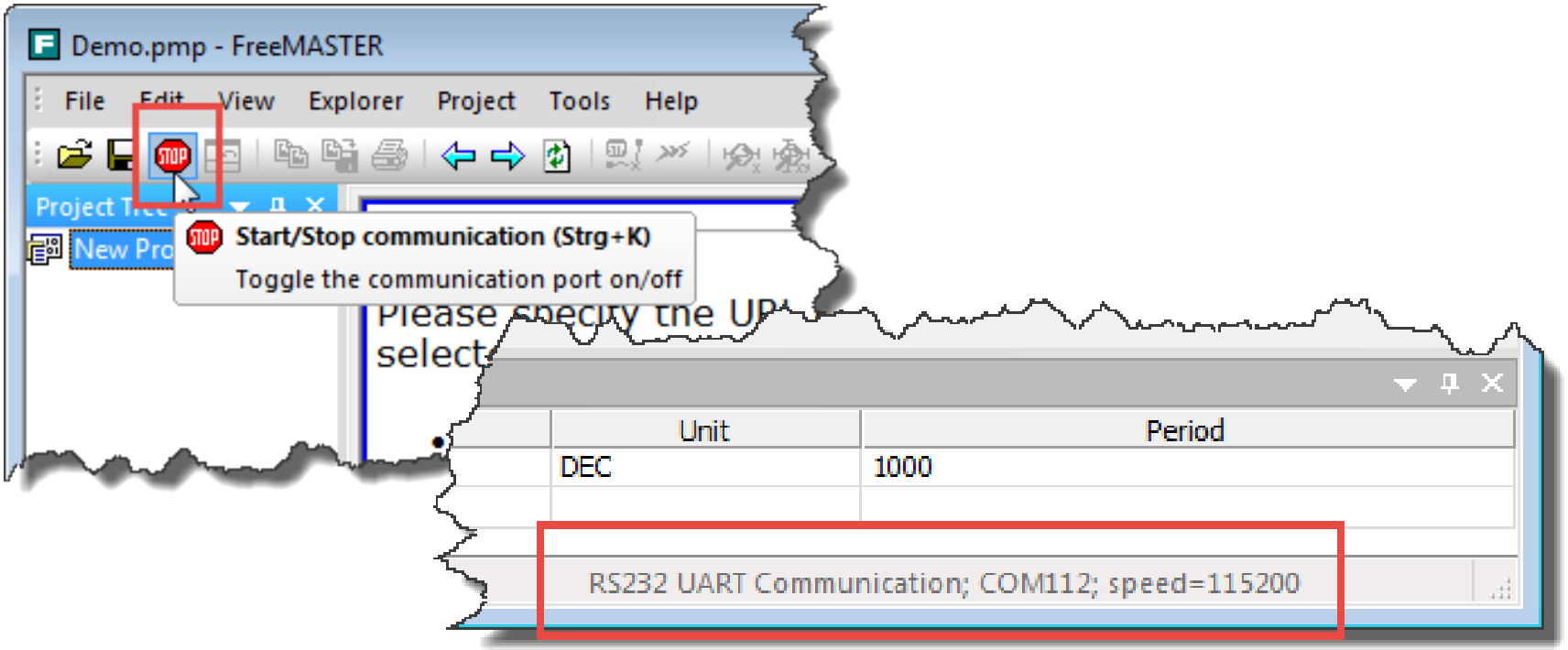
Adding New Watch Variable

- Right Click in 'Variable Watch'
- Select **Watch Properties..,**
- Select **Watch** tab
- **Add-->** variable to watched variables
- Press **OK**



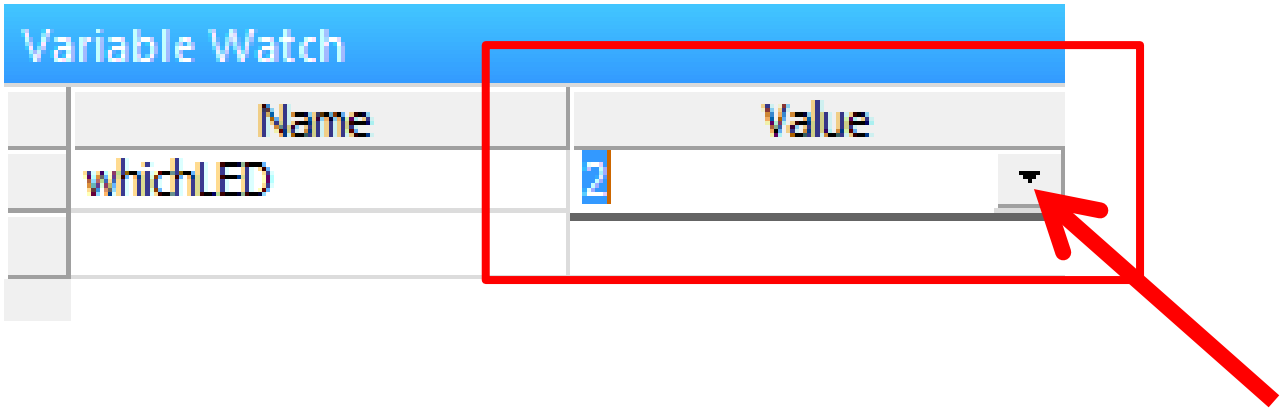
Start Communication

- Run application on Board
  - Application is listening on UART0/communication channel
- **Start** communication with Start/Stop icon
  - FreeMASTER is communicating with application on board



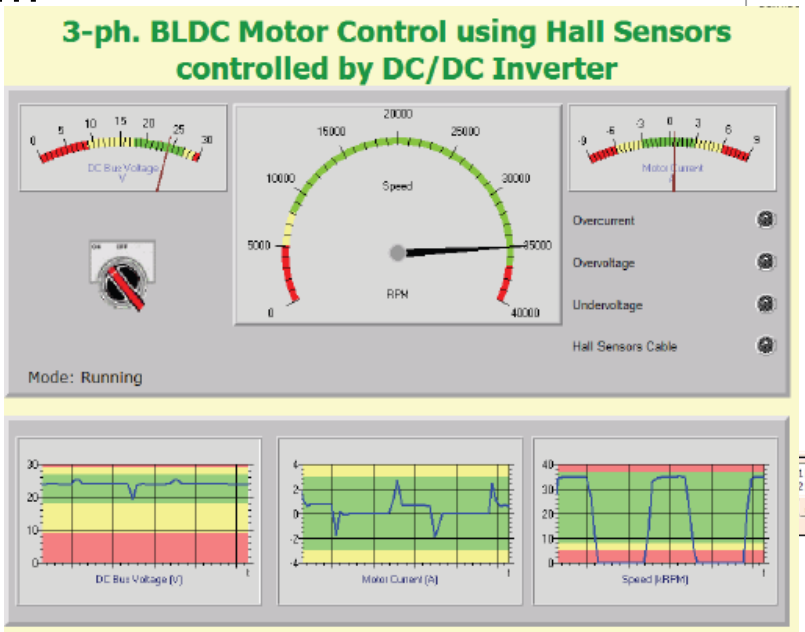
Changing LED with FreeMASTER

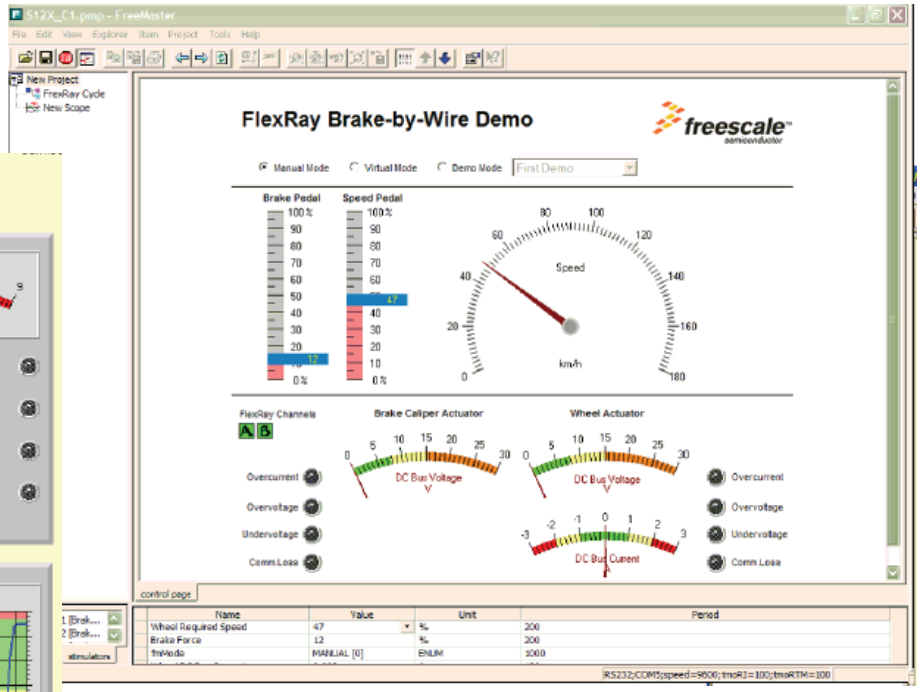
- Click on **'drop down**' icon
- Change variable and press ENTER
- Verify LED color change



Summary

- FreeMASTER used to monitor target application
- Uses Communication Channel (UART, CAN, ….)
- There is a LOT more FreeMaster can do for you
  - Charts/Graphs/Oscilloscope
  - Remote control
  - …





Congratulations!

- Installed KDS, KSDK, FreeMaster
- Created Project with Wizard
- Building, Debugging
- Adding and configuring SDK Drivers
- Configuration with Processor Expert components
- Realtime monitoring and changing of variables with FreeMASTER

Contact: erich.styger@freescale.com
Blog: http://mcuoneclipse.com/

www.Freescale.com

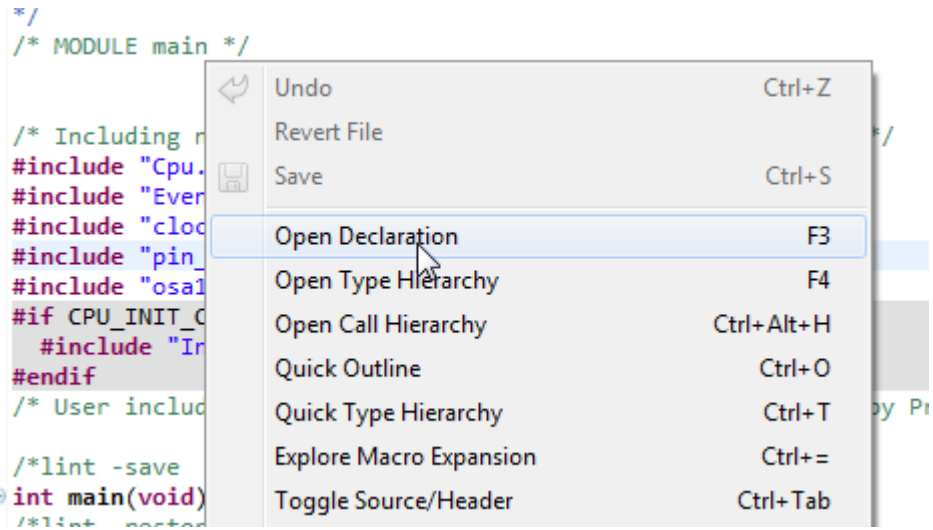
Backup

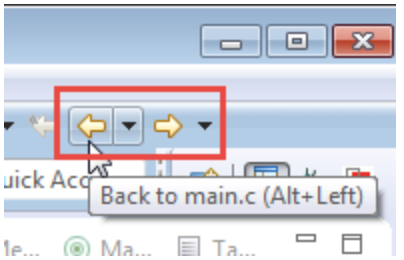
More Eclipse Tips ☺

Public

Tip: Navigation, Open Declaration/F3 + Alt+left/right

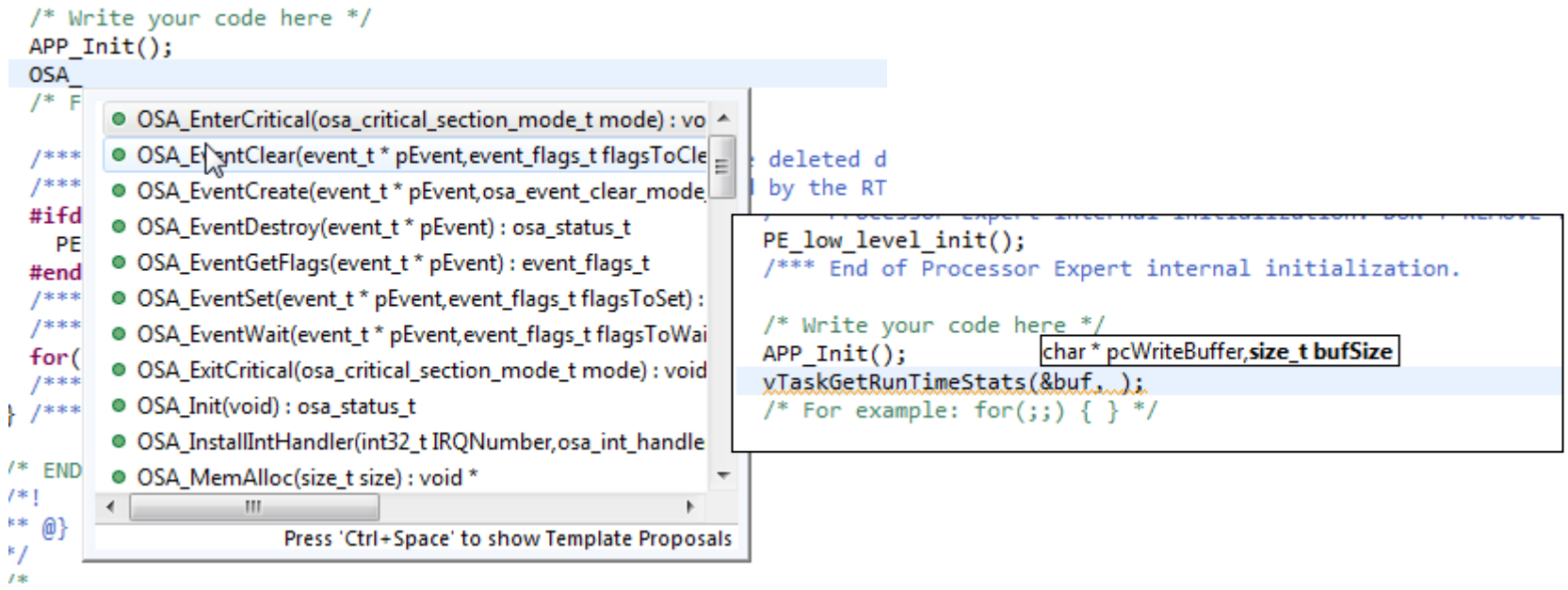
- Click into a variable, include file, etc
- Use 'Open Declaration' (or F3) ➔ Jumps to file/declaration
- Use <Alt>+<Left> and <Alt>+<Right> to navigate forward/backwards





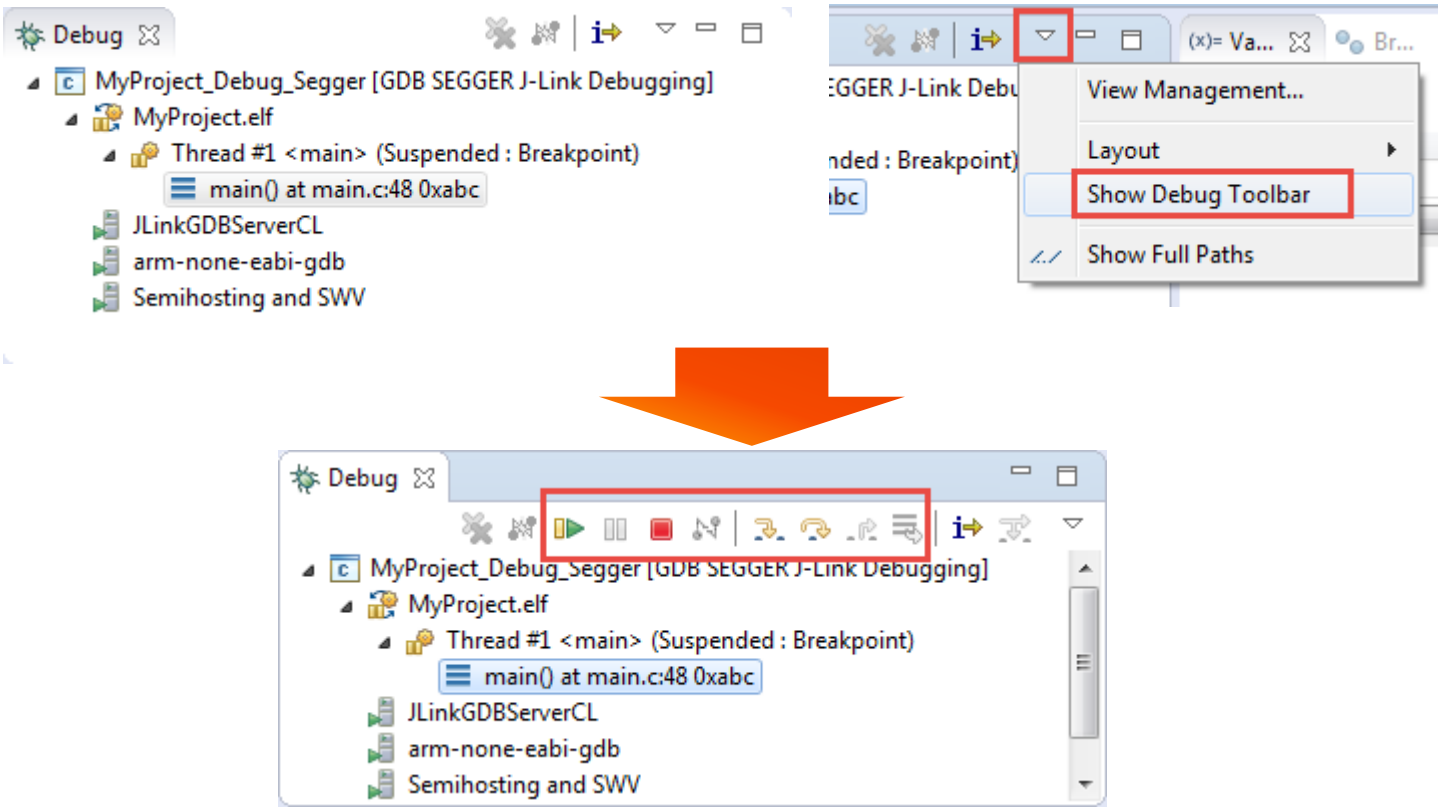
Tip: Auto-Completion <CTRL>+<Space>

- Start typing some function/variable/include/etc name
- Use <CTRL>+<Space> to get proposals
- Select proposal or proceed with typing
- Works for fields, pointers, parameters, include files, …



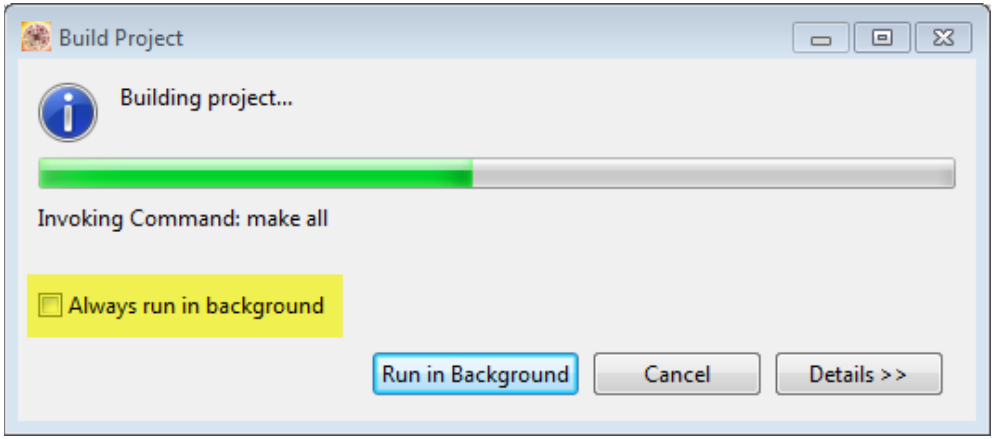
Tip: Enabling Debug Toolbar

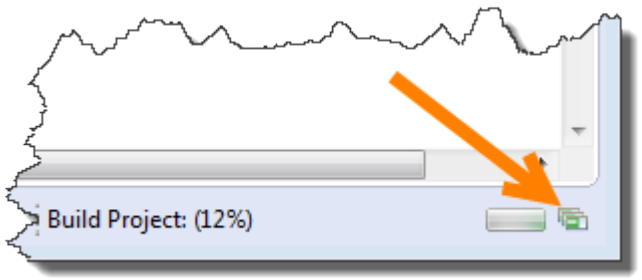
- Show Debug Toolbar in 'Debug' view

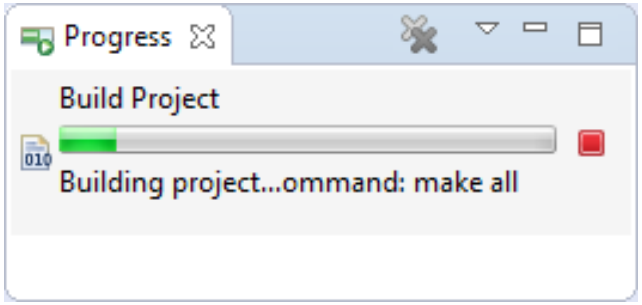


Tip: Running in Background, Progress View

- You might want to run build in the **Background**
- Double click on 'progress' icon to show **Progress** view

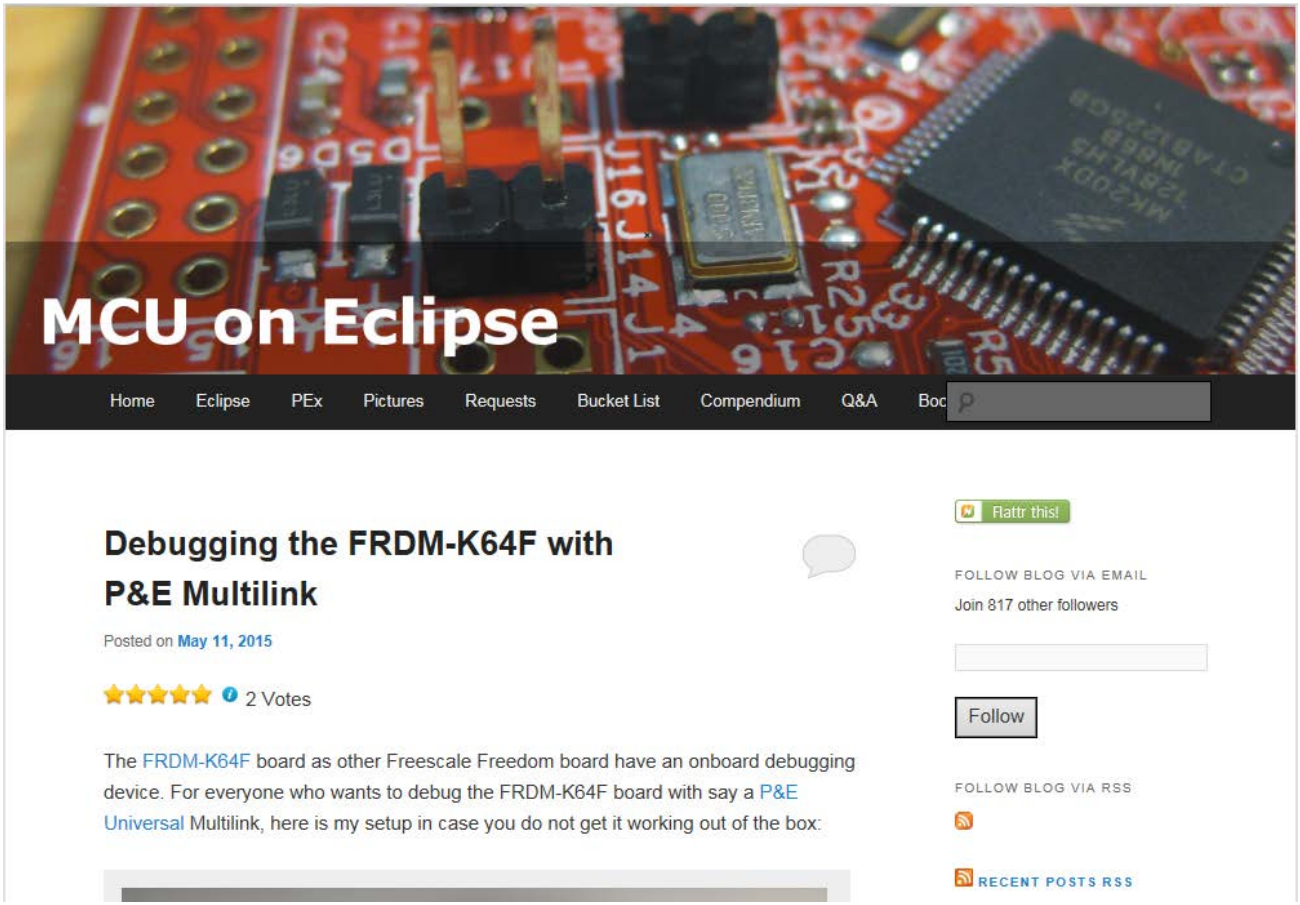






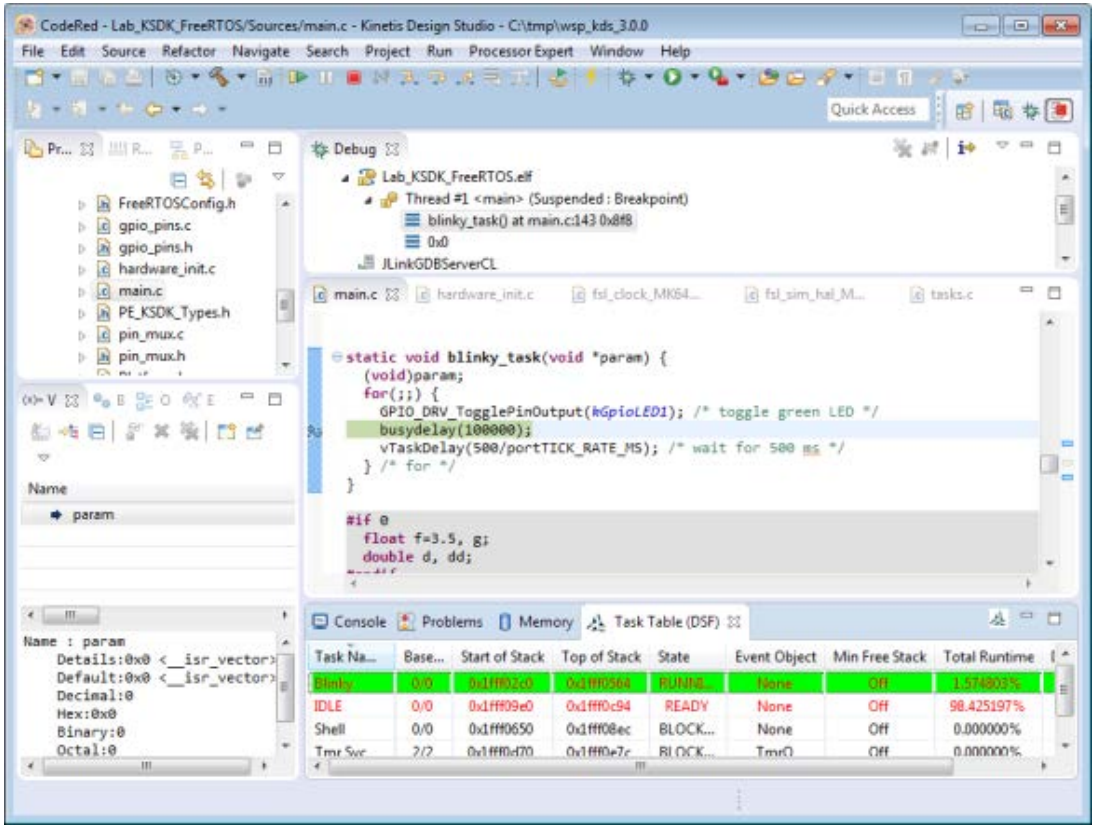
Tip: McuOnEclipse

- Follow my blog ☺:
  - http://mcuoneclipse.com/



Tip: Using 'combined' Debug Perspective

- One perspective ideal for Editing **and** Debugging
- http://mcuoneclipse.com/2015/05/10/codered-debug-perspective-in-kinetis-design-studio/

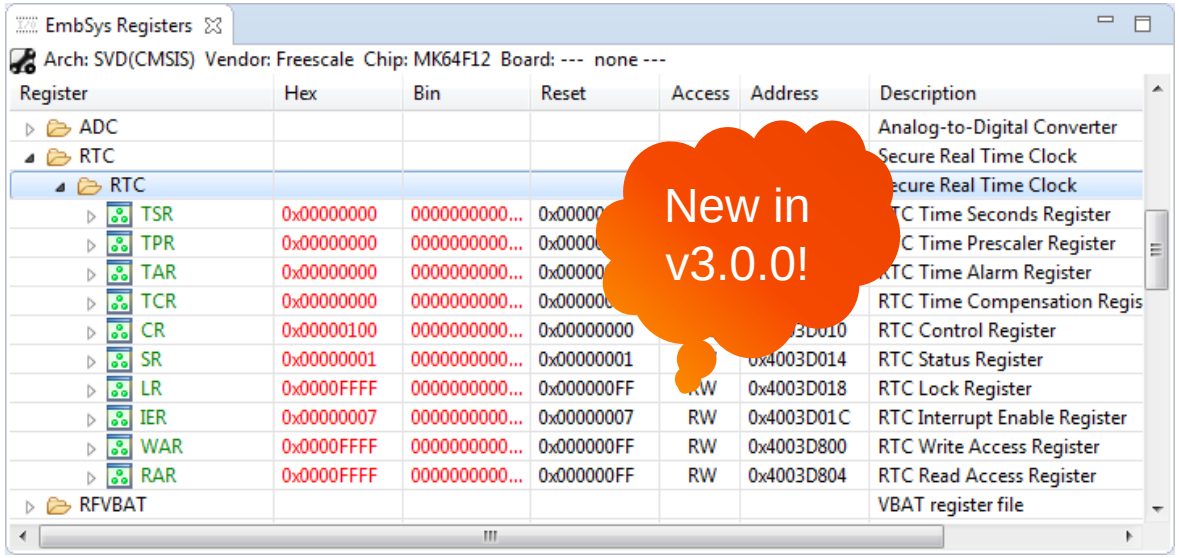


Register Details View

Public

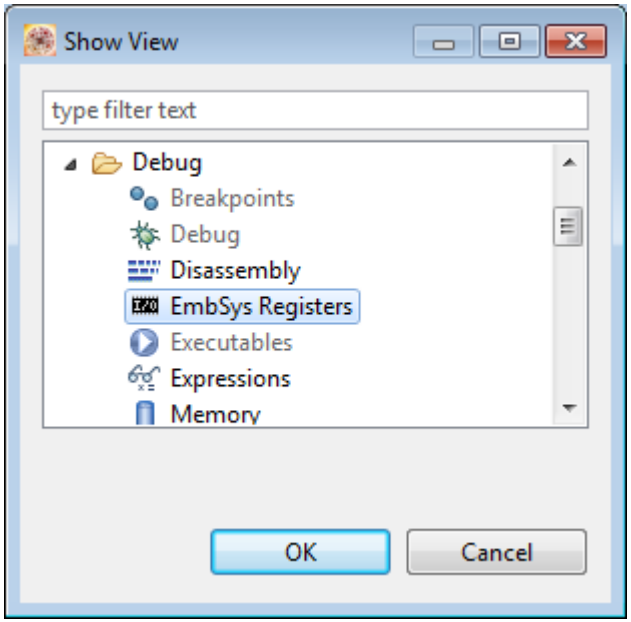
Outline

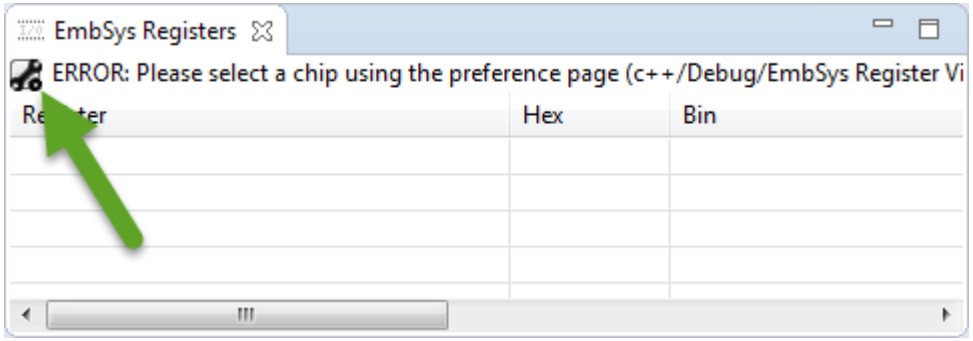
- Ability to inspect/modify device registers
  - Biggest wish for KDS v2.0.0
- Open Source Plugin EmbSysRegView
  - http://embsysregview.sourceforge.net/
- Extended CMSIS-SVD files with core registers



Opening the EmbSysReg View

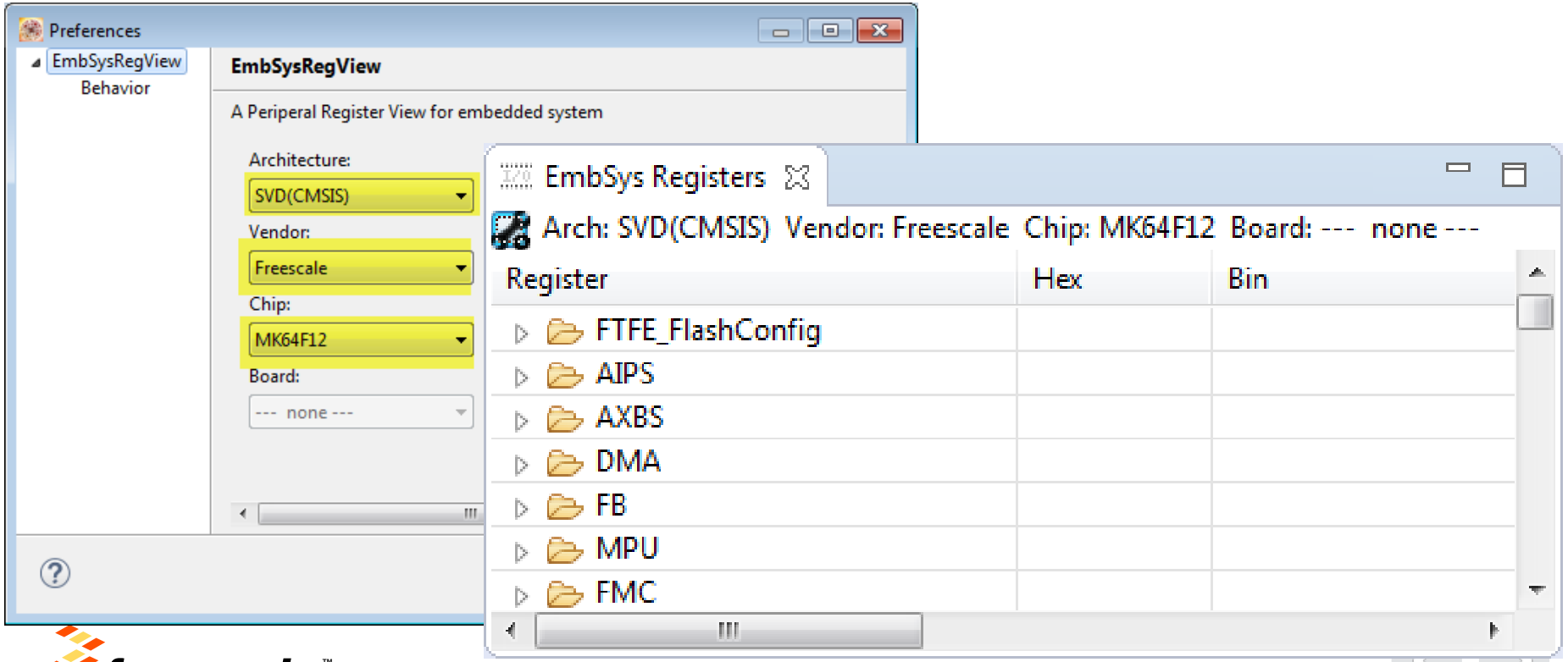
- Menu Window > Show View > Other Debug > EmbSys Registers
- Click on wrench icon to configure it





Device Configuration

- Select SVD(CMSIS), Freescale and Chip device
- Workspace setting (not by project!)
- Menu **Window > Preferences > C/C++ > Debug > EmbSys Register View**



Reading/Changing Registers

- Double click enable/disable the register or group
  - Color turns green
- Inspect values, change, …