FTF 2016
TECHNOLOGY FORUM

# MASTER DEVELOPMENT ON THE LPCXPresso TOOLCHAIN

## WITH THE LPCOpen DEVELOPMENT KIT

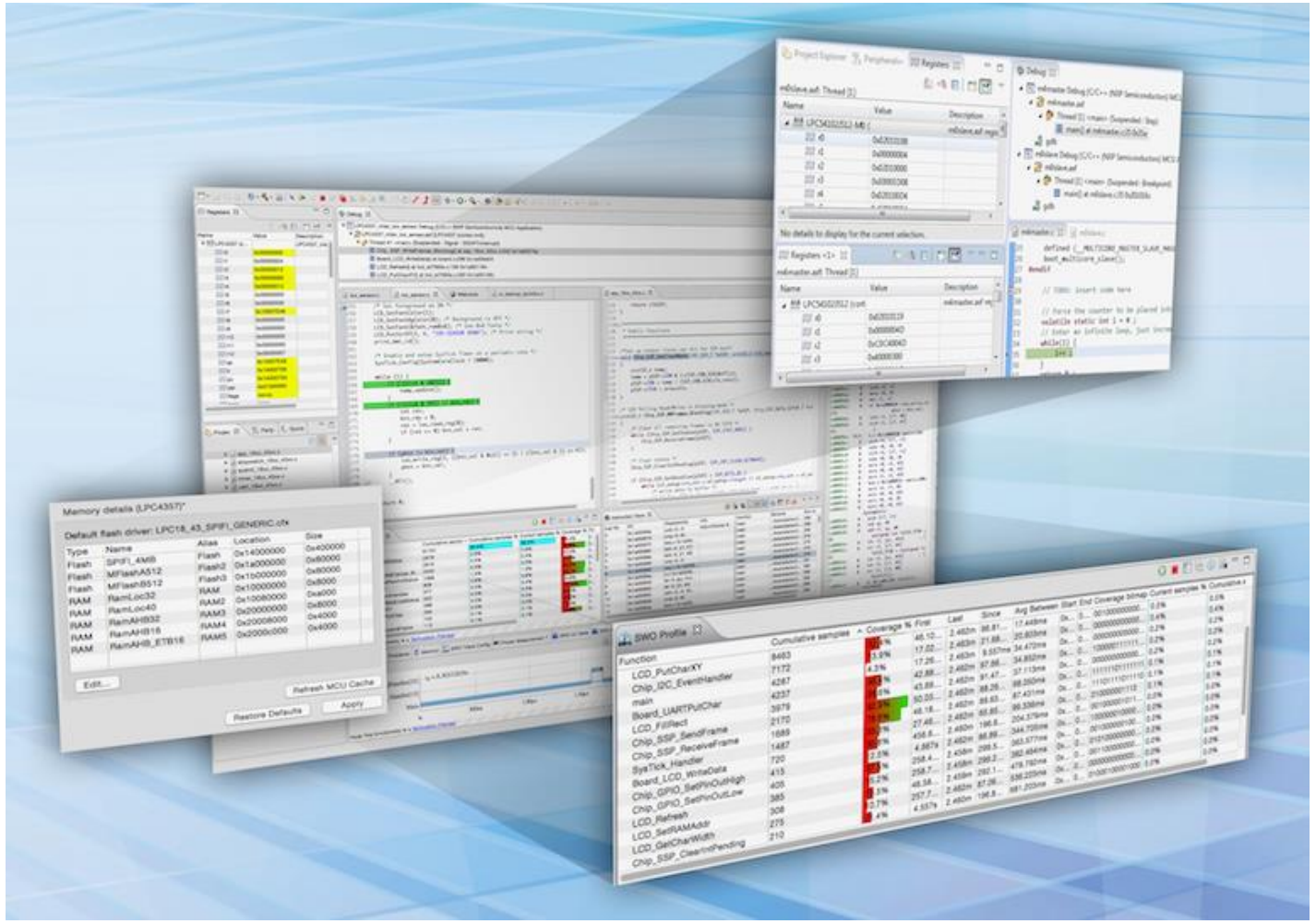
ANDY BEESON
LPC SOFTWARE TOOLS MANAGER
FTF-DES-N1973
MAY 17, 2016

# AGENGA

- LPCXpresso and LPCOpen Overview
- Getting Started
- Launching a debug session
- Creating and editing projects
- Multicore Support
- Instruction Trace
- SWO Trace
- Power Measurement

# What is LPCXpresso IDE

- Enhanced Eclipse Mars + GCC5 based IDE
  - Cross Platform:
    - Windows
    - Mac OS X
    - Linux – recent Ubuntu and Fedora
  - Single installer contains everything needed
  - Focused on ease of use
- Free Edition
  - 256KB download limit
  - Simple registration at LPCware.com
  - Forum support
- Pro Edition - US $495
  - No download limit
  - Additional SWO Trace functionality
  - Professional customer support
- Many tens of thousands of users
  - Increasing popularity at large OEMs



# Latest Release : LPCXpresso IDE v8.1.4

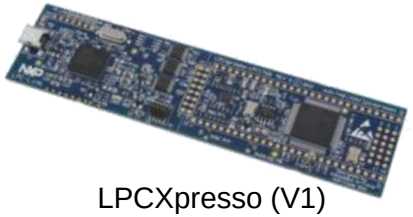
- Major highlights over last year include:
  - New parts support
  - Updated versions of Eclipse and GCC
  - Flash programming performance (Typically ~10x improvement since 7.5.0)
  - Multiple flash driver support
  - SWO Trace using LPC-Link2
  - Power Measurement functionality
  - Support for multiple simultaneous debug probe connections
    - Several targets can be connected to same host
  - Enhanced managed linker scripts and templates (Freemarker)
- More details at:
  - http://www.lpcware.com/content/forum/lpcxpresso-latest-release

# More Information

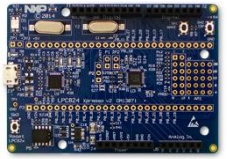
- General Product Information
  - http://www.nxp.com/pages/:LPCXPRESSO
- Where to download
  - http://www.lpcware.com/lpcxpresso/download
- Docs
  - Within product : Built-in help system and PDFs
- FAQs
  - http://www.lpcware.com/faq/lpcxpresso
- Videos
  - https://www.youtube.com/watch?v=NW7GmsMcrKc

# LPCXpresso Boards

- Concept pioneered by NXP
  - Low cost, easy to use development platform with flexible expansion options
- Works with all partner tool-chains (except V1) to provide a low-cost evaluation/development platform
- Features
  - LPC Cortex-M processor with on-board debug probe (OBD)
  - Connector for external debug probe (except V1)
  - OBD probe can debug external target (except MAX / CD)
  - MAX, V2 and V3 boards offer popular Arduino R3 expansion
  - CMSIS-DAP and SEGGER J-link debug protocol support on V2 and V3 boards via on-board LPC-Link2
  - Power measurement circuitry for LPC target and shield board on many V3 boards


LPCXpresso (V1)


LPCXpresso CD

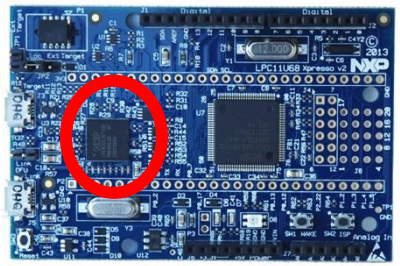

LPCXpresso MAX

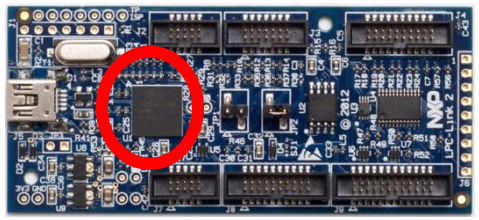

LPCXpresso V3


LPCXpresso V2

# LPC-Link2 Debug Probe

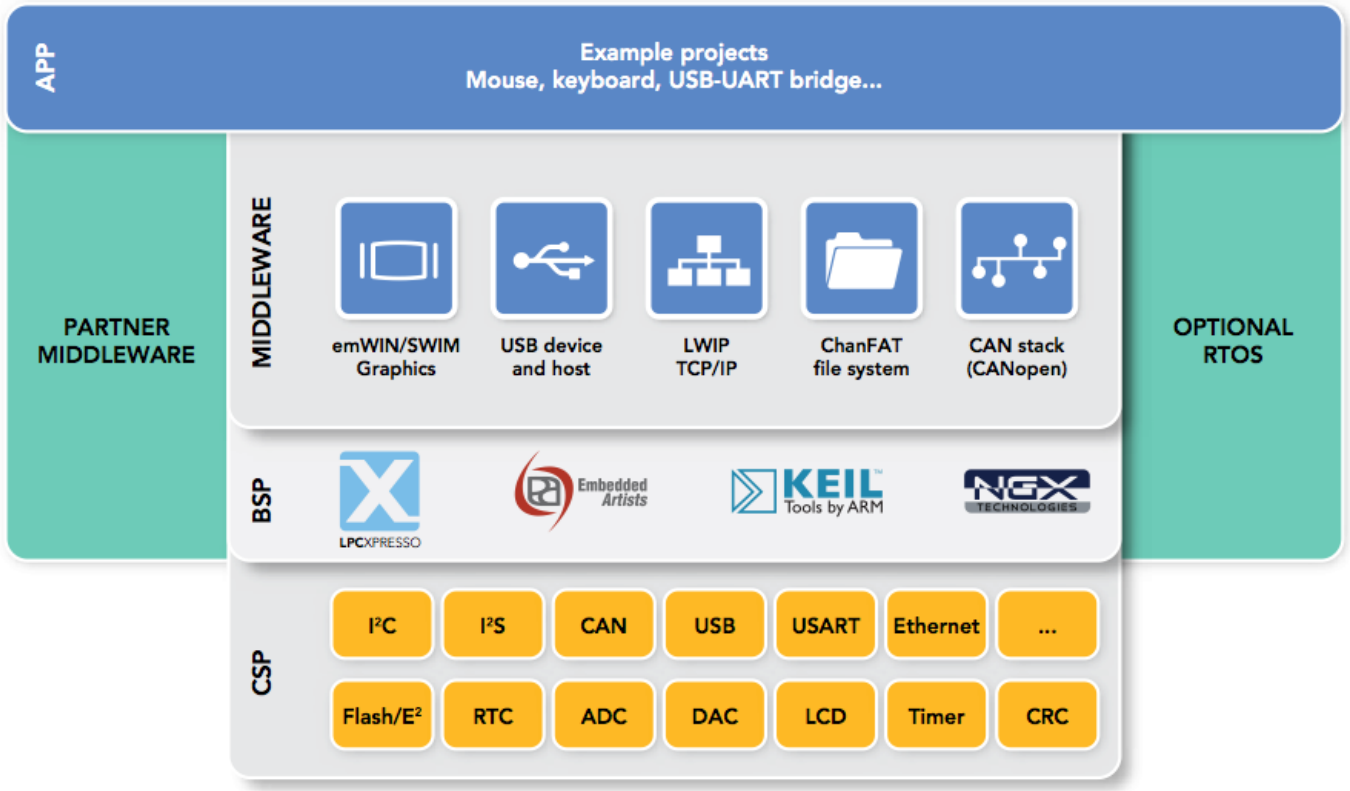
- LPC-Link2 is the LPC43xx-based debug probe technology used as the basis of the :
  1. LPC-Link2 standalone debug probe
     - Uses an LPC4370 +SPIFI device and is also an evaluation board for that MCU
  2. On-board debug (OBD) probe on the LPCXpresso V2/V3 boards
     - Uses LPC4322 (or similar) with internal flash
     - Provides additional functionality – e.g. VCOM
- Supports use of partner toolchains (e.g. Keil, IAR, LPCXpresso IDE, Atollic, Rowley) via different firmware images
  - CMSIS-DAP :
    - LPCXpresso IDE can soft-load using DFU boot (no flashing needed)
    - For other tools, program into flash on probe
  - Segger J-Link : Program into flash on probe.
    - Note: limited version, restricted to evaluation use only
- Firmware programmed by LPCScrypt tool





# LPCOpen

- Extensive array of software drivers and libraries
- MCU peripheral device drivers with meaningful examples
- Common APIs across device families
- Thoroughly tested and maintained
- Commonly needed third party and open source software ports
- Keil, IAR and LPCXpresso projects
- http://www.lpcware.com/lpcopen

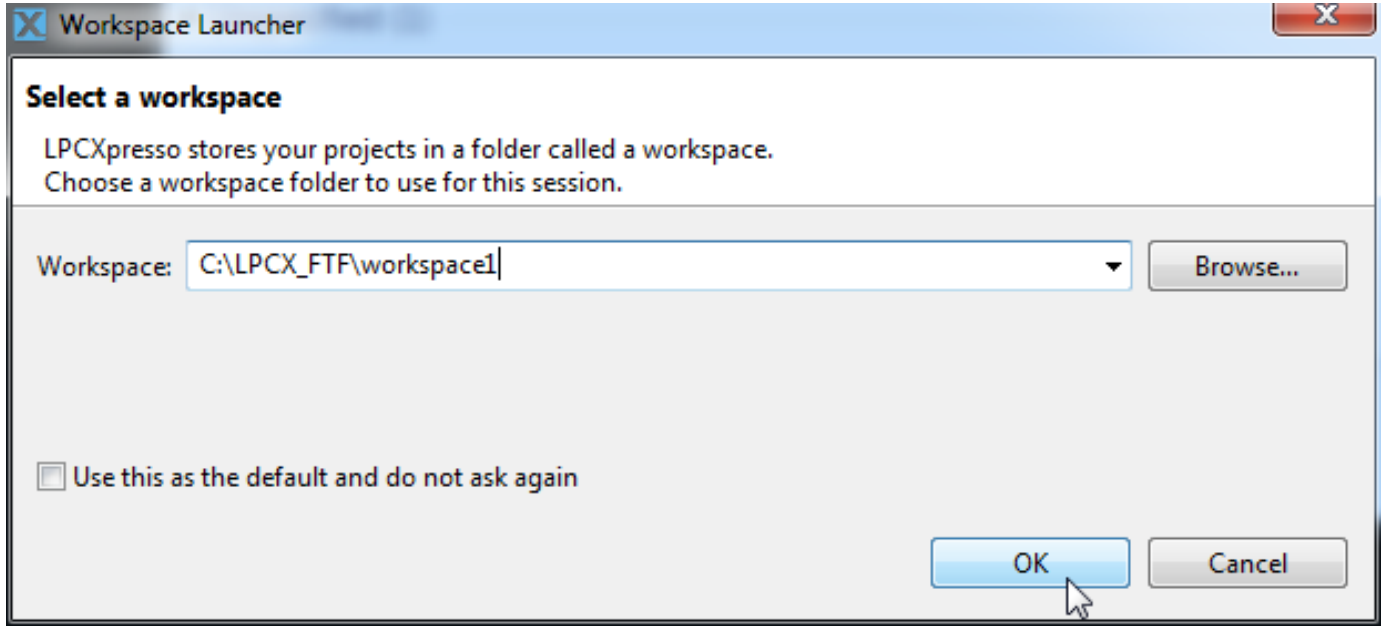



```
int main(void)
{
    SystemCoreClockUpdate();
    Board_Init();

    Board_LED_Set(0, false);

    /* Enable SysTick Timer */
    SysTick_Config(SystemCoreClock / TICKRATE_HZ);

    /* Bail out after timeout */
    while (sysTick <= TIMEOUT_TICKS) {
        __WFI();
    }
```

# GETTING STARTED

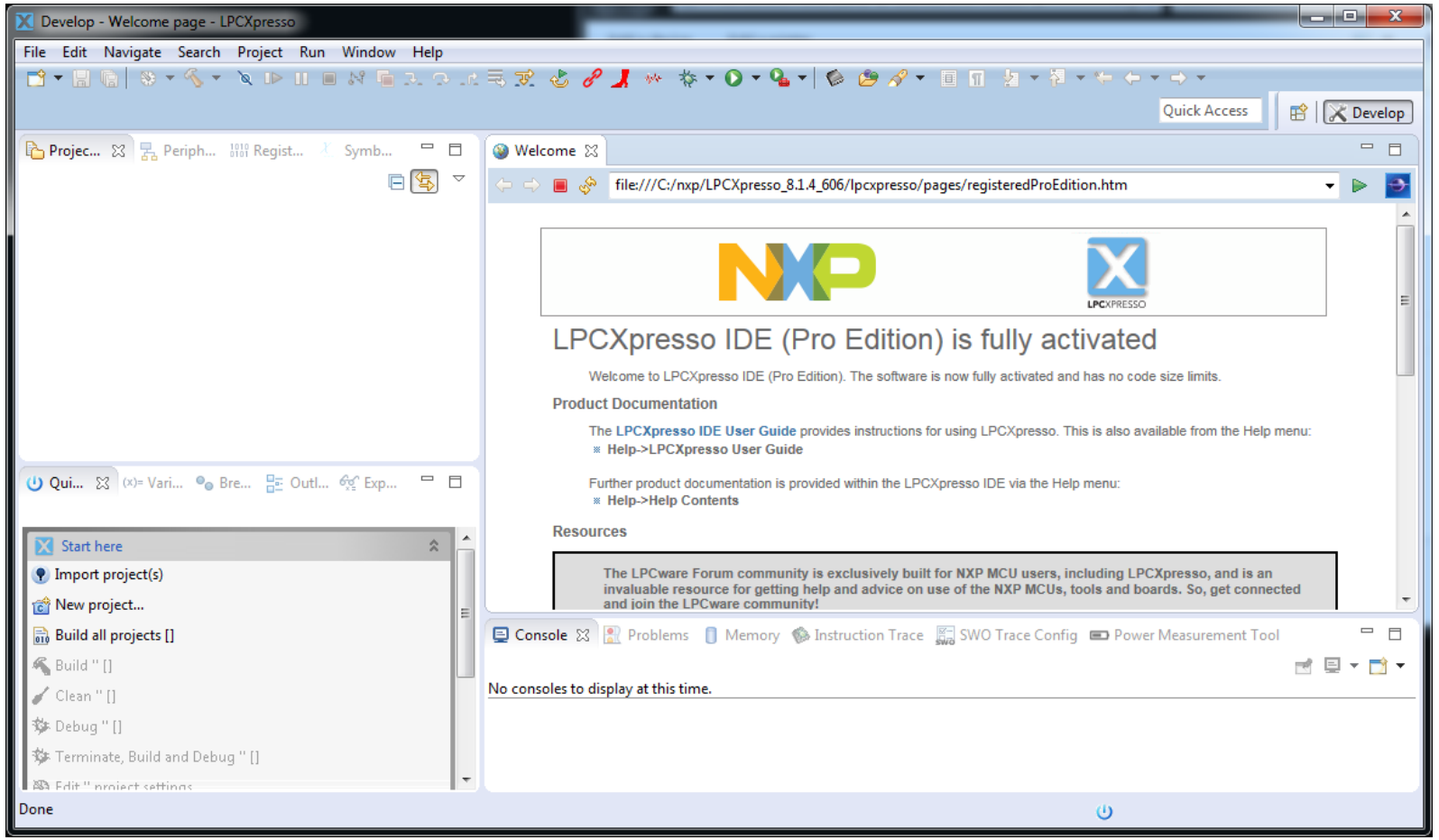
# Start LPCXpresso IDE v8.1.4 – New Workspace

- Start LPCXpresso IDE on your system
- At the dialog box, enter a location for your workspace then click OK
  - Suggest C:\LPCX_FTF\workspace1
- Note: A workspace is just a folder containing the projects that you want to actively work on during this IDE session



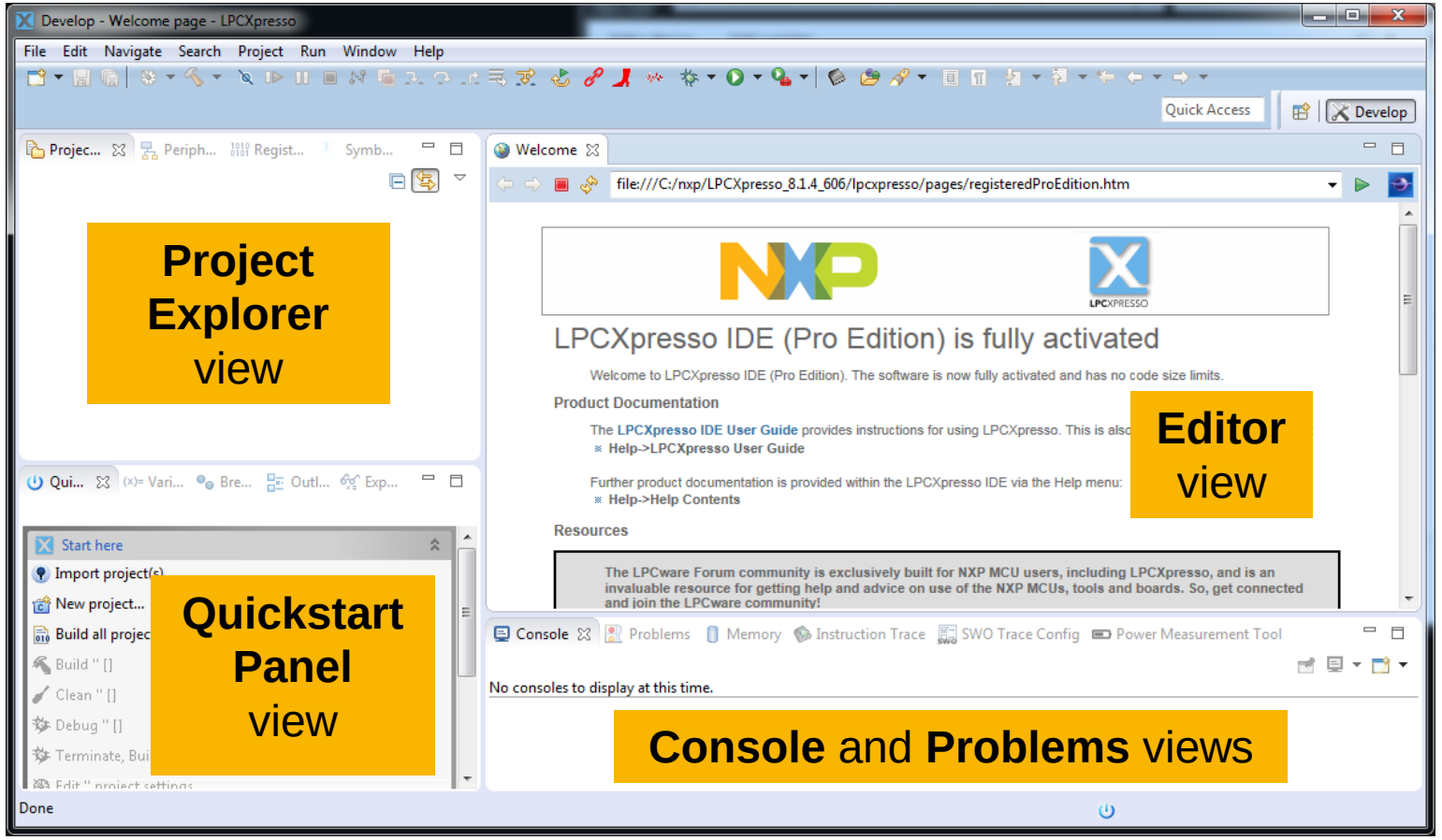
# New Workspace after Creation

- LPCXpresso IDE will startup in your new empty workspace with no initial projects
- Note that Welcome Page shows that IDE has been activated (in this a case with a Pro Edition License)
- With a fresh install, use *Help -> Activate* to install license



# Develop Perspective

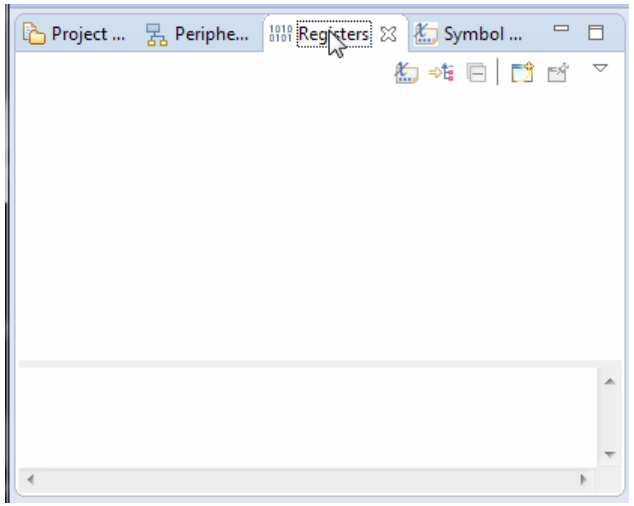
- A “perspective” is a collection of different “views”
- The Develop Perspective was created to provide a single combined Project Management and Debugging view
- In addition to the default Develop perspective, the LPCXpresso IDE also supports traditional Eclipse C/C++ and Debug perspectives

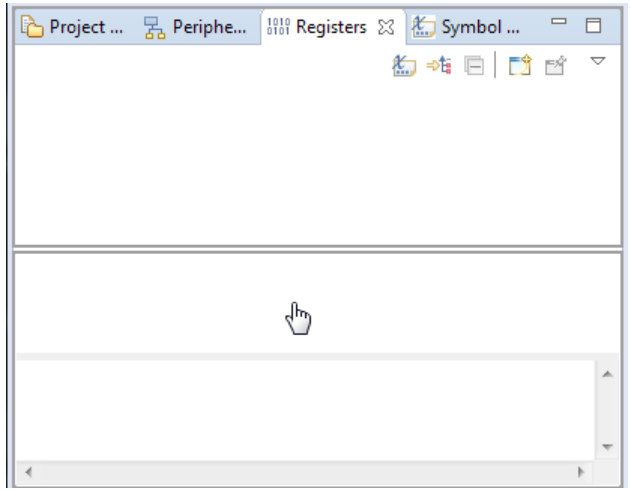


# Changing the Layout of the Develop Perspective

- Layout of views within a perspective can be tailored to meet your personal needs
- For example, if we wanted to have the Registers view always visible…

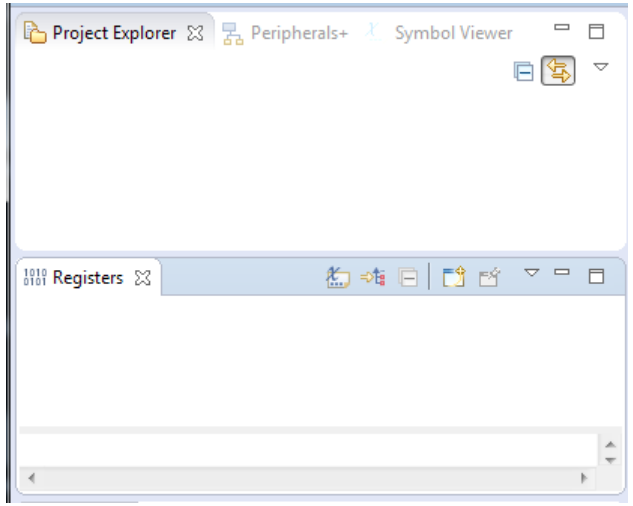
**Click and hold down on the View you want to move**

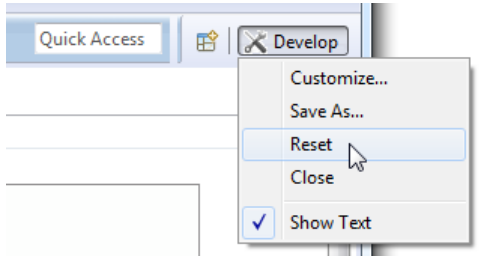



**Continue to hold down and drag the cursor to the location you want to view to be displayed**

**Then release the mouse click, and the view will be placed at the required position**

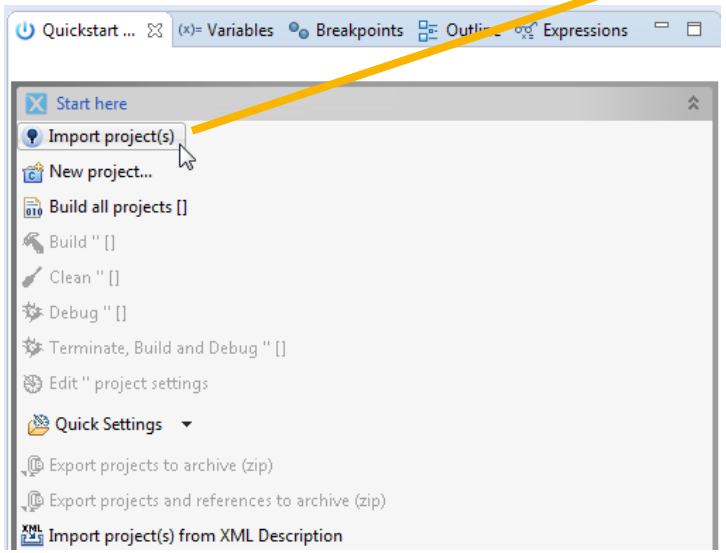


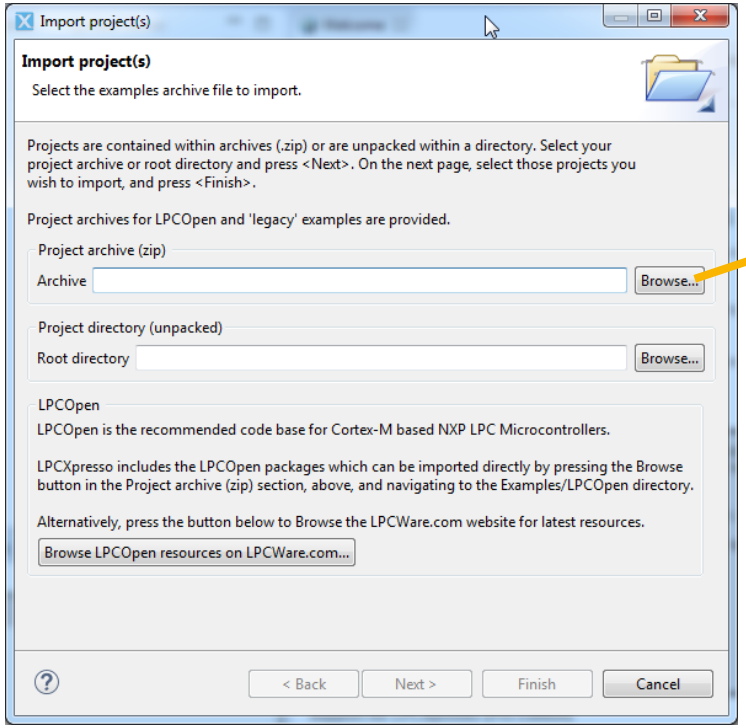



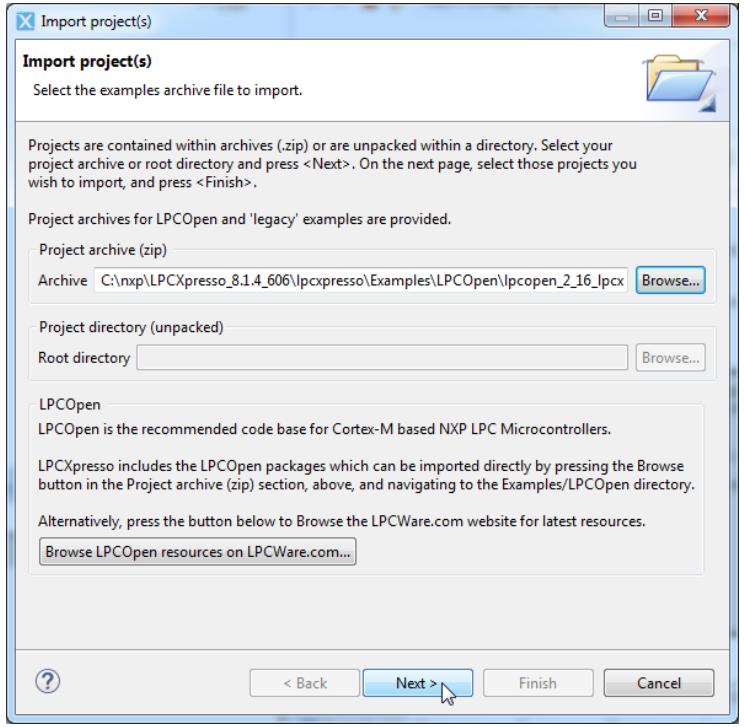

**Right click on the Perspective button (top right of IDE window) to reset the layout back to the default**

# Import Some Projects

- Quickstart -> Import project(s) -> Project Archive, select ZIP, then click Next
  - `C:\nxp\LPCXpresso_8.1.4_606\lpcxpresso\Examples\LPCOpen`
    - `lpcopen_2_16_lpcxpresso_nxp_lpcxpresso_4337.zip`

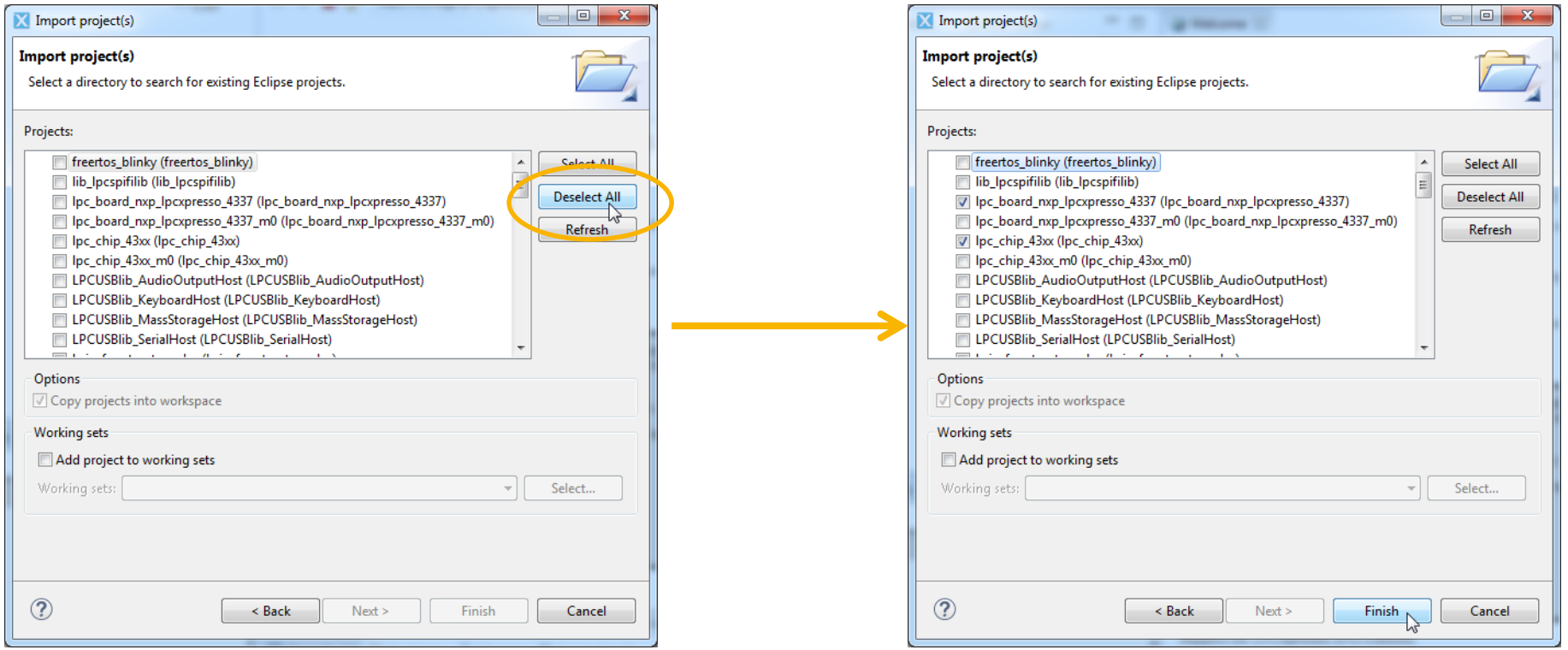


# Select Projects to Import

- Use the “Deselect All” option, then explicitly select required projects from scrollable list, then click “Finish”
  - `lpc_board_nxp_lpcxpresso_4337` and `lpc_chip_43xx`
  - `periph_timers`




Note: Don’t import the board and chip libraries for the M0 (with”_m0” postfix)

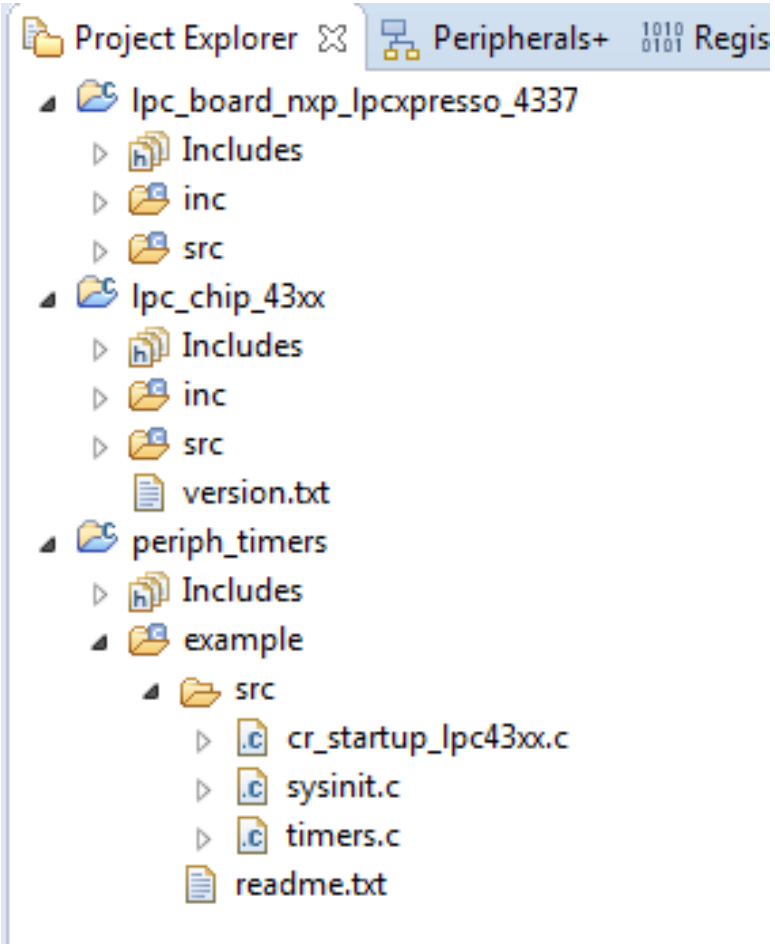
# Project Explorer View after Import

- Following the import, you should see three projects inside the Project Explorer View…

- The board library : `lpc_board_nxp_lpcxpresso_4337`
  - contains library headers and code for features of the board

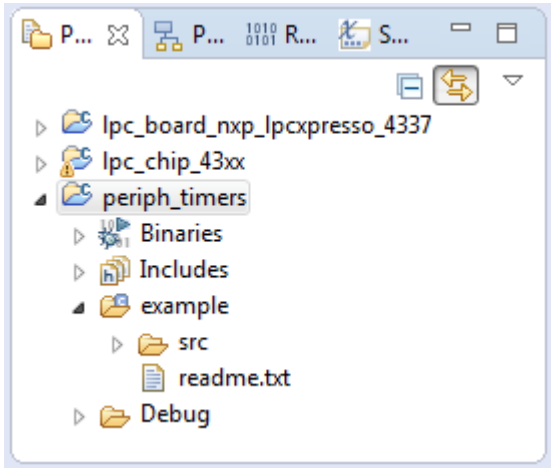
- The chip library : `lpc_chip_43xx`
  - contains library headers and code for the features of the MCU

- The application project : `periph_timers`
  - contains the example application code

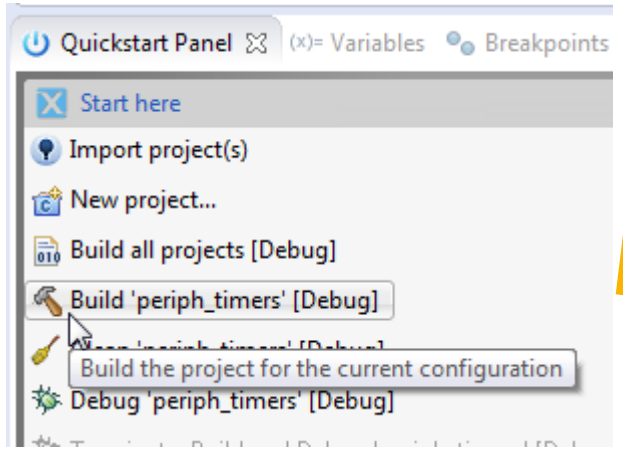


# Building the Application Project

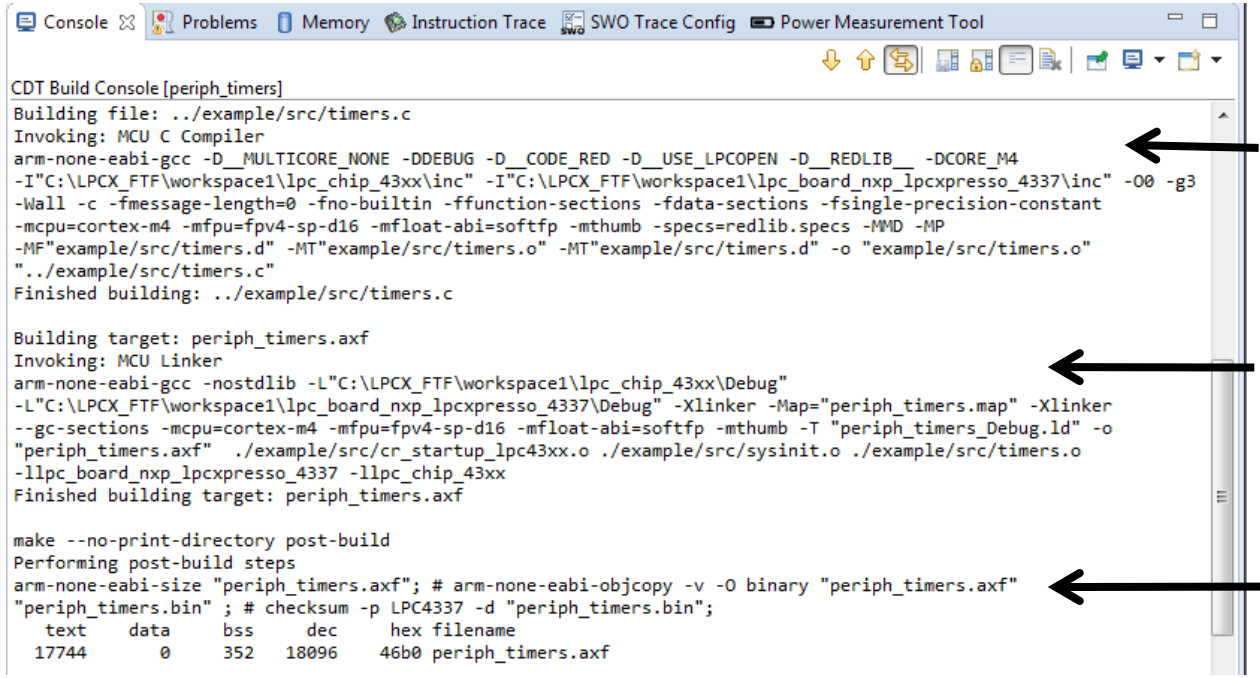
**Select Project**




**Build it**




**Check the build log**



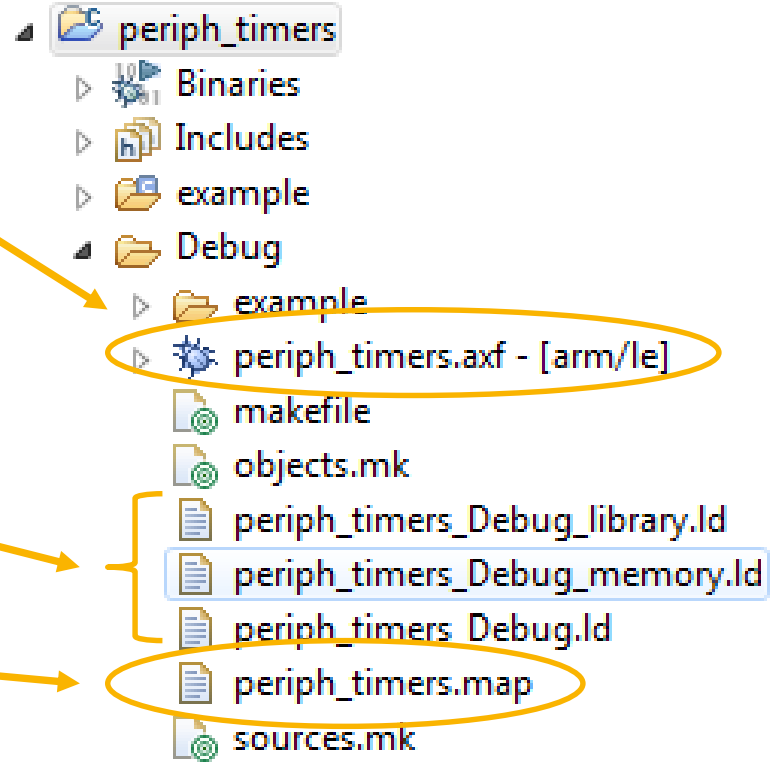

Compile step

Link step

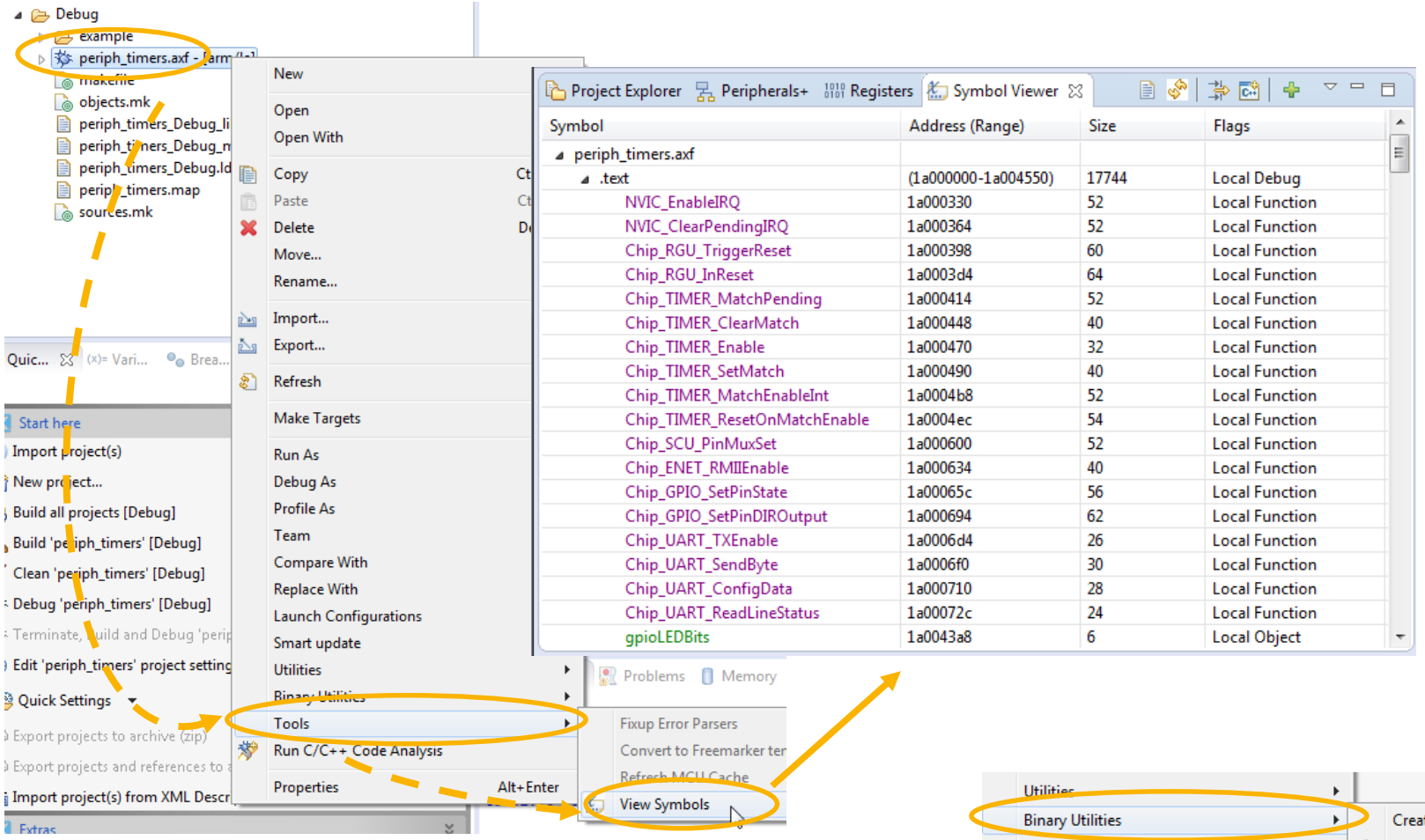
Post build step [size of image]

# Generated Image

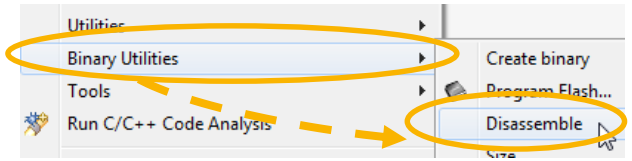
- Link step will generate an AXF file
  - Standard ARM Executable Format – ELF/DWARF
  - LPCXpresso IDE can directly download to target
  - Post build step can be used to convert to other formats, such as binary or hex (using arm-none-eabi-objcopy)
- Linker scripts, controlling placement of code and data in memory, generated automatically by IDE
- MAP file generated by linker can be very useful too
  - Shows where code and data has been placed, and sizes of individual sections



# Symbol Viewer and Disassembly




| Symbol | Address (Range) | Size | Flags |
|---|---|---|---|
| periph_timers.axf | | | |
| .text | (1a000000-1a004550) | 17744 | Local Debug |
| NVIC_EnableIRQ | 1a000330 | 52 | Local Function |
| NVIC_ClearPendingIRQ | 1a000364 | 52 | Local Function |
| Chip_RGU_TriggerReset | 1a000398 | 60 | Local Function |
| Chip_RGU_InReset | 1a0003d4 | 64 | Local Function |
| Chip_TIMER_MatchPending | 1a000414 | 52 | Local Function |
| Chip_TIMER_ClearMatch | 1a000448 | 40 | Local Function |
| Chip_TIMER_Enable | 1a000470 | 32 | Local Function |
| Chip_TIMER_SetMatch | 1a000490 | 40 | Local Function |
| Chip_TIMER_MatchEnableInt | 1a0004b8 | 52 | Local Function |
| Chip_TIMER_ResetOnMatchEnable | 1a0004ec | 54 | Local Function |
| Chip_SCU_PinMuxSet | 1a000600 | 52 | Local Function |
| Chip_ENET_RMIIEnable | 1a000634 | 40 | Local Function |
| Chip_GPIO_SetPinState | 1a00065c | 56 | Local Function |
| Chip_GPIO_SetPinDIROutput | 1a000694 | 62 | Local Function |
| Chip_UART_TXEnable | 1a0006d4 | 26 | Local Function |
| Chip_UART_SendByte | 1a0006f0 | 30 | Local Function |
| Chip_UART_ConfigData | 1a000710 | 28 | Local Function |
| Chip_UART_ReadLineStatus | 1a00072c | 24 | Local Function |
| gpioLEDBits | 1a0043a8 | 6 | Local Object |




periph_timers.dis

```
1171 1a000760:   f000 f836   bl  1a0007d0 <Board_UARTPutChar>
1172 1a000764:   697b        ldr r3, [r7, #20]
1173 1a000766:   3301        adds    r3, #1
1174 1a000768:   617b        str r3, [r7, #20]
1175 1a00076a:   687b        ldr r3, [r7, #4]
1176 1a00076c:   697a        ldr r2, [r7, #20]
1177 1a00076e:   429a        cmp r2, r3
1178 1a000770:   d3f1        bcc.n   1a000756 <__sys_write+0x12>
1179 1a000772:   687b        ldr r3, [r7, #4]
1180 1a000774:   4618        mov r0, r3
1181 1a000776:   3718        adds    r7, #24
1182 1a000778:   46bd        mov sp, r7
1183 1a00077a:   bd80        pop {r7, pc}
1184
1185 1a00077c <Board_UART_Init>:
1186 1a00077c:   b580        push    {r7, lr}
1187 1a00077e:   b082        sub sp, #8
```

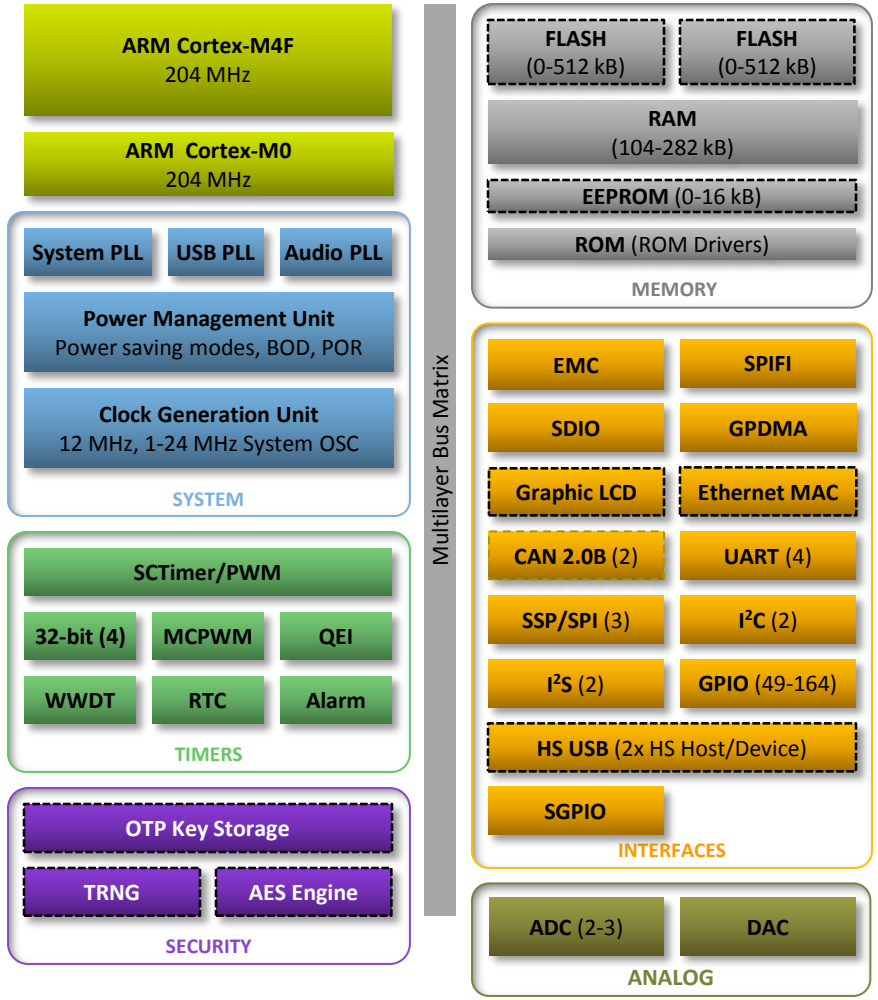
**Display the symbols from the generated AXF file**

**Display the code in the generated AXF file**

Disassembly can also be displayed using : Binary Utilities -> Disassemble

# LAUNCHING A DEBUG SESSION

# LPC4300 MCUs

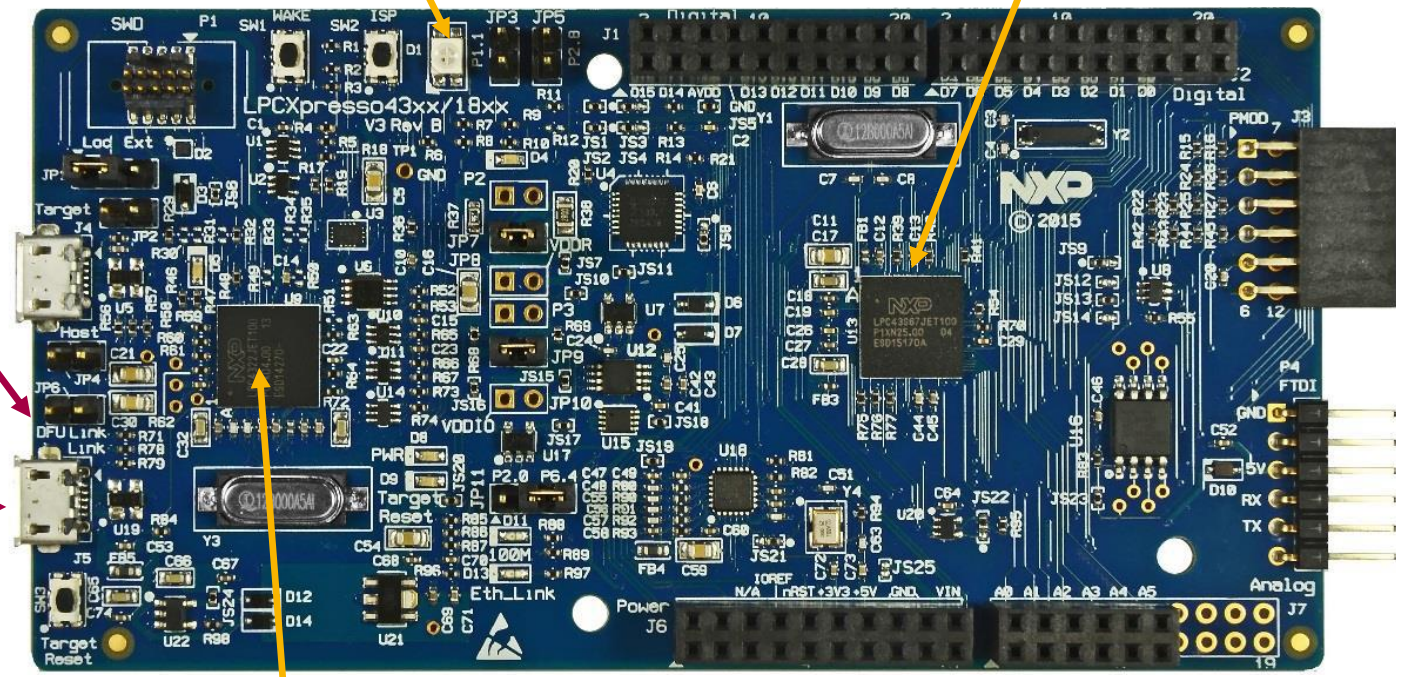



- 204 MHz Cortex-M4F processor and Cortex-M0 co-processor (x2 on LPC4367/4370)
  - Up to 1 MB dual-bank Flash; 282 kB RAM
  - Flashless + XIP from QSPI via SPIFI
  - Signal processing capabilities
- High-speed connectivity, display, timing
  - FS/HS USB w/on-chip FS/HS PHY, dual-host capabilities
  - Graphic LCD, free emWin graphic libraries
  - SCTimer/PWM, SGPIO
- Security features (LPC43Sxx), including
  - Hardware AES-128 encryption engine
  - Two 128-bit non-volatile OTP memories
  - True random number generator
- Pin compatible with LPC18/18S
- Available families:
  - LPC43x2, x3, x5, x7)
  - LPC43x0 (Flashless)
  - LPC437x
  - LPC43Sxx (security features for code & data protection

# LPCXpresso4367 Board Configuration

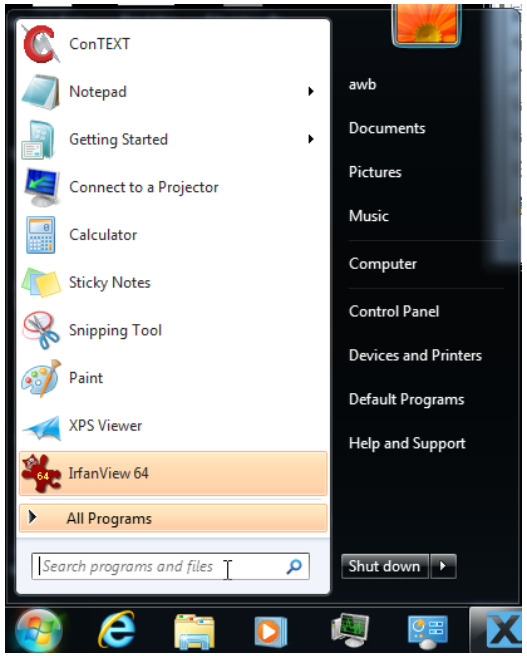
Before connecting to USB, ensure that “DFU Link” (JP6) jumper is fitted so that LPCXpresso IDE can softload probe firmware

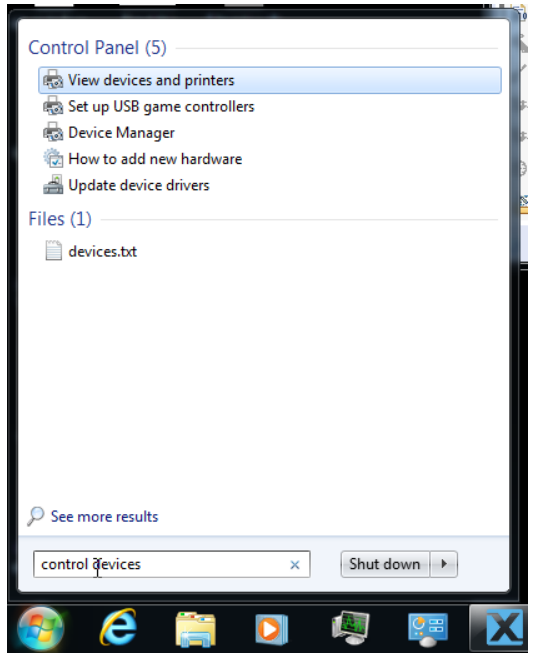
Connect USB cable from PC to “Link” port (J5) to provide debug

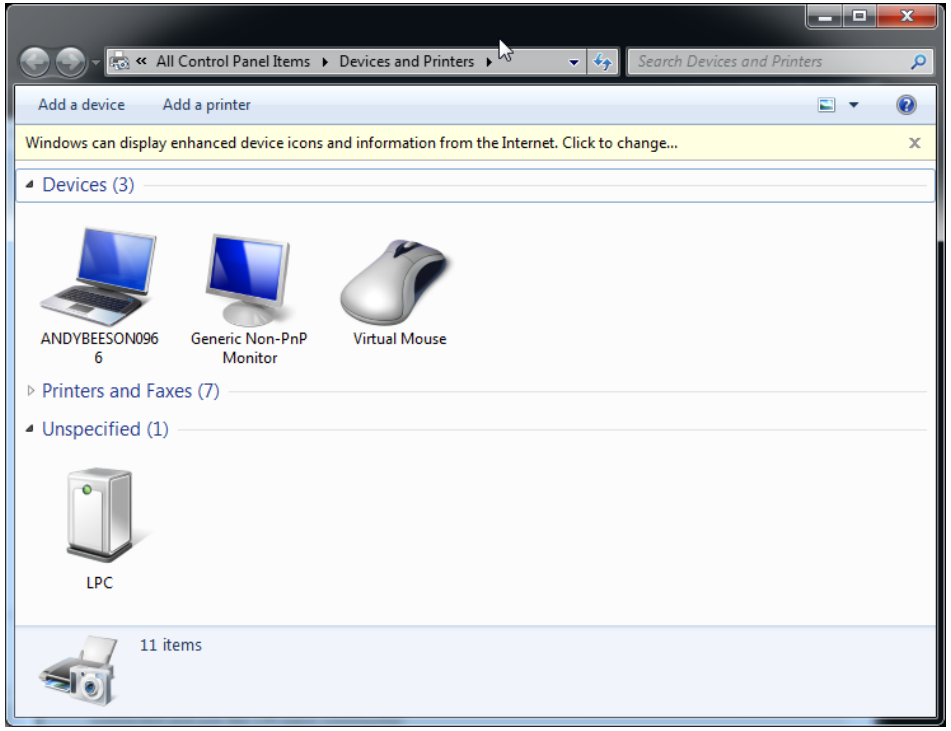


# Boot The Probe (Manually) – 1

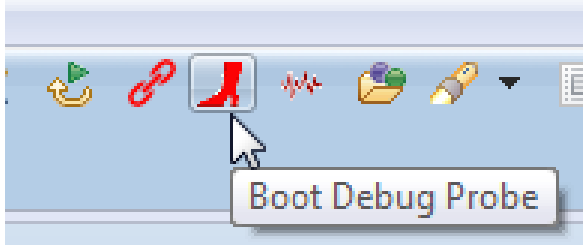
- Enter “control devices” into Start Menu search box and hit Enter
  - This will display “*Devices and Printers*” dialog within Control Panel
- Once your LPCXpresso4367 board is connected over USB, you should see an “LPC” device appear



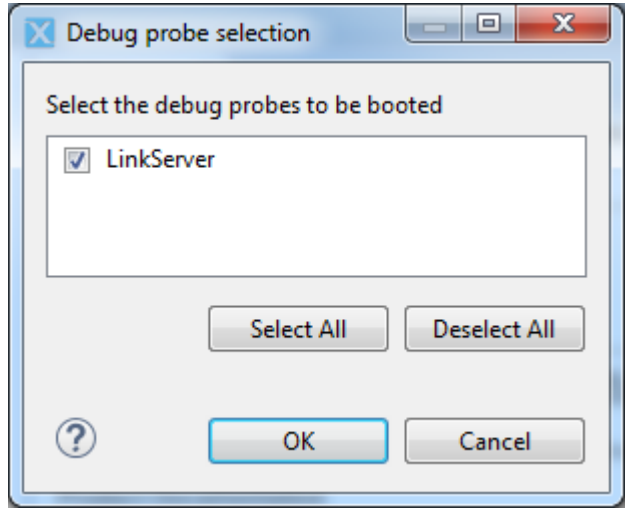



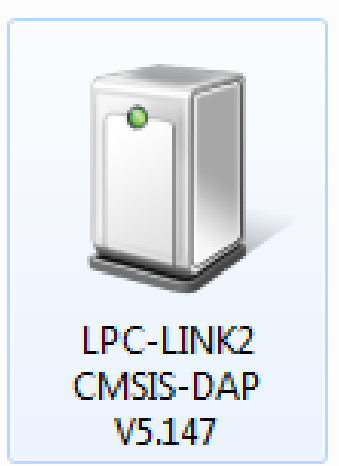



# Boot The Probe (Manually) – 2

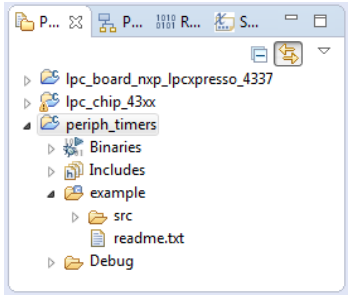


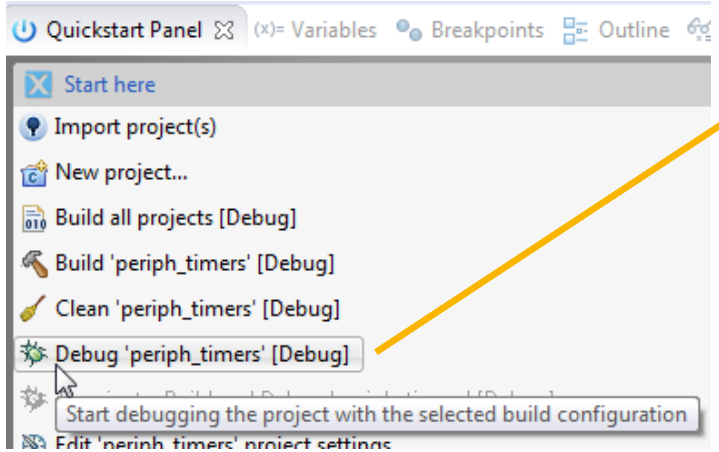

- Now click on the “Boot Debug Probe” button in the LPCXpresso IDE menu bar
- Select LinkServer and click OK
  - A popup should appear telling you the probe firmware is initializing.
- Once the firmware is downloaded, and Windows has installed any necessary drivers, then “*Control Panel / Devices and Printers*” should update to show your probe ready to access
  - Just OK any firewall messages
- **NOTE:** In normal LPCXpresso IDE operation, there is no need to manually boot the probe in this way – booting will done automatically in the background by the IDE when you start a debug session



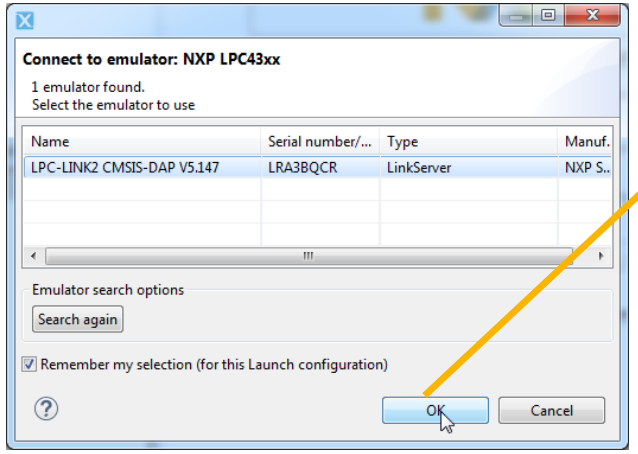
# Start Debug Session



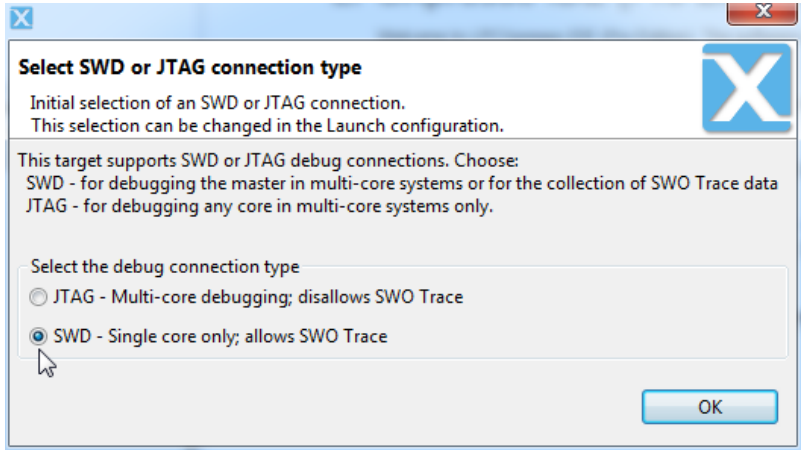




Select project in the Project Explorer View, then click on Debug in the Quickstart Panel

These settings will be remembered for next time you debug this project



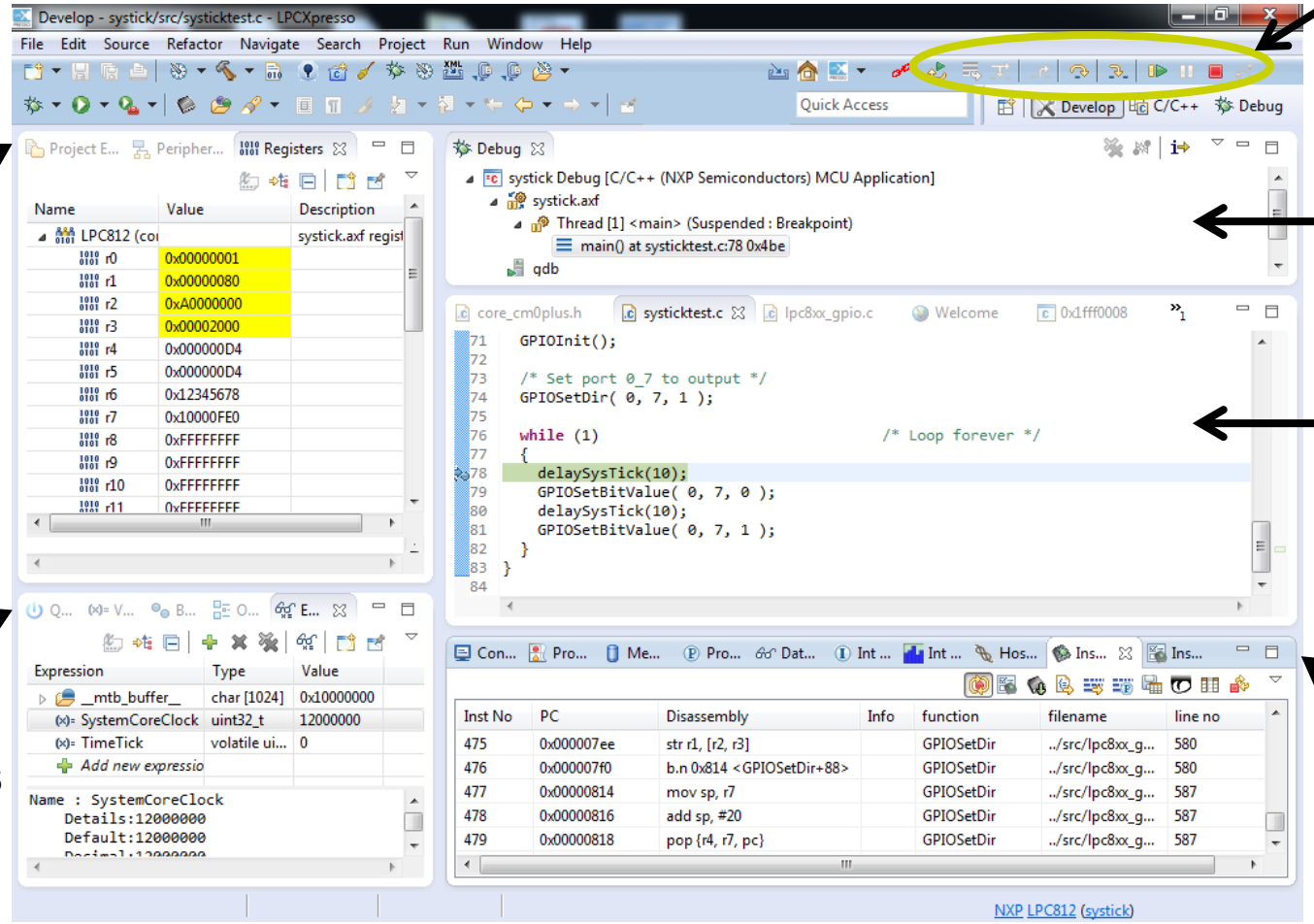

The IDE should identify your booted LPC-Link2 debug probe – click OK




For LPC43xx parts, you need to tell the debugger what debug connection type to use – select ***SWD*** and click OK

Note: By default, selecting “Debug” will trigger a build before the debug session is launched

# Develop Perspective – Debugging




Run Controls

**Debug** view

**Editor** view

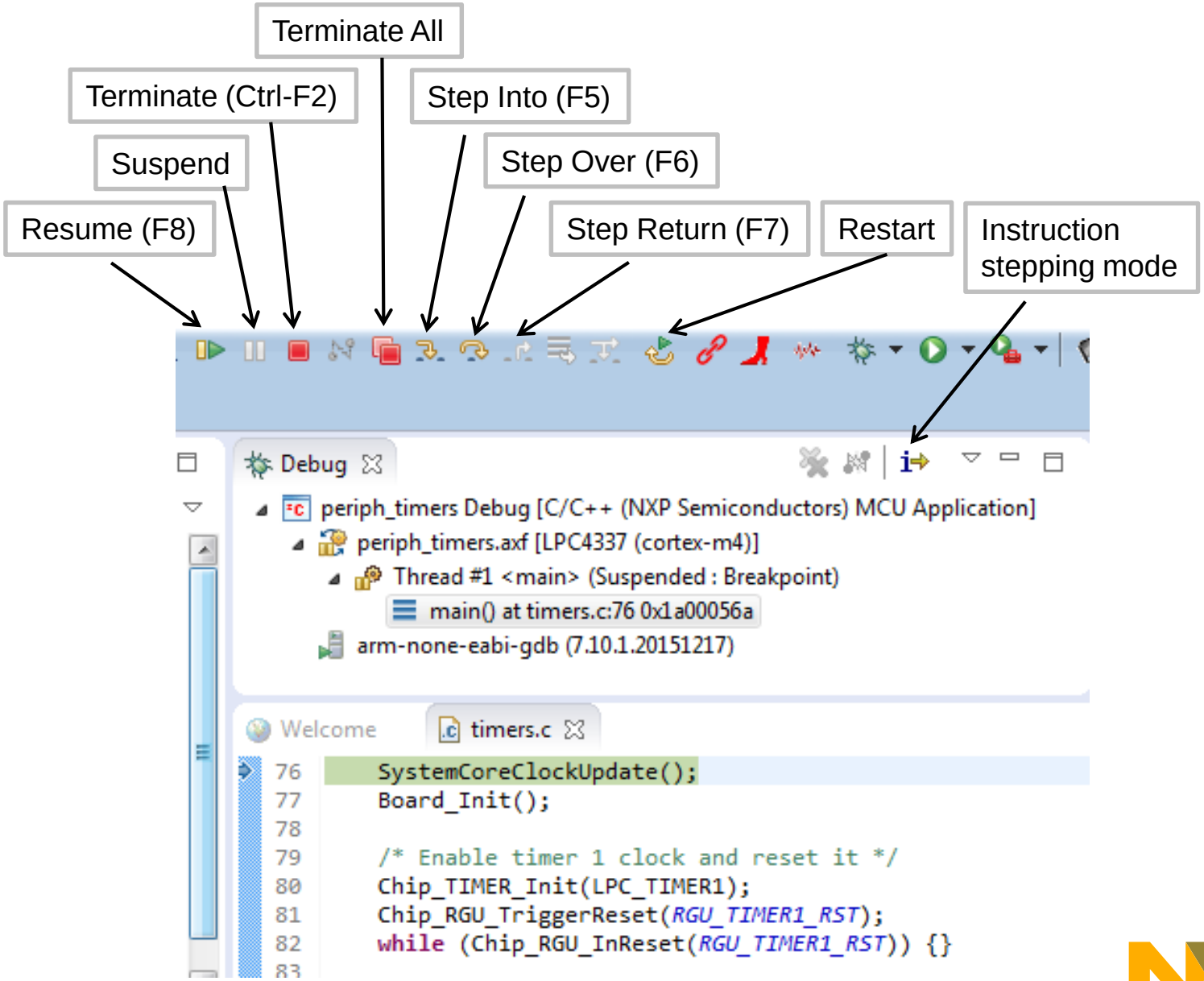
**Registers** and **Peripherals** views

**Variables, Expressions** and **Breakpoints** views

**Console, Memory** and **Trace** views

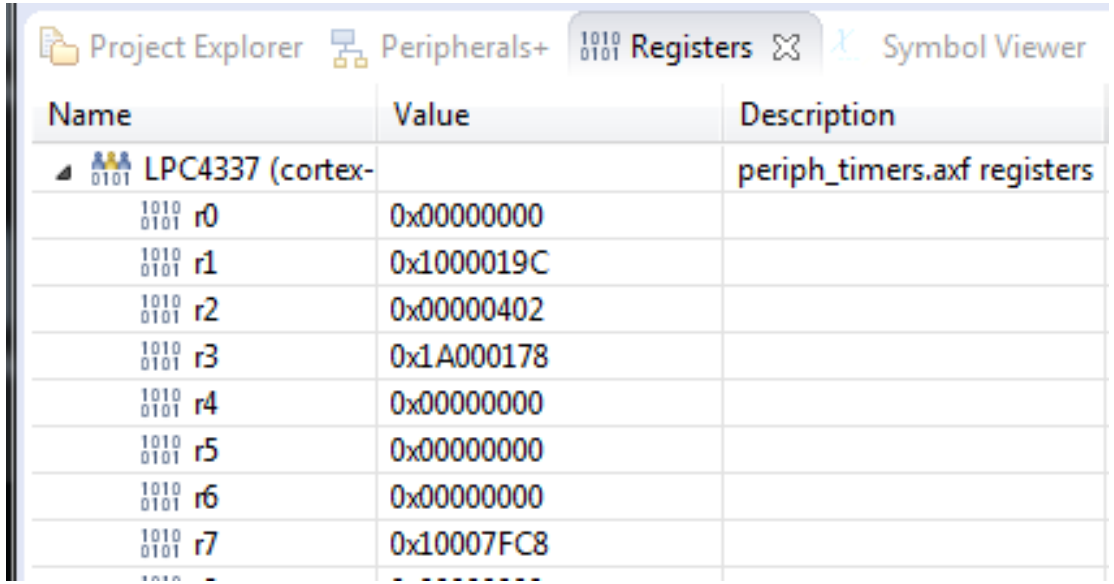
# Stopped At Main()

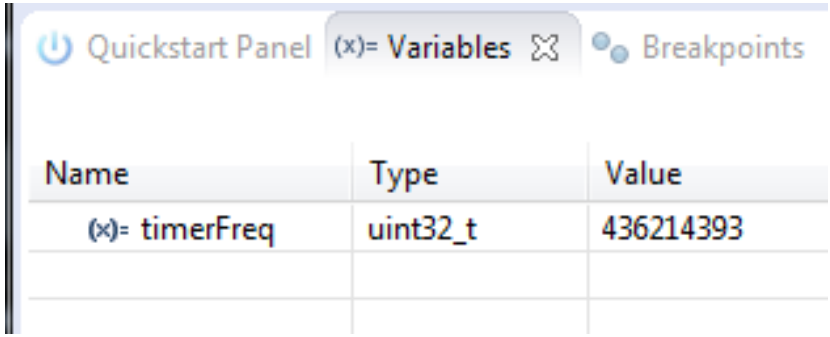
- Image downloaded to flash and executed
  - Default breakpoint set on function main()
- Debug View displayed automatically
  - Shows / controls current scope and target (multicore)
  - Run controls are on main toolbar
- **But before you begin to run the code …**



# Registers and Local Variables

- Look at the contents of the Registers View and Variables View

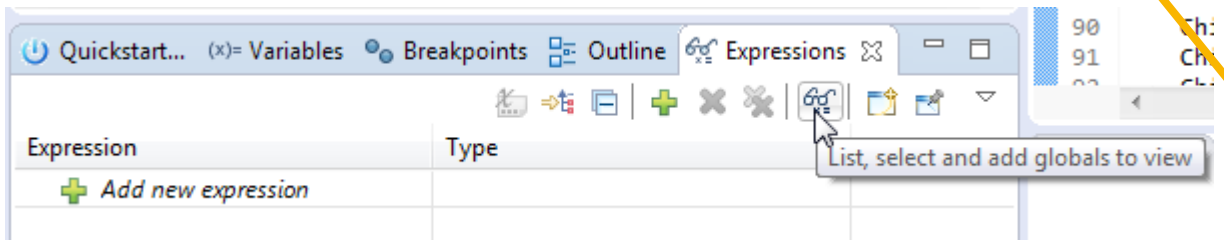


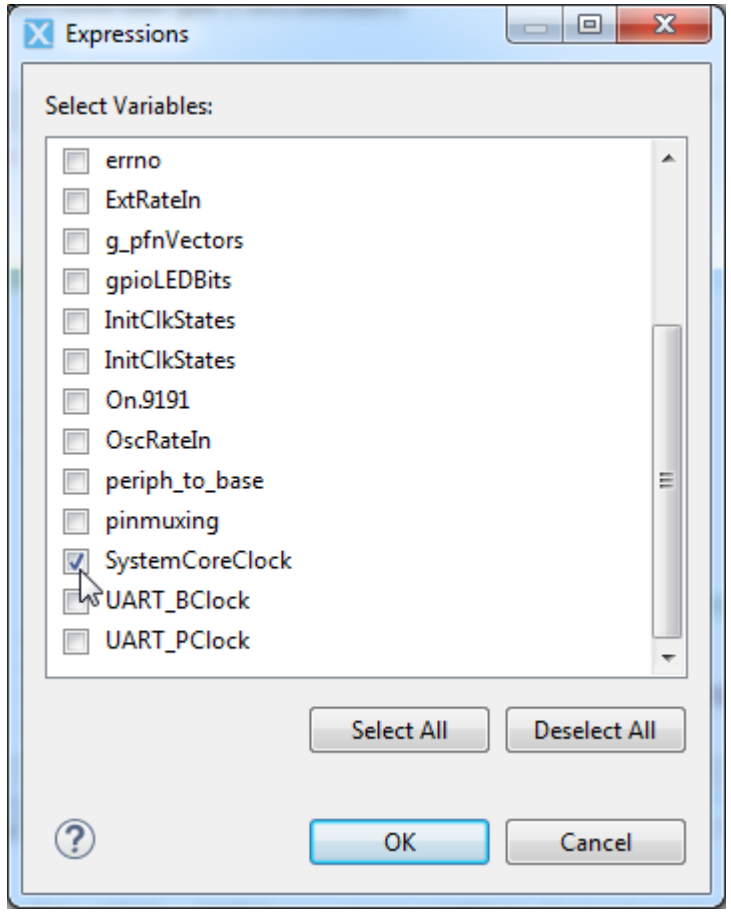



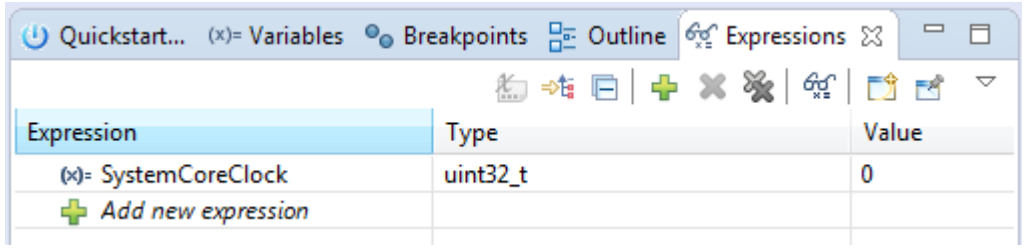

In-scope local variables displayed
Locals displayed will change as move up and down the call stack

# Add a Global Variable to the Expressions View

Switch to the Expressions View and click on the “List…Globals” button




SystemCoreClock global is now visible in the Expressions View

Scroll down and select “SystemCoreClock”, which will hold the main CPU clock speed

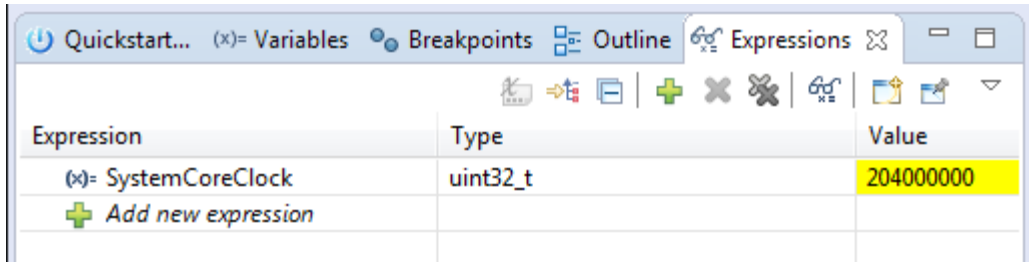
# Step Over and Updates to Registers and Expressions Views

Now do a “Step Over” 

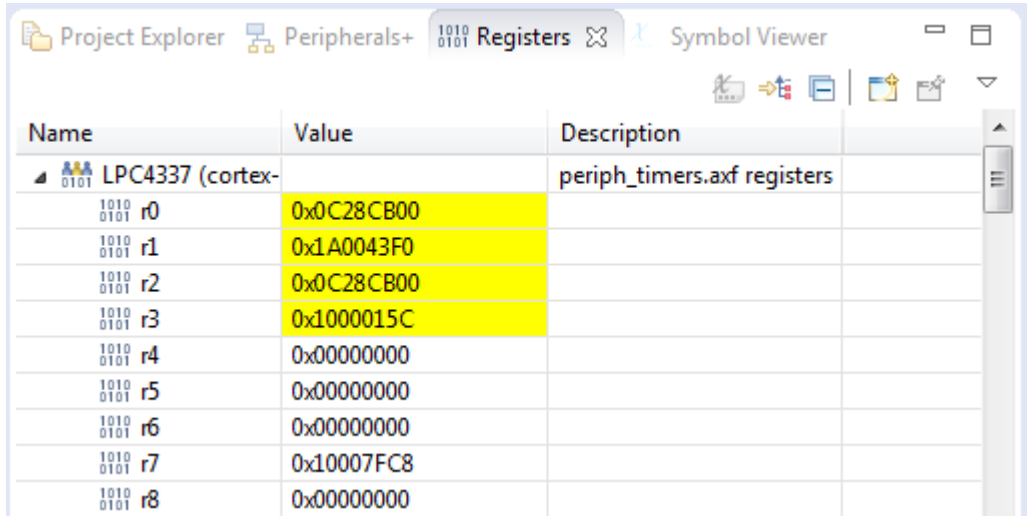
After doing a “Step Over”, you should have executed and returned from the call to SystemCoreClockUpdate()…

timers.c

```
72  int main(void)
73  {
74      uint32_t timerFreq;
75
76      SystemCoreClockUpdate();
77      Board_Init();
78
79      /* Enable timer 1 clock and reset it */
80      Chip_TIMER_Init(LPC_TIMER1);
```



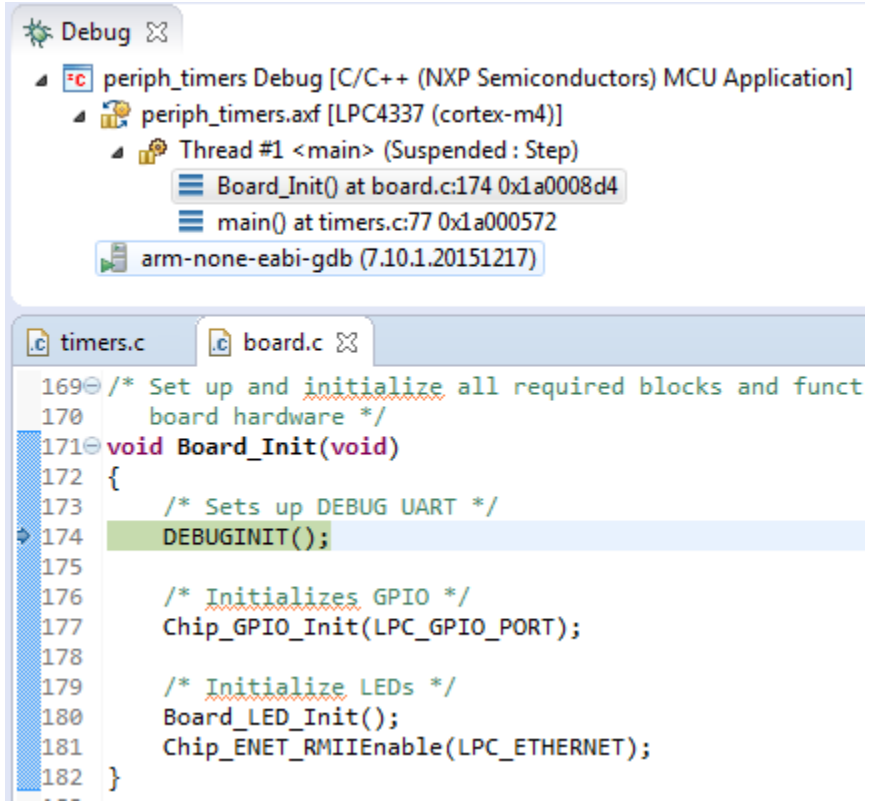

Changes in Register View and Expressions View are highlighted when execution pauses



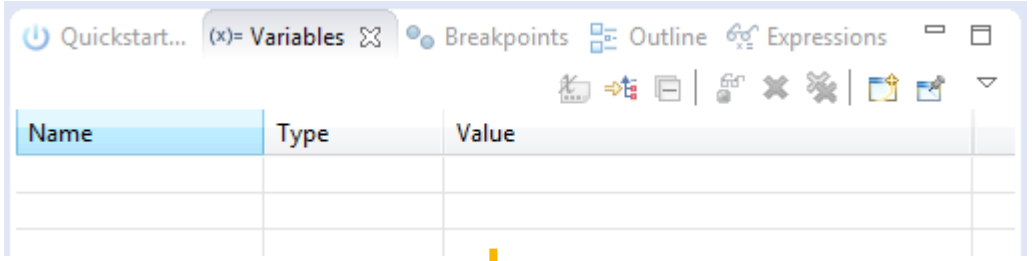
# Step Into and Updates to Debug and Variables Views

Now do a “Step Into”

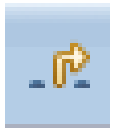
You should now be at the start of Board_Init()
Note that the Debug View has updated to show the call stack

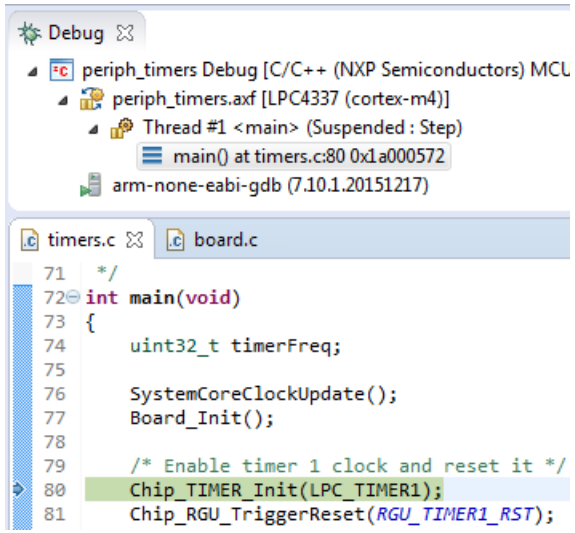



The Variables View will also be updated to show the Local Variables at the current location (none in this case)



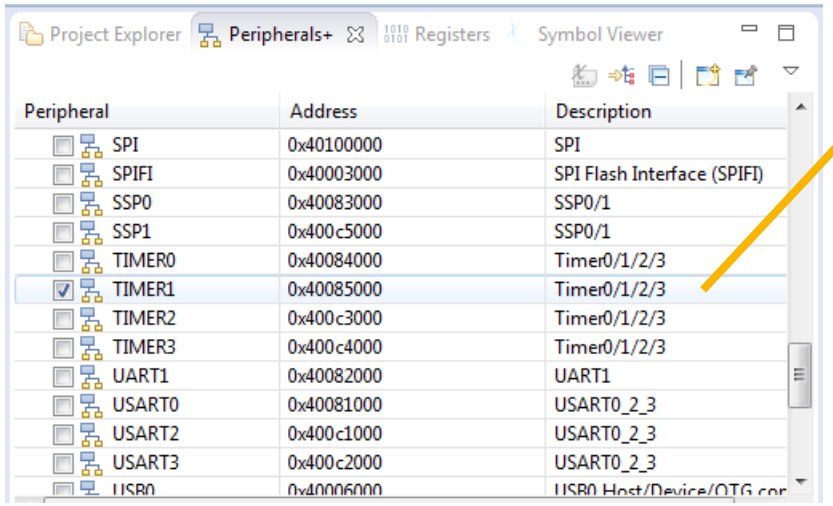

Now use “Step Out” to return to main()



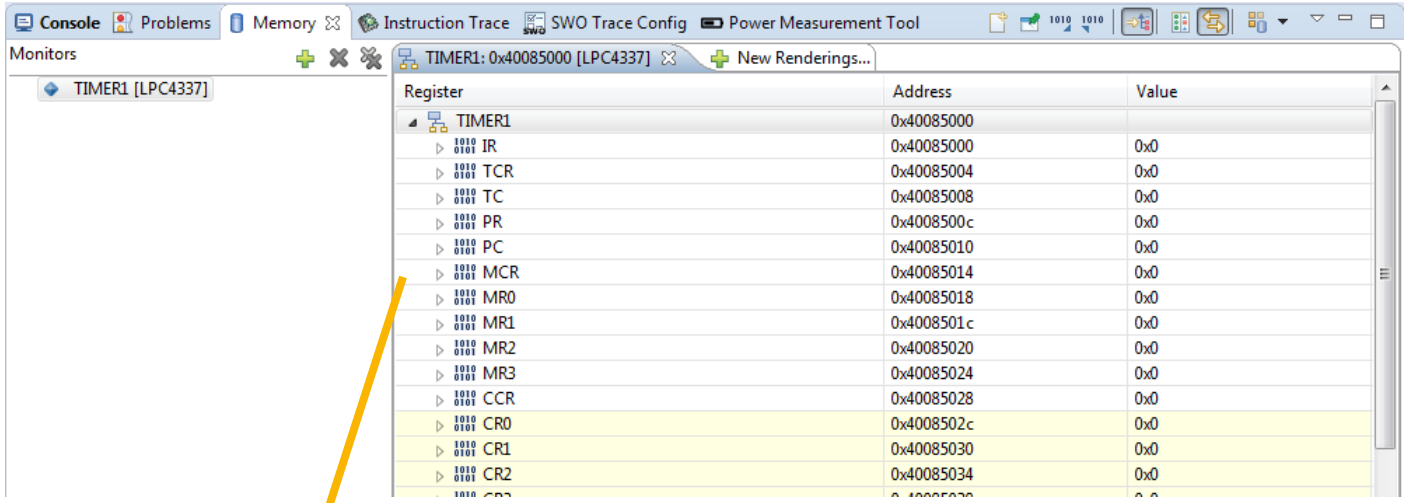


# Display Peripheral Registers

Open the Peripherals+ View, and select Timer1

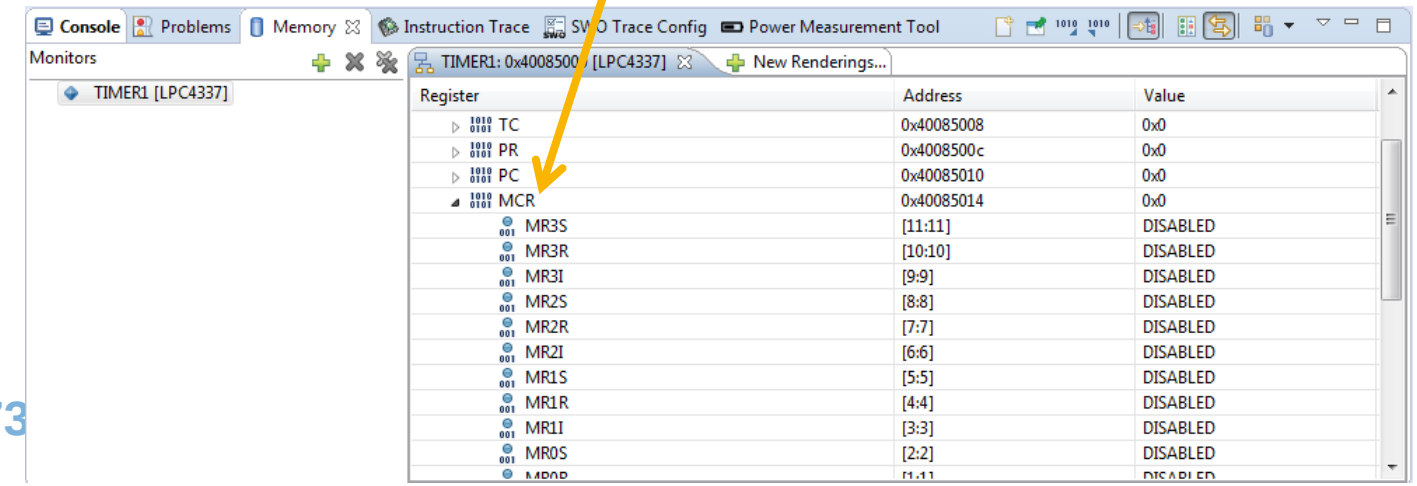



Timer1 registers are displayed in the Memory View

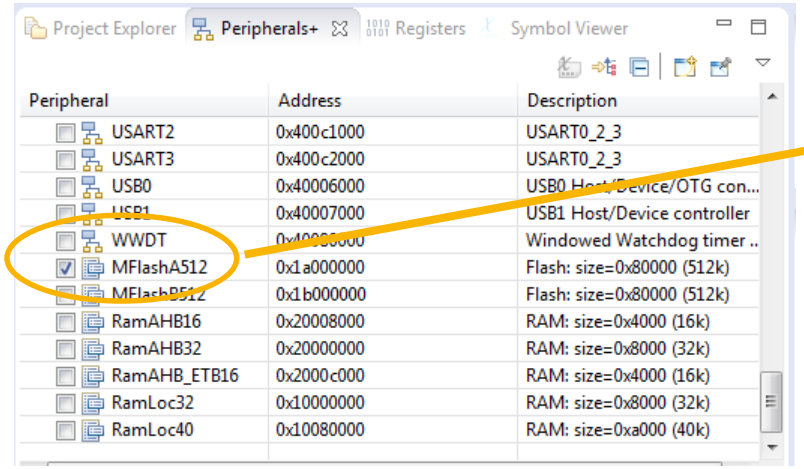


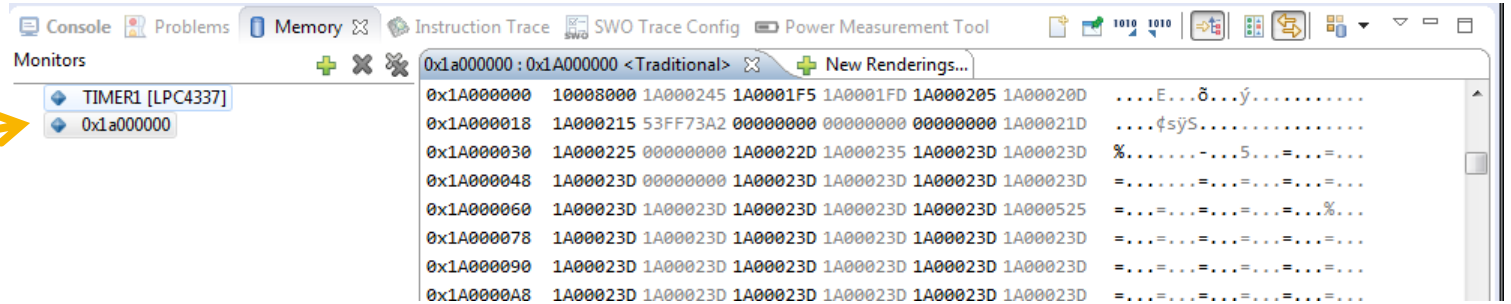

Beige registers are Read Only

Expand a register (e.g MCR) to see the bitfields within, including enumerations

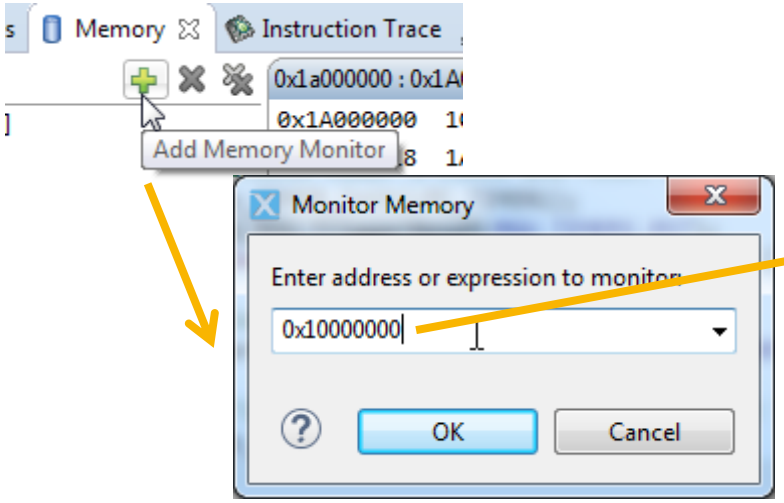


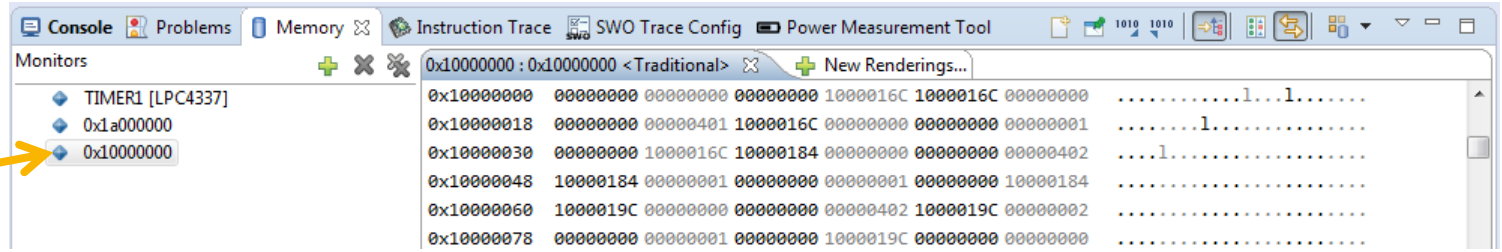
# Viewing Memory




Add a memory region to Memory View via the Peripherals+ View (e.g. MFlashA512)

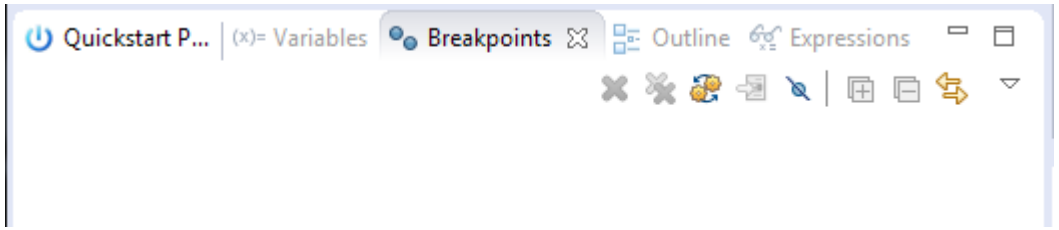


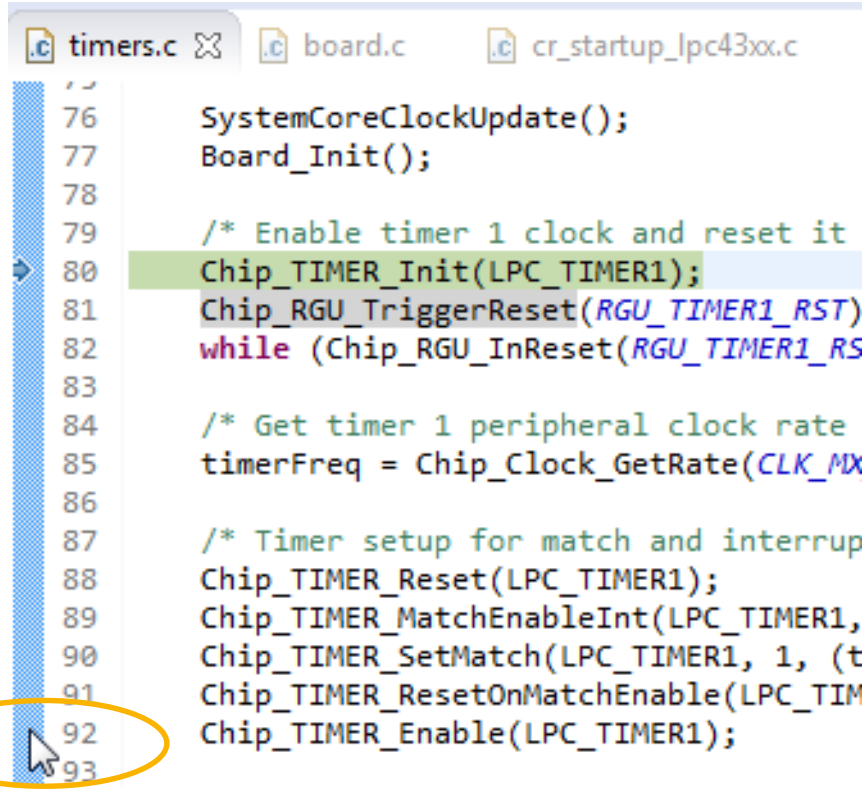

Add an arbitary address using the “Add Memory Monitor” button

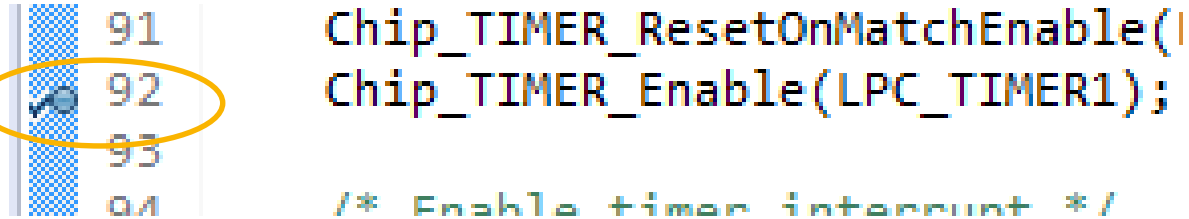
# Add a Breakpoint

Switch to the Breakpoints View.
No breakpoints currently set

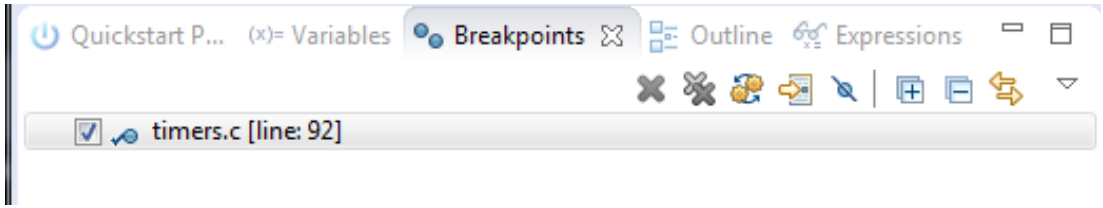



Double click in the margin of the Editor View to set a breakpoint on the call to Chip_TIMER_Enable()



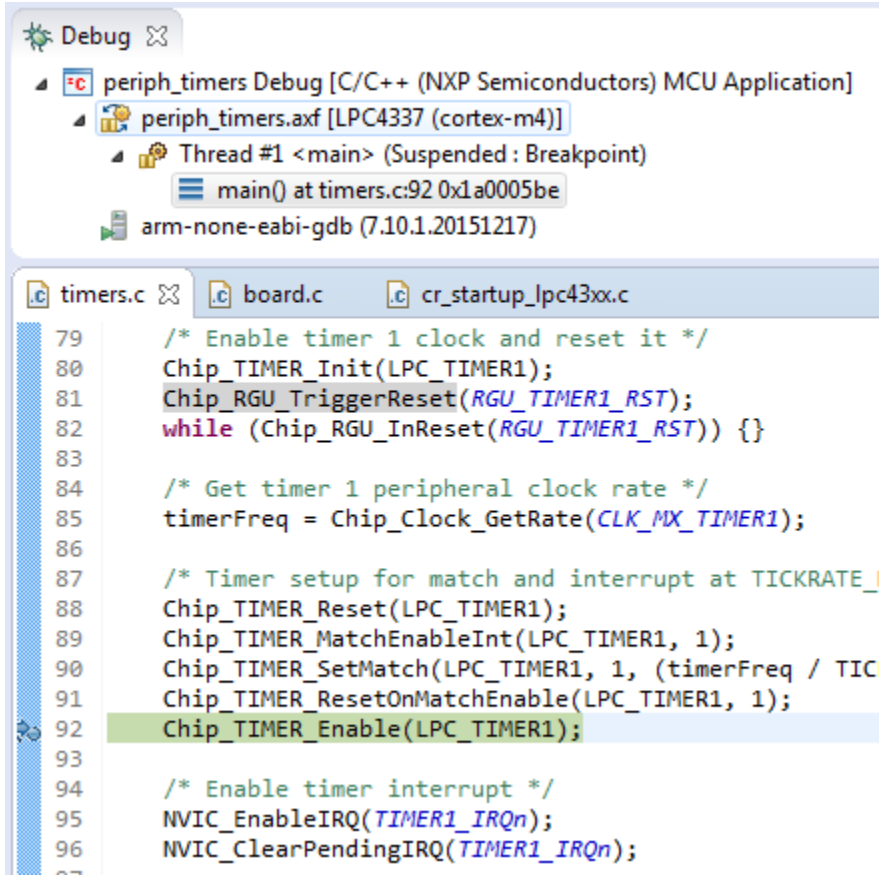

Breakpoint shown in Editor View and Breakpoint View



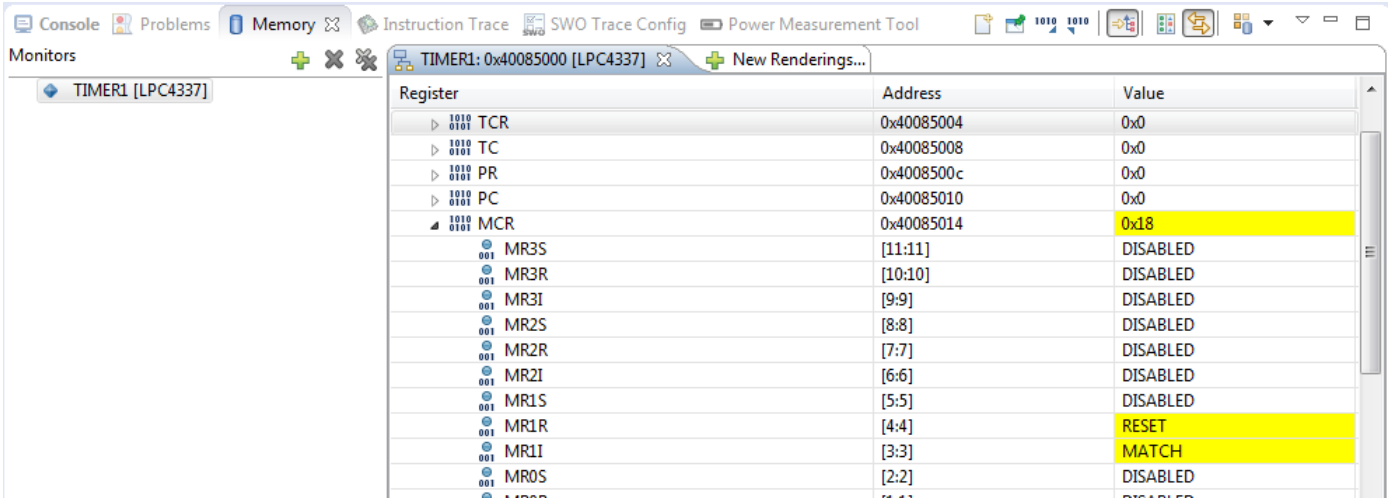

More breakpoint options on Right-Click context sensitive Menu in Editor View margin and in Breakpoints View

# Resume Execution

Resume execution (Go!) and the breakpoint should be hit




Note that the TIMER1 registers have updated




Now Resume execution again

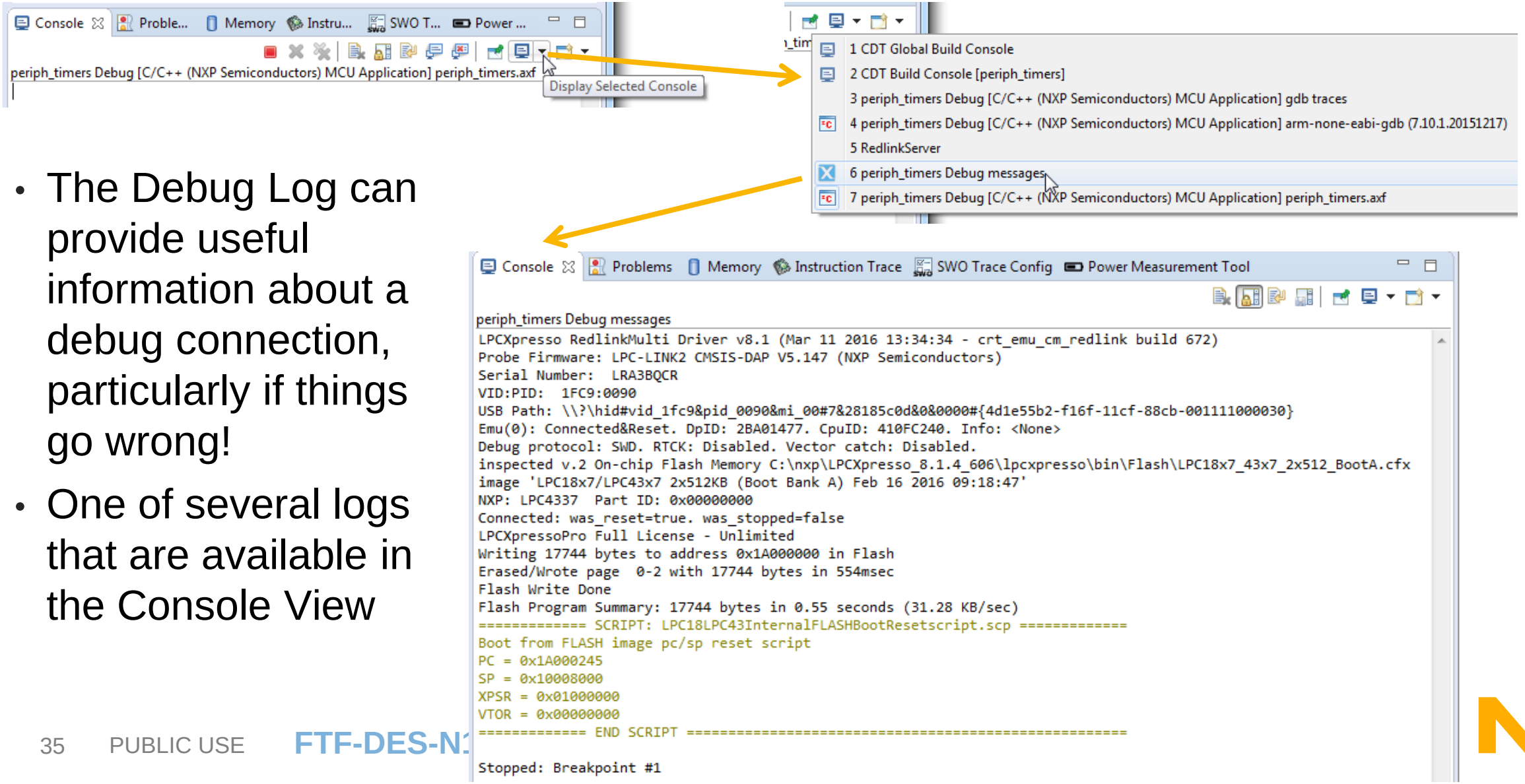
The LED on the board should blink blue

Use Suspend to pause execution

Use Restart to go back to the beginning of the Application

Use Terminate to quit debug session

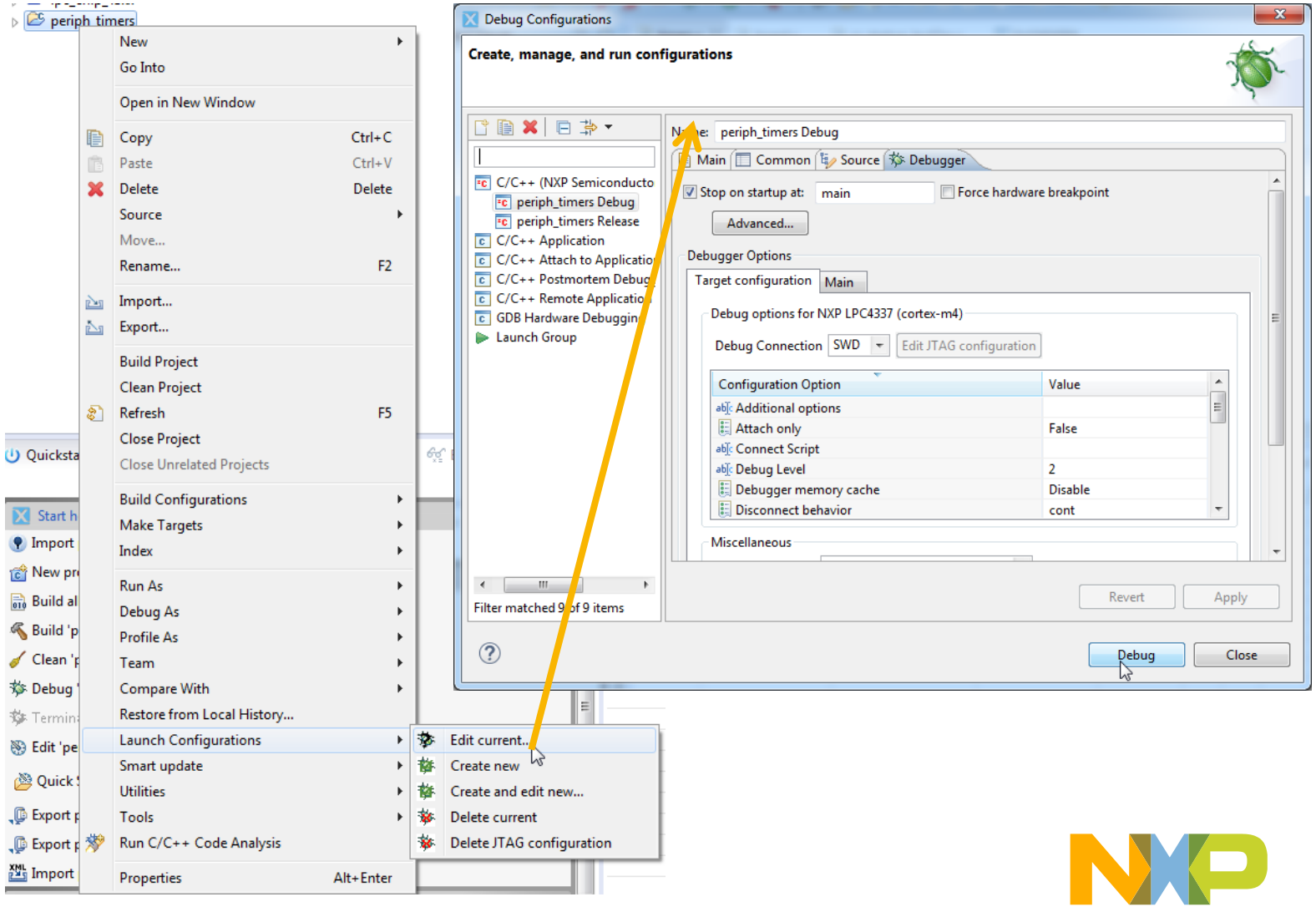
# Console – Debug Log (Messages)




- The Debug Log can provide useful information about a debug connection, particularly if things go wrong!
- One of several logs that are available in the Console View

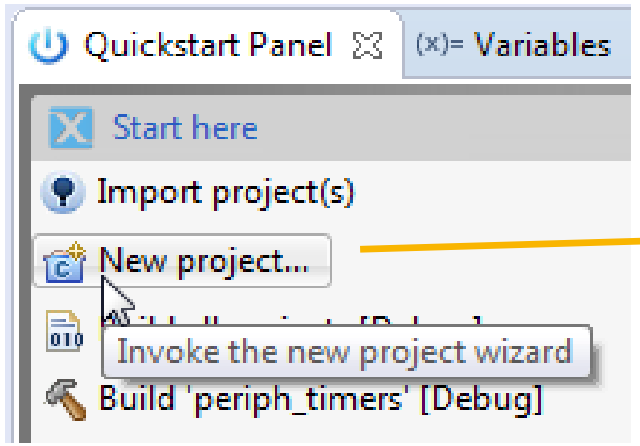
# Launch Configurations

- LPCXpresso IDE automatically creates “launch configuration” files in a project to store the settings for a debug connection (1 per build configuration)
  - "***<projname>* Debug.launch**”
  - "***<projname>* Release.launch**"
- Normally no need to touch launch configurations, as default settings should work in most cases without problem
- Can be accessed if required the "Launch Configurations" entry on the context sensitive menu available from the Project Explorer view...

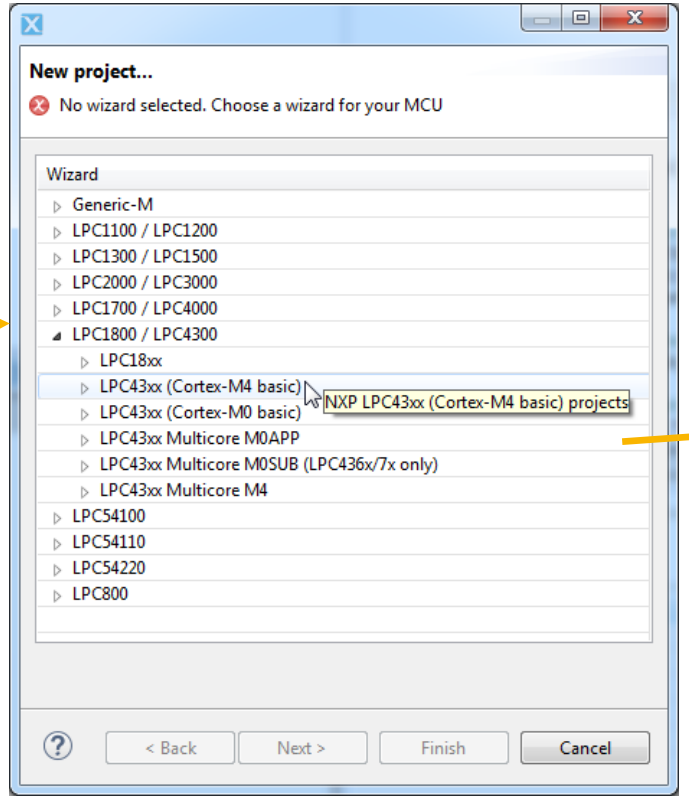


# CREATING AND EDITING PROJECTS

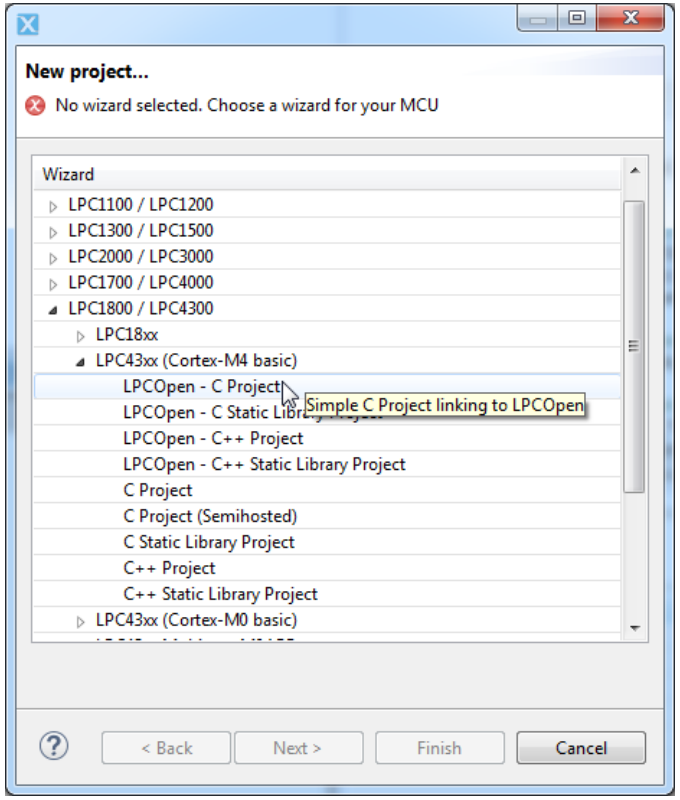
# Create a New Project




Click on ***New project*** in the Quickstart Panel

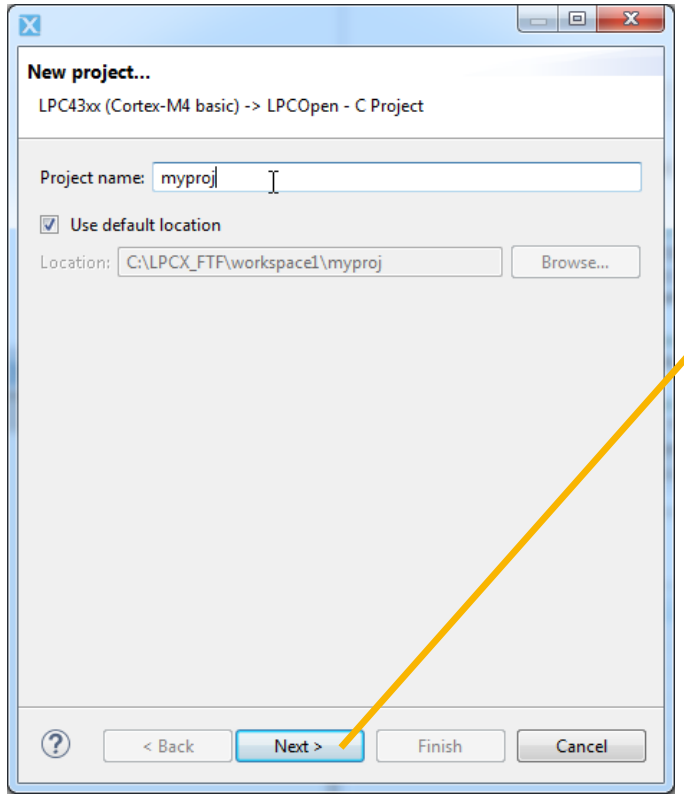



Open up the "*expander*" for the part family you want to create a project for

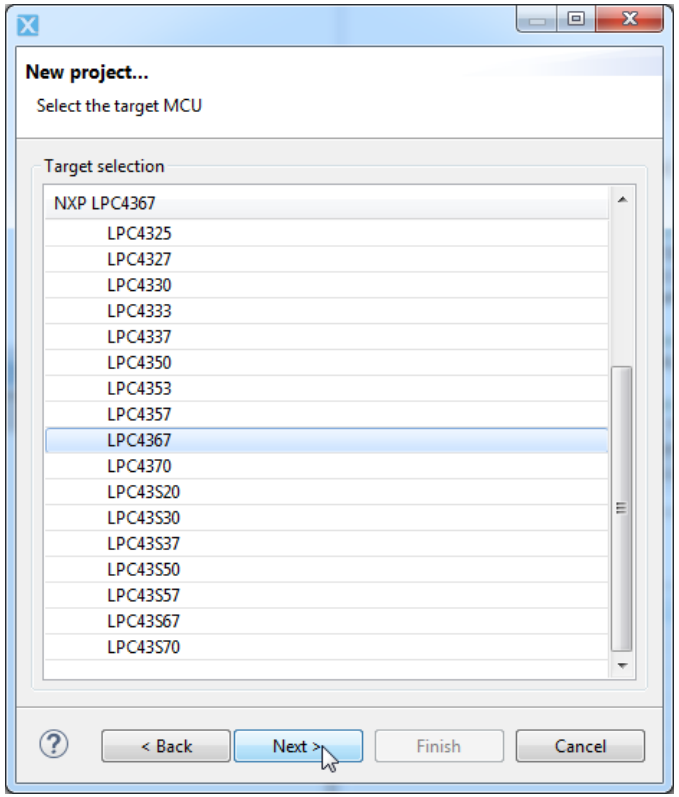



Expand again and select the type of project you want to create – generally an LPCOpen one

# Project Name and MCU



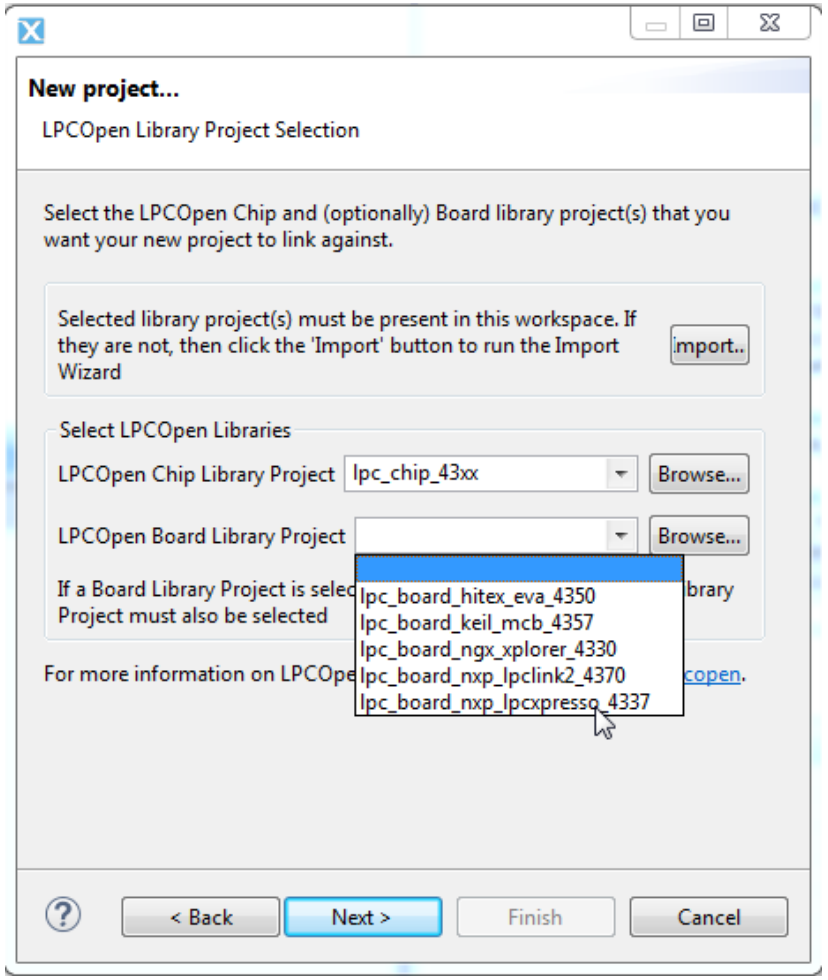

Enter a name for your project, say “myproj”, then click Next




Select the target MCU for your project, then click Next

- Selecting the correct target MCU is important:
  - CPU settings (e.g. Cortex-M4 vs Cortex-M0)
  - Auto-generated source files (e.g. startup code)
  - MCU specific options within rest of project wizard
  - Default memory map used in automatic generation of linker scripts
  - Debug launch configurations

# LPCOpen Library Selection

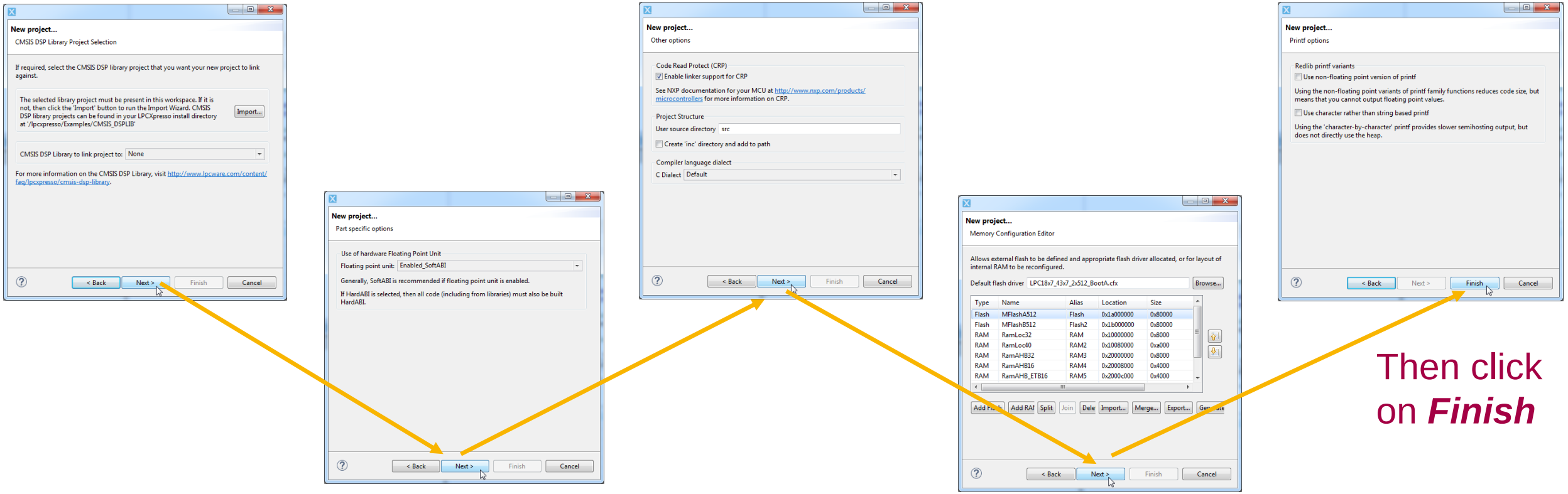



- For LPCOpen based projects, need to select the Chip and Board libraries that will be linked against
  - Chip Library automatically selected from MCU selection
  - Board Library : Needs to be selected based on your target board
    - For LPCXpresso4367 board, select `lpc_board_nxp_lpcxpresso_4337`

- Note that if you have not yet imported the LPCOpen library projects into the workspace, you can use the Import option to pull them in without needing to terminate the wizard.

# Other New Project Dialogs

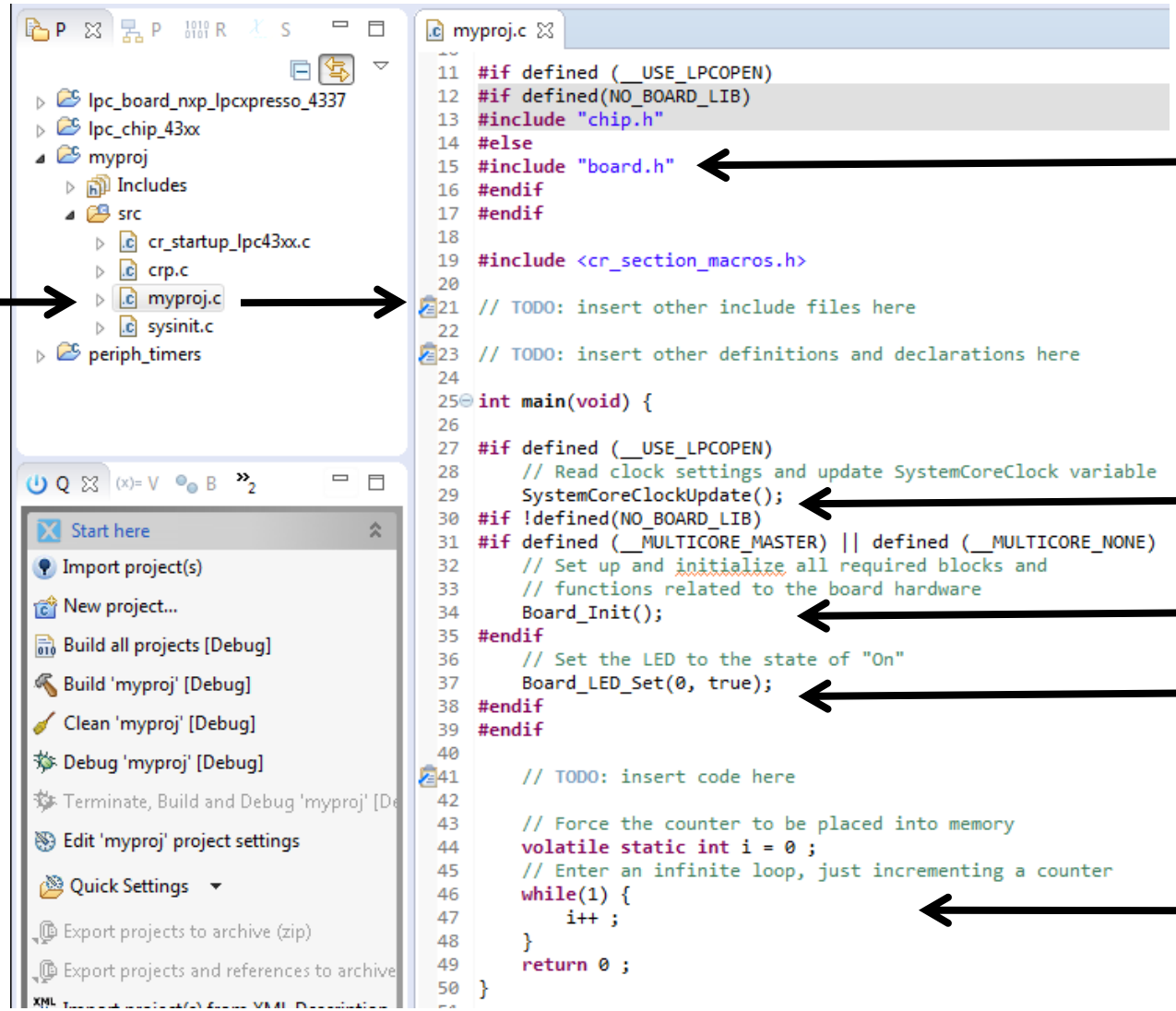
- You will then see a number of additional (part / project type specific) dialogs
  - For now, click on “Next” on all of them until you reach the screen with “Finish” Enabled




Then click on ***Finish***

# Project Created …

File containing a simple main() function automatically generated for project




myproj.c

```
11 #if defined (__USE_LPCOPEN)
12 #if defined(NO_BOARD_LIB)
13 #include "chip.h"
14 #else
15 #include "board.h"
16 #endif
17 #endif
18
19 #include <cr_section_macros.h>
20
21 // TODO: insert other include files here
22
23 // TODO: insert other definitions and declarations here
24
25 int main(void) {
26
27 #if defined (__USE_LPCOPEN)
28     // Read clock settings and update SystemCoreClock variable
29     SystemCoreClockUpdate();
30 #if !defined(NO_BOARD_LIB)
31 #if defined (__MULTICORE_MASTER) || defined (__MULTICORE_NONE)
32     // Set up and initialize all required blocks and
33     // functions related to the board hardware
34     Board_Init();
35 #endif
36     // Set the LED to the state of "On"
37     Board_LED_Set(0, true);
38 #endif
39 #endif
40
41     // TODO: insert code here
42
43     // Force the counter to be placed into memory
44     volatile static int i = 0 ;
45     // Enter an infinite loop, just incrementing a counter
46     while(1) {
47         i++ ;
48     }
49     return 0 ;
50 }
```

LPCOpen header file

Read MCU clock settings (as setup by initialization code)

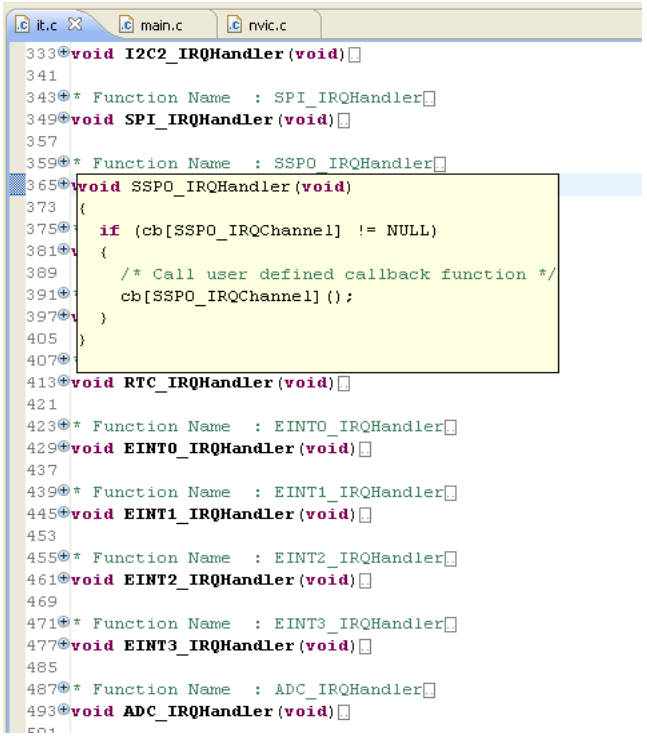
Board initialization

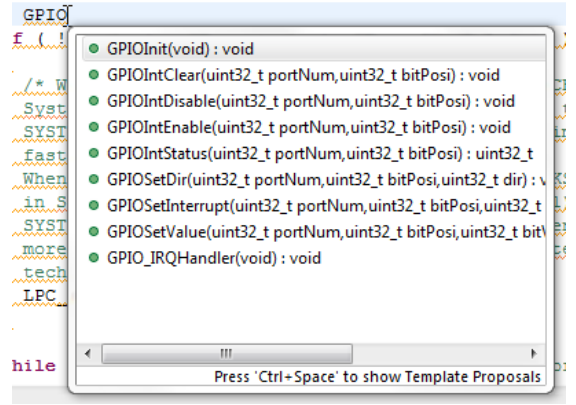
Turn LED 0 on!

Increment counter inside infinite loop

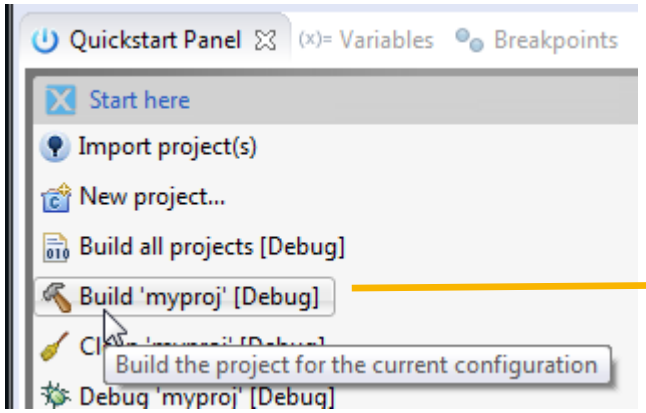
# Editor View

- Source outlining and folding
  - Within the editor, functions, structures etc may be folded to show the structure and hide the detail
  - Separate outline view lists each element of the current file
- Editor templates and Code completion
  - Ctrl-Space at any point will list available editor template, function names etc
  - Ctrl-Shift-Space for parameters
  - Alt-/ for word completion
  - Predefined templates are user extensible
- Miscellaneous
  - Syntax coloring, Brace matching, Source formatting and indenting, Comment/uncomment block, Line numbers



# Build the Project

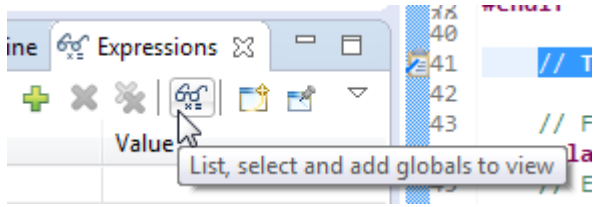



Highlight your new project, then click on Build

```
Building file: ../src/sysinit.c
Invoking: MCU C Compiler
arm-none-eabi-gcc -D__MULTICORE_NONE -DDEBUG -D__CODE_RED -DCORE_M4 -D__USE_LPCOPEN -D__LPC43XX__ -D__REDLIB__
-I"C:\LPCX_FTF\workspace1\lpc_board_nxp_lpcxpresso_4337\inc" -I"C:\LPCX_FTF\workspace1\lpc_chip_43xx\inc"
-I"C:\LPCX_FTF\workspace1\lpc_chip_43xx\inc\usbd" -O0 -fno-common -g3 -Wall -c -fmessage-length=0 -fno-builtin
-ffunction-sections -fdata-sections -fsingle-precision-constant -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -mfloat-abi=softfp
-mthumb -specs=redlib.specs -MMD -MP -MF"src/sysinit.d" -MT"src/sysinit.o" -MT"src/sysinit.d" -o "src/sysinit.o"
"../src/sysinit.c"
Finished building: ../src/sysinit.c

Building target: myproj.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\LPCX_FTF\workspace1\lpc_board_nxp_lpcxpresso_4337\Debug"
-L"C:\LPCX_FTF\workspace1\lpc_chip_43xx\Debug" -Xlinker -Map="myproj.map" -Xlinker --gc-sections -Xlinker
-print-memory-usage -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -mfloat-abi=softfp -mthumb -T "myproj_Debug.ld" -o "myproj.axf"
./src/cr_startup_lpc43xx.o ./src/crp.o ./src/myproj.o ./src/sysinit.o   -llpc_board_nxp_lpcxpresso_4337 -llpc_chip_43xx
Memory region         Used Size  Region Size  %age Used
      MFlashA512:        6516 B       512 KB      1.24%
      MFlashB512:          0 GB       512 KB      0.00%
        RamLoc32:          16 B        32 KB      0.05%
        RamLoc40:          0 GB        40 KB      0.00%
        RamAHB32:          0 GB        32 KB      0.00%
        RamAHB16:          0 GB        16 KB      0.00%
    RamAHB_ETB16:          0 GB        16 KB      0.00%
      RamM0Sub16:          0 GB        16 KB      0.00%
       RamM0Sub2:          0 GB         2 KB      0.00%
Finished building target: myproj.axf

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "myproj.axf"; # arm-none-eabi-objcopy -v -O binary "myproj.axf" "myproj.bin" ; # checksum -p LPC4367 -d
"myproj.bin";
   text    data     bss     dec     hex filename
   6516       0      16    6532    1984 myproj.axf
```

Projects created in LPCXpresso IDE v8.1.4 (and later) will use `-print-memory-usage` linker option to display additional memory usage info

# Try for Yourself…

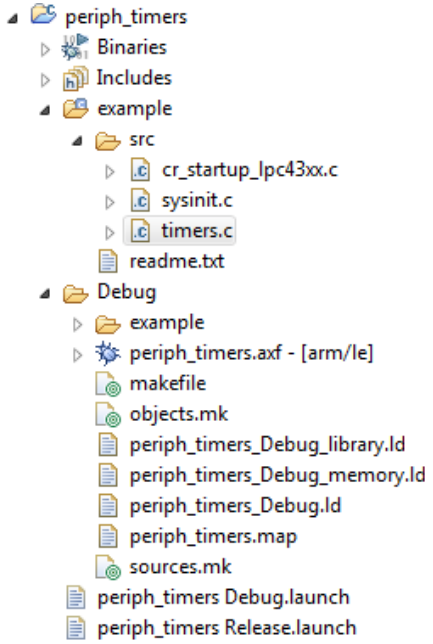


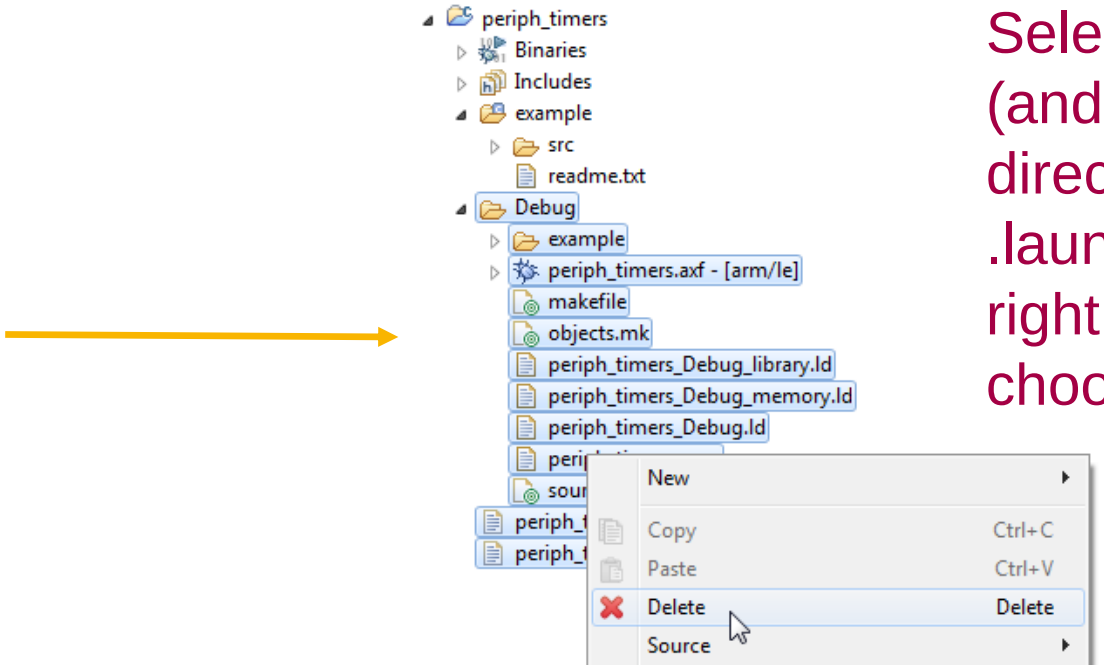

1. Start a debug session for myproj
2. Display the “Expressions” view, and add “SystemCoreClock” to it
3. Single step through the code
   - Watch for SystemCoreClock updating after the call to SystemCoreClockUpdate()
   - View the registers to see them changing as you step
   - Switch to the “Variables” view and see the value of “i” updating as you execute the while loop
4. Terminate the debug session
5. Replace “`// TODO: insert code here`” with a function call to turn LED 1 on.
6. Debug and single step again, noting when each LED color gets turned on.

# Copying Projects

- Sometimes using an existing project might be a better starting point than using the new project wizard.
- Can copy and paste a project, but needs to be done with care – otherwise you end up with files with the original project name in your new project, which can confuse!
- Note that using “Clean” does not remove all the relevant files.

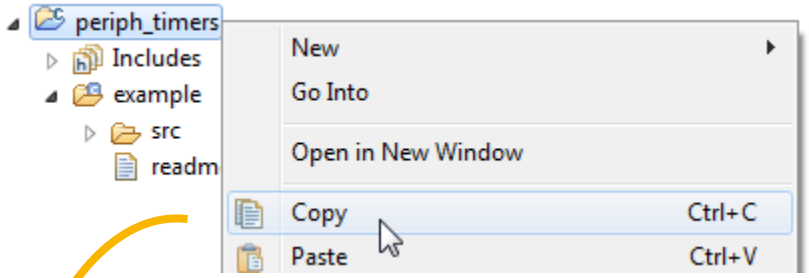
Open up the periph_timers project using the “expanders”



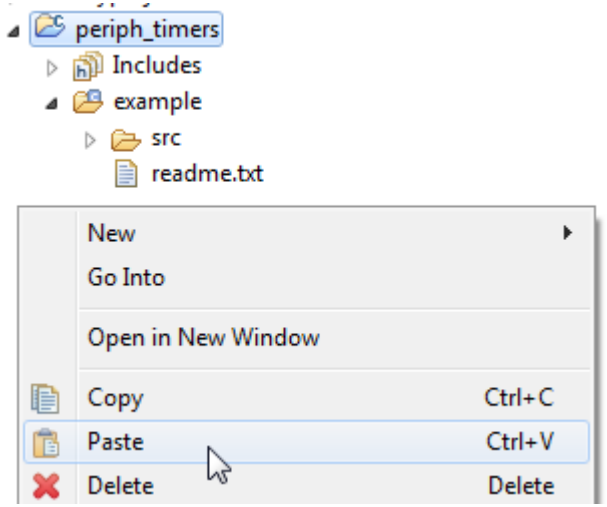



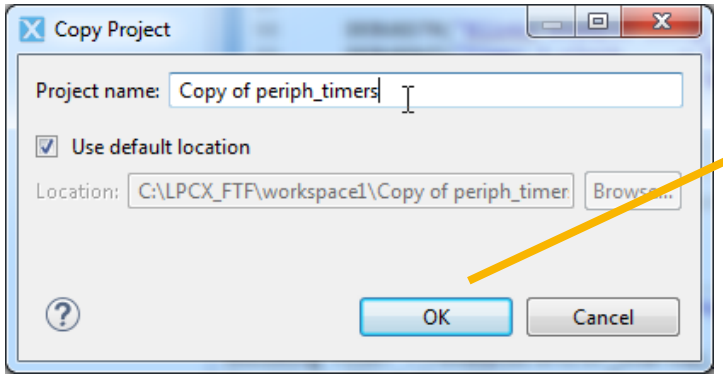

Select the Debug (and Release) directory, plus the .launch files, then right click and choose Delete

# Copying Projects (Cont'd)

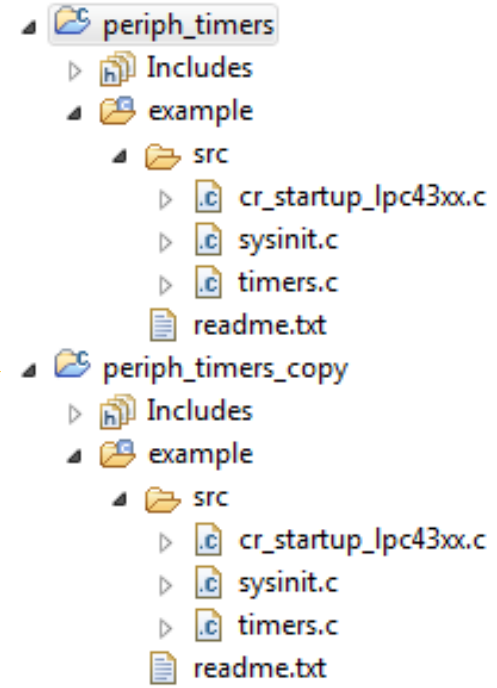



Now right-click and copy the "clean" project, then right click and paste




Enter the name of the new project
e.g. periph_timers_copy




New project created in workspace

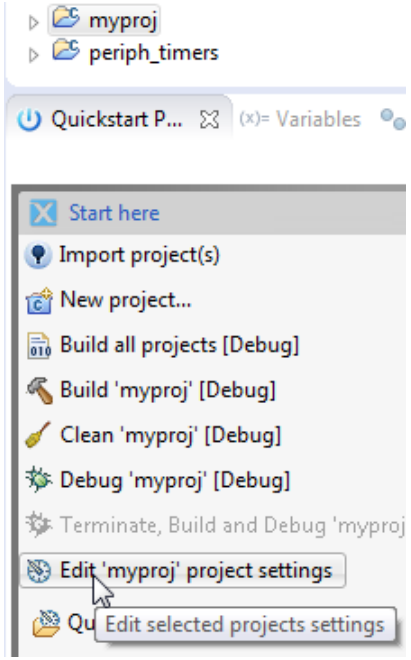
# Changing Project Settings

Open the Properties for the “myproj” project via the Quickstart Panel

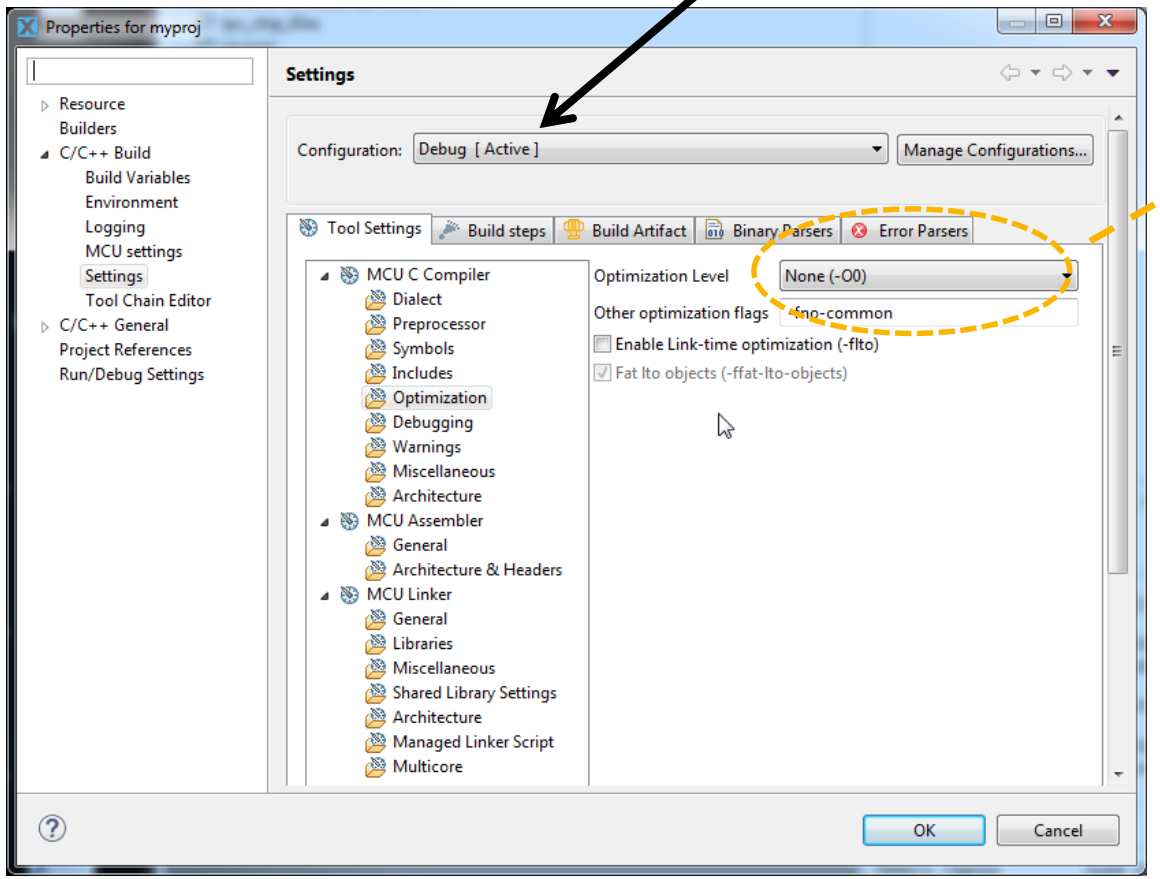
Switch to Compiler Optimization settings

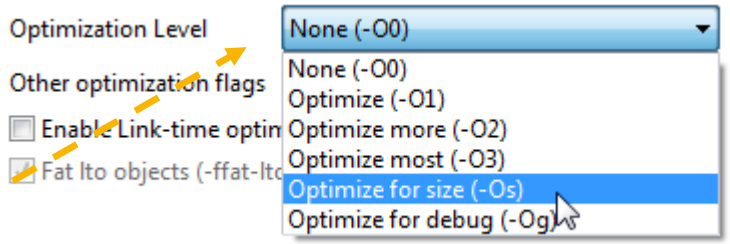
Settings are for specified Build Configuration

Change optimization level, click OK, then trigger a build




Or use “Properties” entry on Project Explorer right-click menu or press Alt-Menu (Windows)




## Default Build at -O0

```
Building target: myproj.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\LPCX_FTF\workspace1\lpc_b
Memory region         Used Size  Region Size  %age Used
      MFlashA512:        6516 B        512 KB      1.24%
      MFlashB512:           0 GB       512 KB      0.00%
        RamLoc32:          16 B         32 KB      0.05%
```

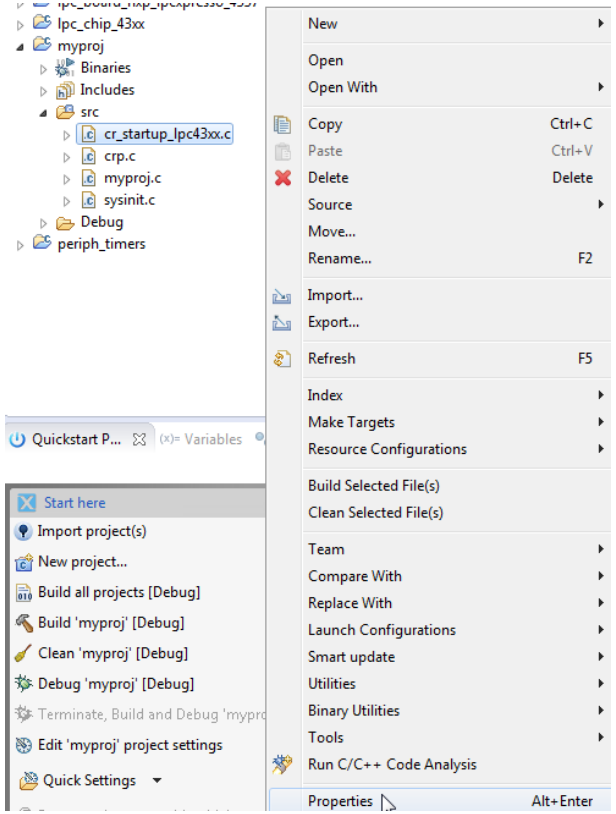
## Project changed to -Os

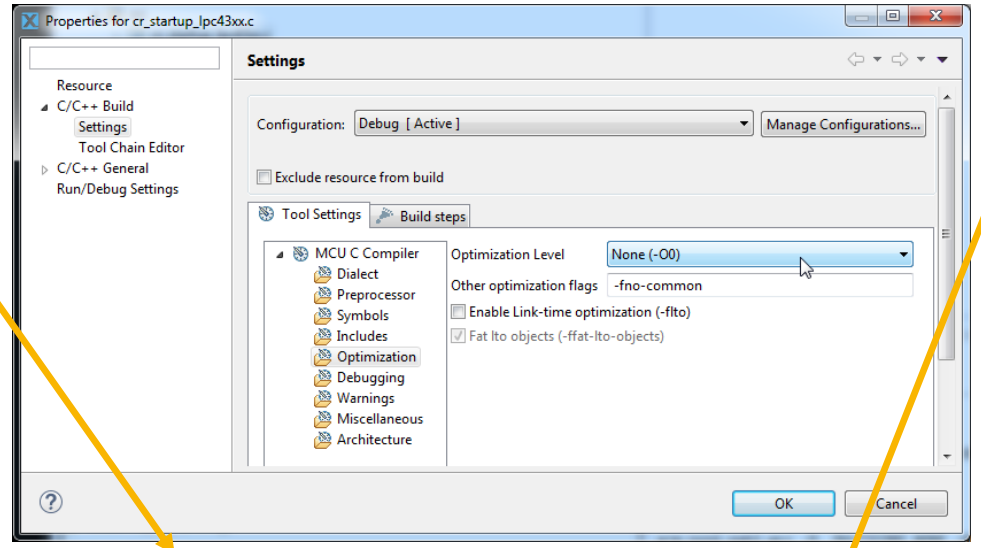
```
Building target: myproj.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\LPCX_FTF\workspace1\lpc
Memory region         Used Size  Region Size  %age Used
      MFlashA512:        6424 B        512 KB      1.23%
      MFlashB512:           0 GB       512 KB      0.00%
        RamLoc32:          16 B         32 KB      0.05%
```

Note : We have only changed the application, not library projects

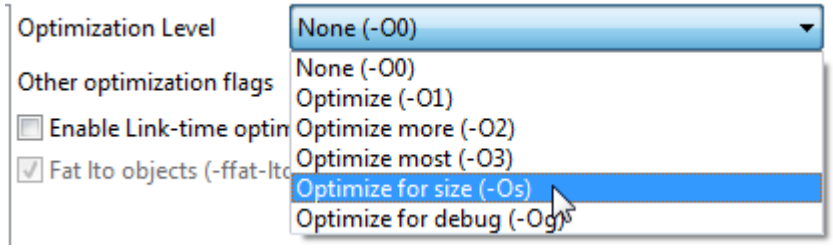
# Per-file Settings

First reconfigure the project back to build -O0
Then select the startup file, then “Properties” in the right-click menu

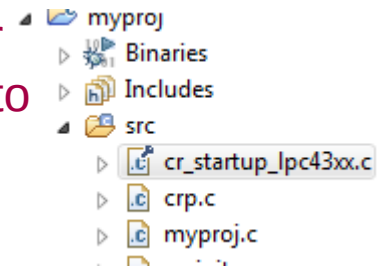



Now change the optimization level for this file to -Os




Note file decorator icon has updated to show a change to the file properties




Now rebuild the project and compare to the default settings

Default Build at -O0

```
Building target: myproj.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\LPCX_FTF\workspace1\lpc_b
Memory region         Used Size  Region Size  %age Used
      MFlashA512:        6516 B        512 KB      1.24%
      MFlashB512:           0 GB       512 KB      0.00%
        RamLoc32:          16 B        32 KB       0.05%
```

Just startup changed to -Os

```
Building target: myproj.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\LPCX_FTF\workspace1\lpc
Memory region         Used Size  Region Size  %age Used
      MFlashA512:        6440 B        512 KB      1.23%
      MFlashB512:           0 GB       512 KB      0.00%
        RamLoc32:          16 B        32 KB       0.05%
```

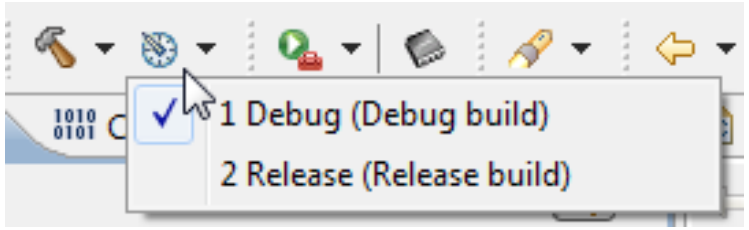
Can change per-directory settings in the same way

# Debug and Release Build Configurations

- New projects are created with two default build configurations, with each configuration having separate compiler/assembler/linker settings.

- Debug
  - Code is compiled to give high level of source level debugging functionality (-O0)

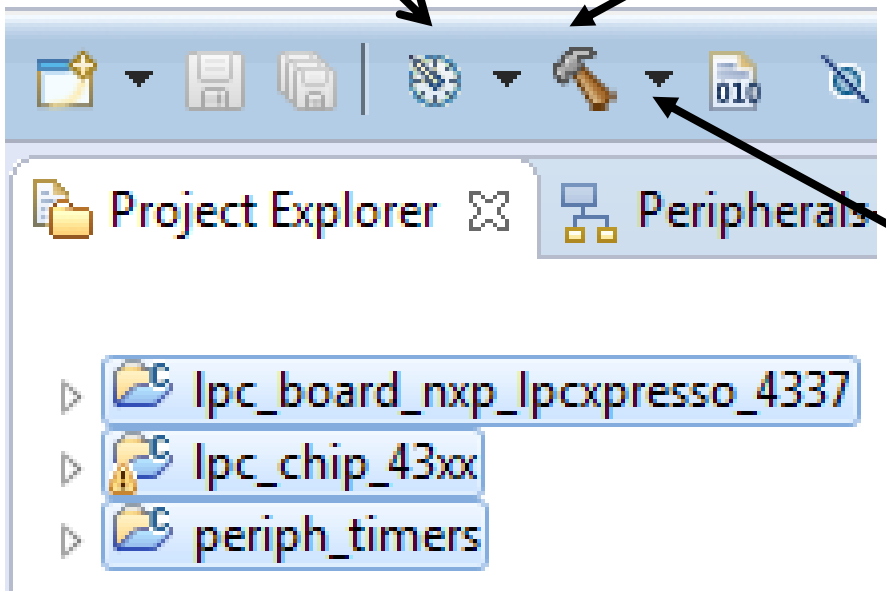
- Release
  - Code is compiled optimized for space (-Os)
  - Provides smallest code size, though with reduced debug view



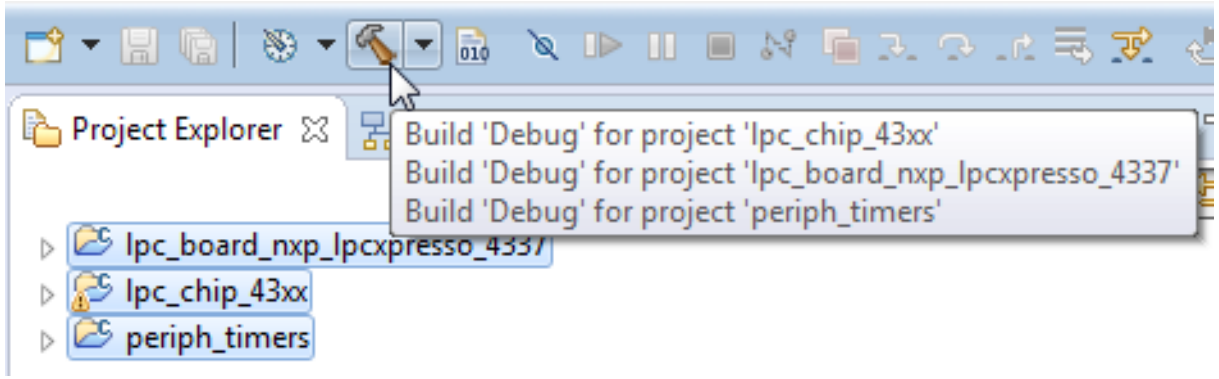

- Can easily switch the build configuration being used for the currently selected project(s)
  - Generally need to switch library projects as well as application project

# Switching between Debug and Release Configurations

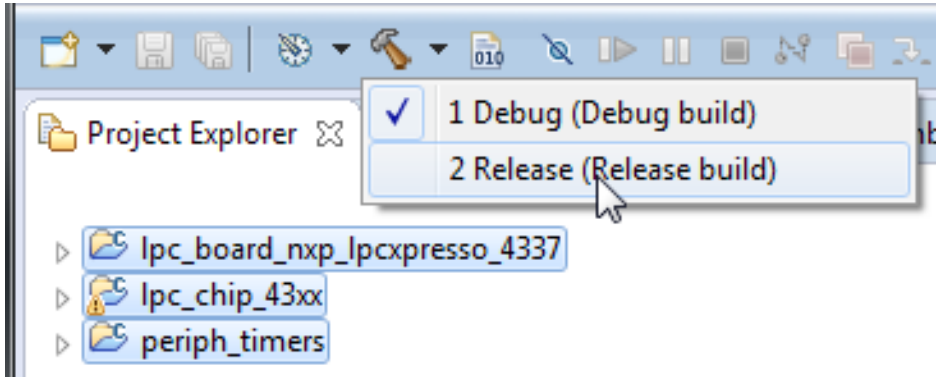
Sets build configuration of selected project(s)



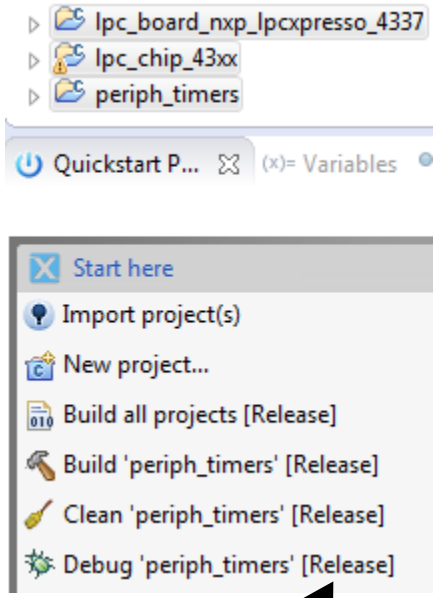

Click on the button directly to trigger build using current build configuration



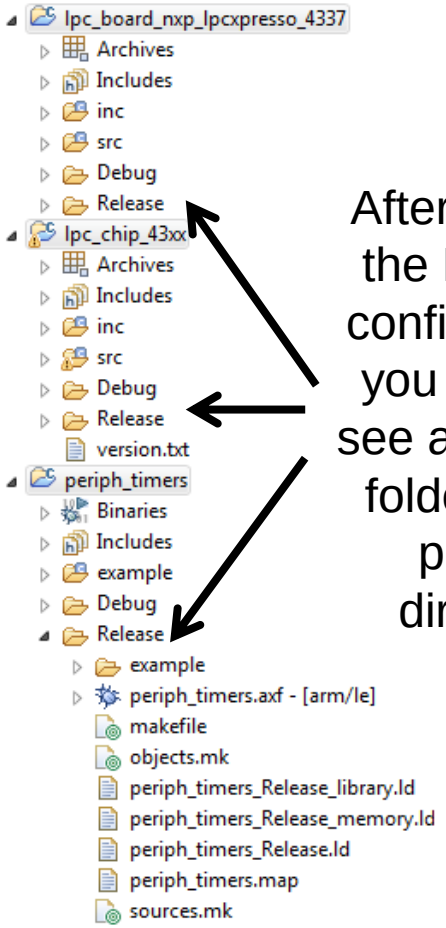

Click on “drop down” to set build configuration of selected projects and then trigger build



# Result of Changing Build Configuration




After changing the current build configuration setting, the Quickstart Panel will update to match




After building the Release configuration, you will now see a Release folder in the project directory

```
Building target: periph_timers.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\LPCX_FTF\workspace1\lpc_chip_43xx\Debug"
Finished building target: periph_timers.axf

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "periph_timers.axf"; # arm-none-eabi-objcopy -v -O binar
   text    data     bss     dec     hex filename
  17744       0     352   18096    46b0 periph_timers.axf
```

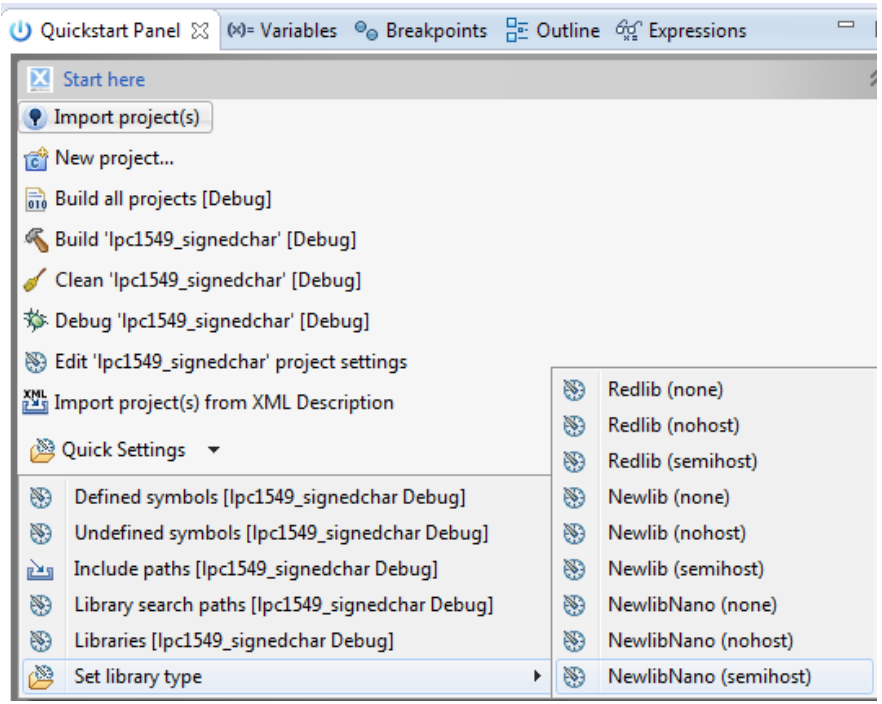
Now build the Debug and Release configurations of your projects and compare the code size

[ You may need to do a “Clean” to force a rebuild ]

```
Building target: periph_timers.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\LPCX_FTF\workspace1\lpc_chip_43xx\Release"
Finished building target: periph_timers.axf

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "periph_timers.axf"; # arm-none-eabi-objcopy -v -O binary
   text    data     bss     dec     hex filename
  14144       0     352   14496    38a0 periph_timers.axf
```

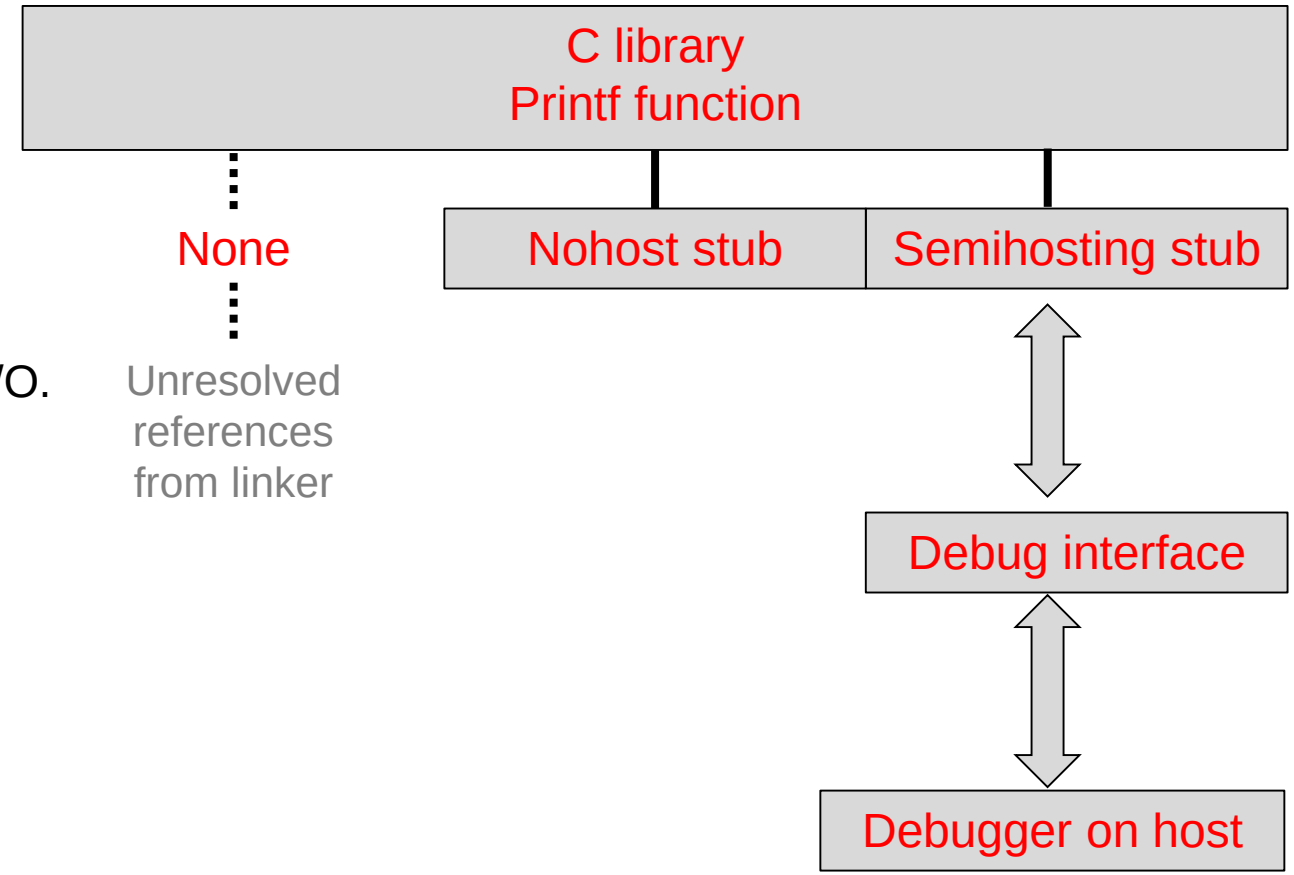
# C/C++ Library Selection

- C projects
  - Default to Redlib
  - C90 library, with some C99 extensions
  - Optimized for code size
  - Select use of integer printf in wizard
- C++
  - Default to Newlib
  - Provides C++ support, plus full C99
  - Can switch C projects to use Newlib if required
- LPCXpresso also supports “Newlib-Nano”
  - Code size optimized version of Newlib
  - Can switch C or C++ projects to use this
  - Integer only printf by default – enable floating point in Linker options
  - http://www.lpcware.com/content/faq/lpcxpresso/newlib-nano-support



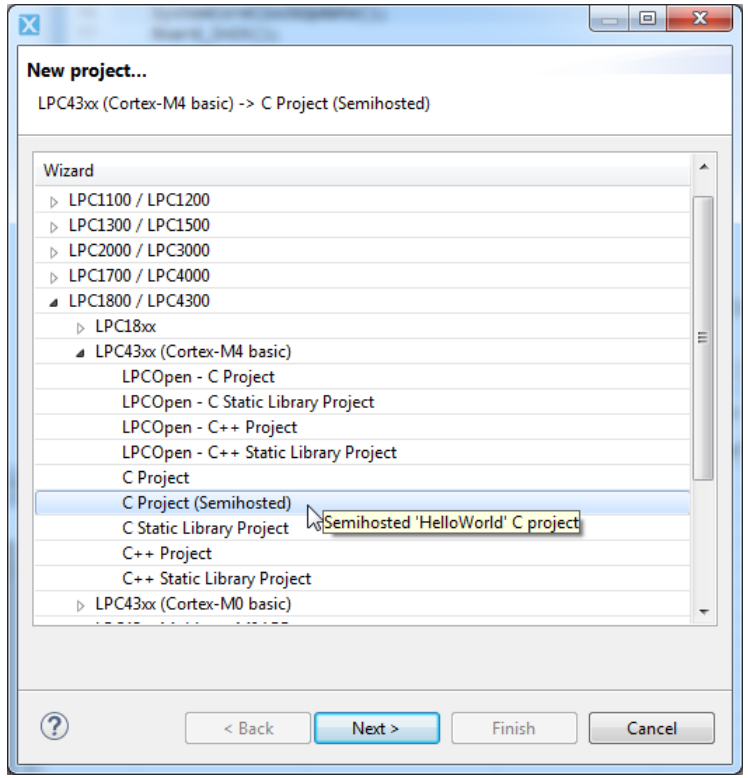
# Library Variants

- Libraries are provided in number of variants, with different underlying “stub” providing support functions:
  - None
    - Smallest footprint. Excludes low-level file I/O
    - For Newlib, excludes memory handling functions
  - Nohost
    - Provides memory handling functions and some file I/O.
    - However, it assumes no host, and so file I/O will do nothing
  - Semihost
    - Full functionality.
    - I/O resources are on the host side.

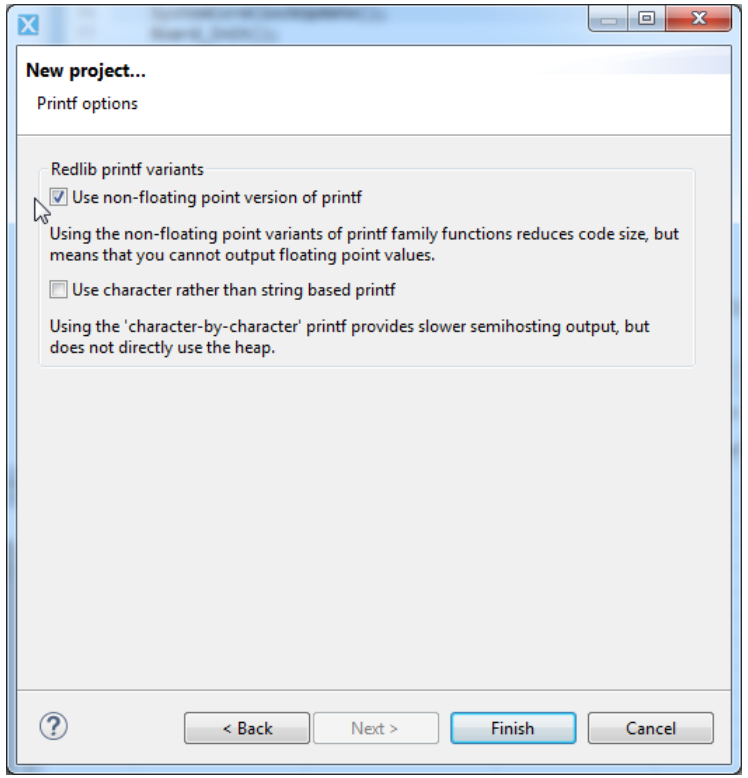



- More C library information at:
  - http://www.lpcware.com/faq/c-library

# Creating a Semihosted Project



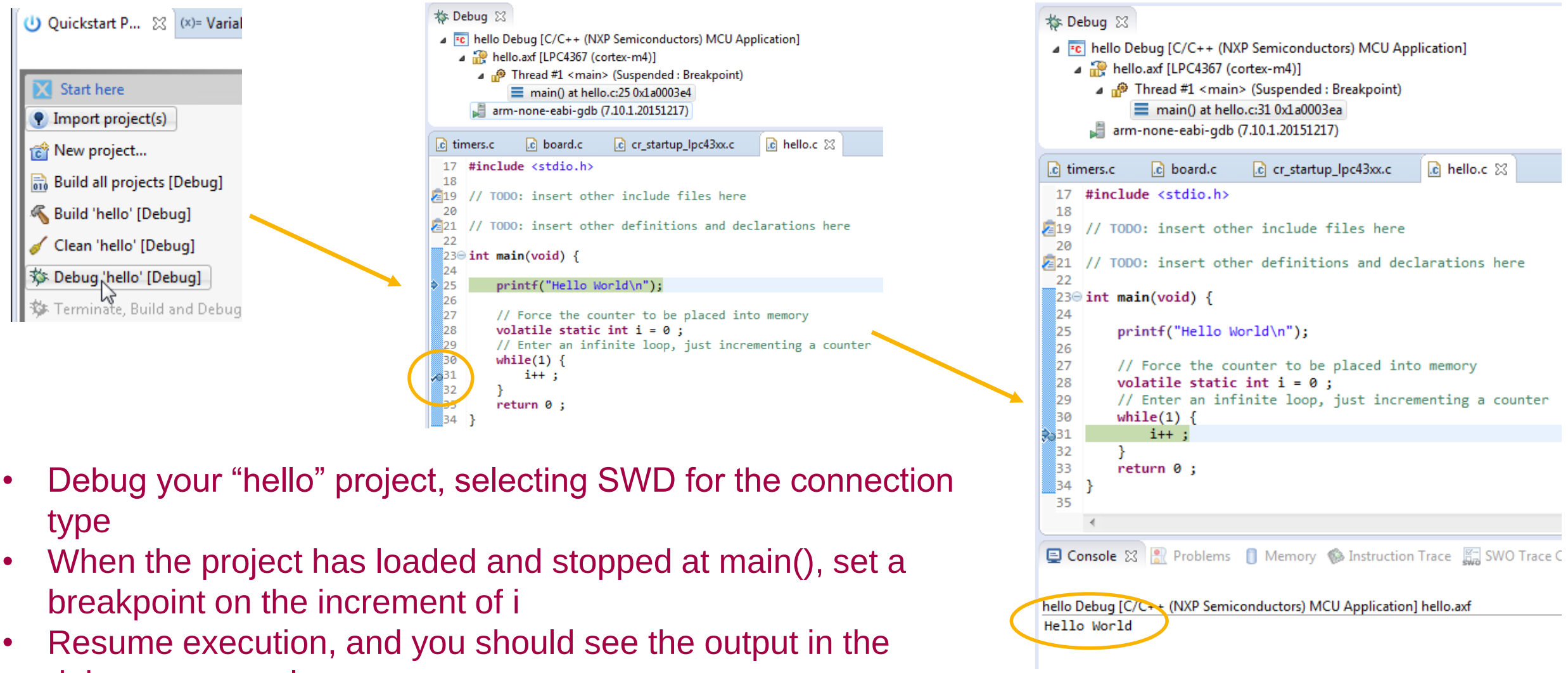

- Give the project the name “hello”
- Choose LPC4367 for the target MCU
- Click Next through the following few pages without changing default options, until you reach the “printf options” page …




Start the New Project Wizard for an LPC43xx (Cortex-M4 basic) and select C Project (Semihosted)

Select the non-floating point (ie integer only) printf option, to reduce code size, then click Finish

# Running a Semihosted Project




- Debug your “hello” project, selecting SWD for the connection type
- When the project has loaded and stopped at main(), set a breakpoint on the increment of i
- Resume execution, and you should see the output in the debugger console.

# Converting an LPCOpen Project to Use Semihosting

Semihosting causes CPU to drop into debug state

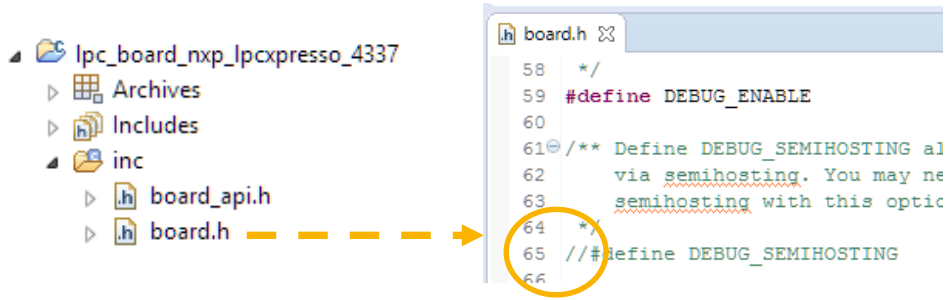
- Interrupts not serviced until semihosting operation completes.
- Code will fail if debugger not connected

Most LPCOpen packages redirect output to UART by default

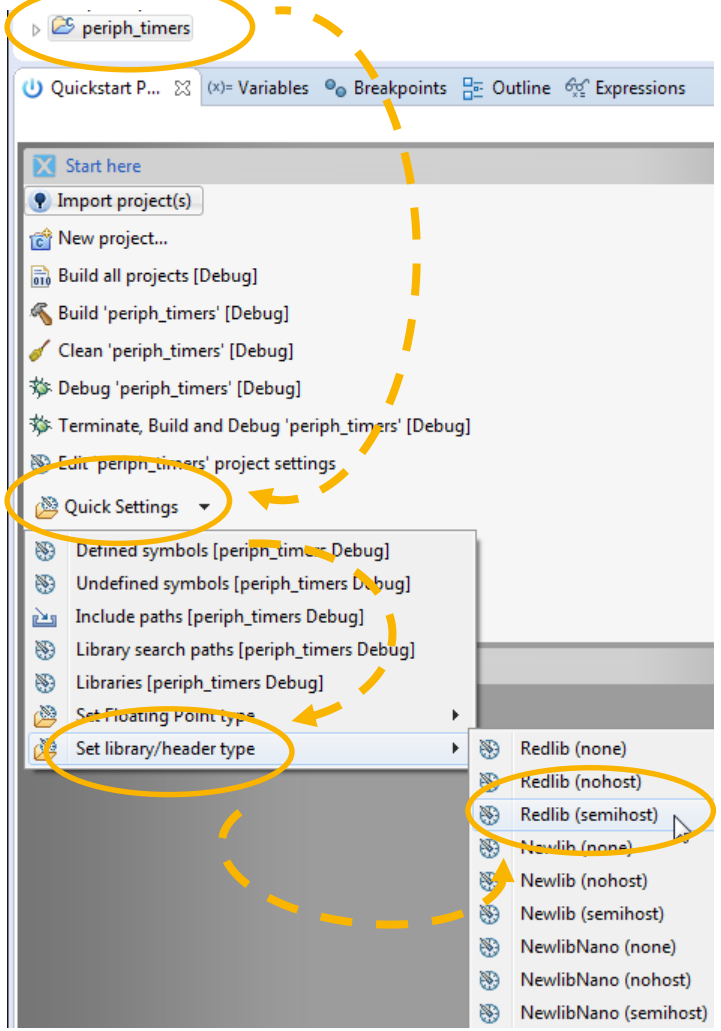
- Need to reconfigure board library and project to use semihosting instead

Configure board library to use Semihosting

- Edit board.h
- Uncomment the statement
  #define DEBUG_SEMIHOSTING

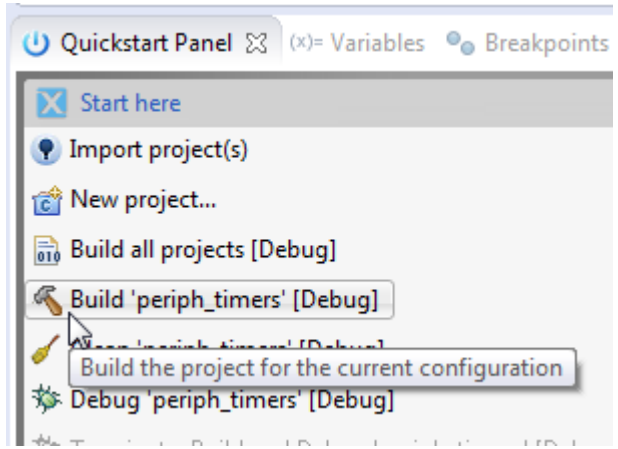


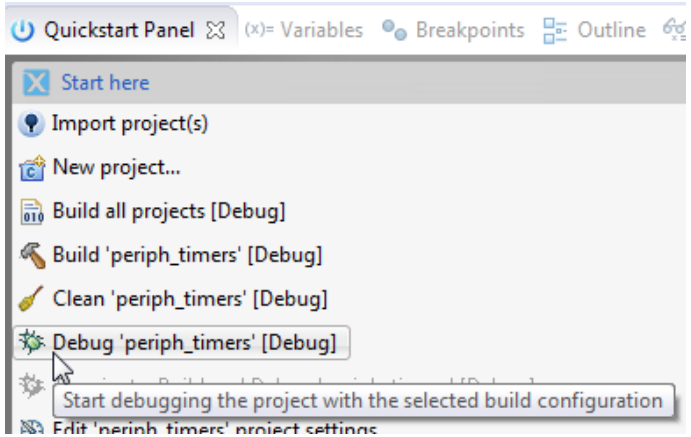

Configure periph_timers project to link against the Semihosting library variant

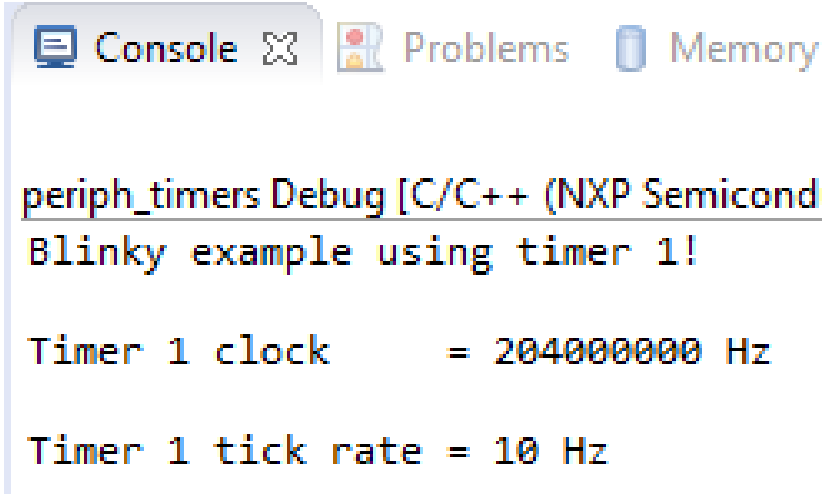


# Running Semihosted LPCOpen Project

- Now when you rebuild, debug and run the periph_timers example, you should see the introductory text printed out to the LPCXpresso IDE console



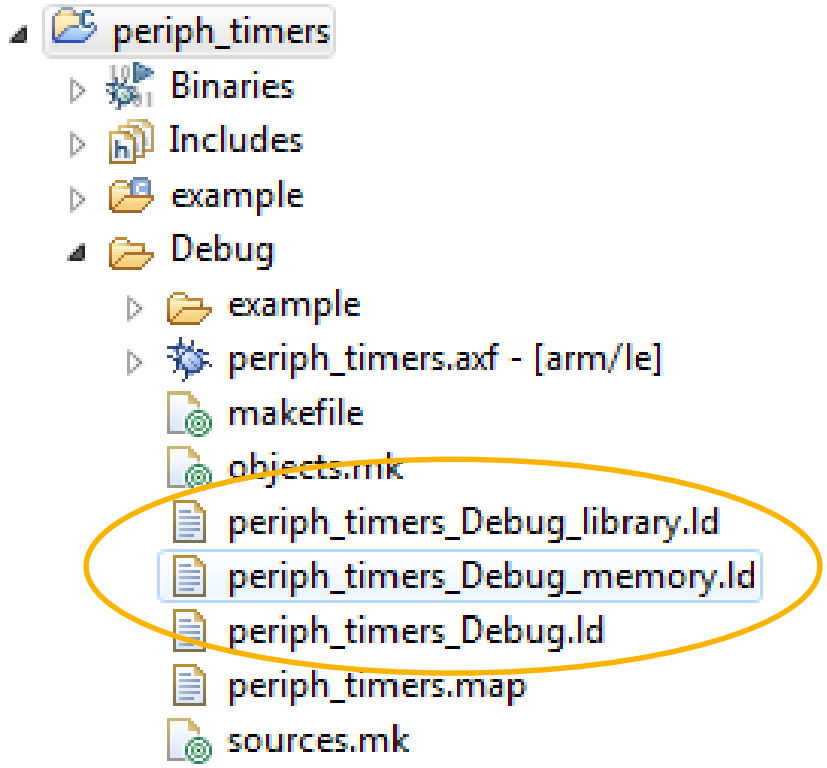



# Linker Scripts

- By default, use of “managed linker scripts” is enabled
  - Linker scripts automatically generation based on built-in knowledge of MCU memory map
  - Take into account changes made in memory config editor and library variant
- Can modify script that is generated by providing modified “Freemarker” template scripts locally in project (starting point in \Wizards\linker).
- Can also modify default locations used for specific code, data and bss by decorating source using set of predefined macros:

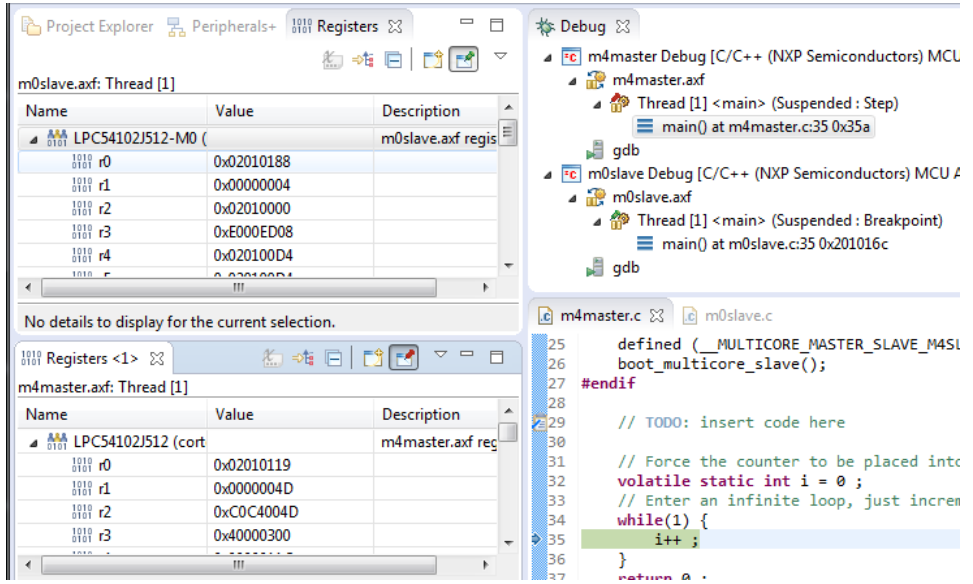
```
#include <cr_section_macros.h>
__BSS(RAM2) char bss_buffer[128];
__RAMFUNC(RAM) (void)foo(void) {…
```

- Can disable managed linker scripts if required and provide your own…
  - Properties -> C/C++ Build -> Settings -> MCU Linker -> Target
- See FAQs for more details



# MULTICORE SUPPORT

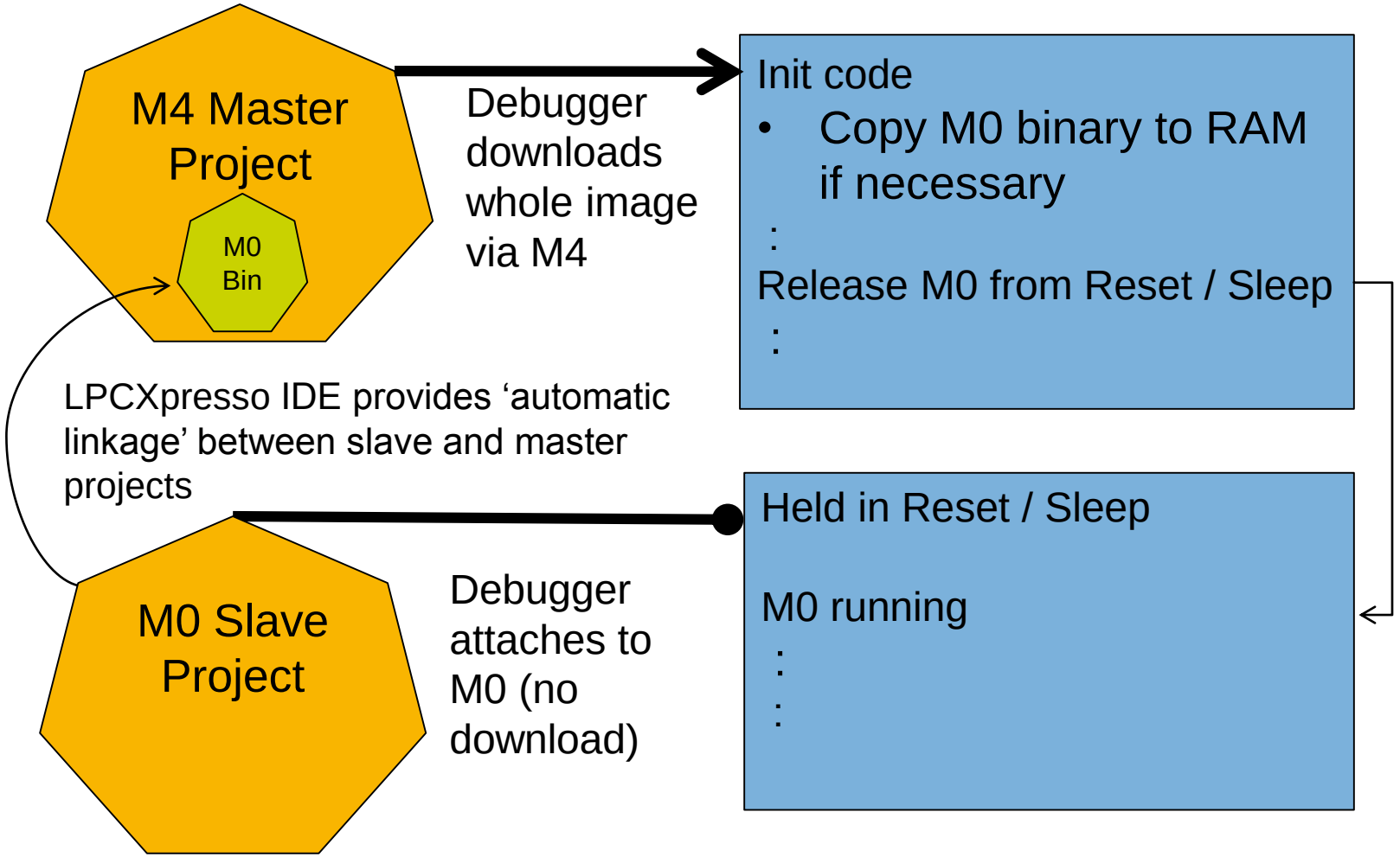
# Multicore Introduction




- The LPC43xx and LPC541xx MCUs have loosely couple multiple CPUs which share memory map and peripherals
- LPC43xx
  - LPC432x / 433x / 435x – 1 CM4, 1 CM0 (M0APP)
  - LPC4370 / LPC4367– 1 CM4, 2 CM0 (M0APP and M0SUB)
  - http://www.lpcware.com/content/faq/lpcxpresso/lpc43xx-multicore-apps
  - Note : LPC43xx LPCOpen packages do not currently ship with LPCXpresso style multicore projects, but a converted example can be found at:
    - http://www.lpcware.com/content/forum/lpcxpresso-multiprocessor-example
- LPC54102 / LPC54114
  - 1 CM4, 1 CM0+
  - http://www.lpcware.com/content/faq/lpcxpresso/lpc541xx-multicore-apps
  - LPCOpen does ship LPCXpresso style multicore example projects

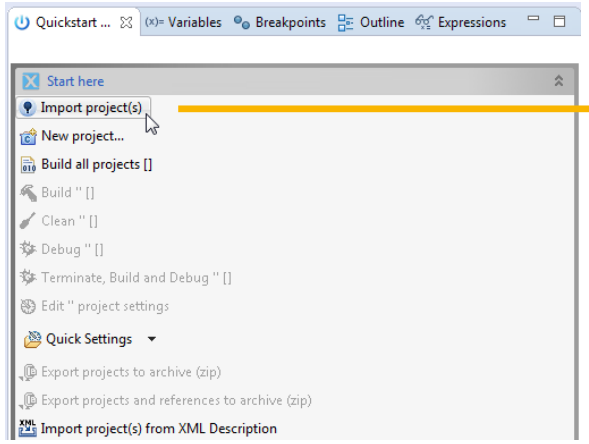
# Multicore Project Setup and Debug

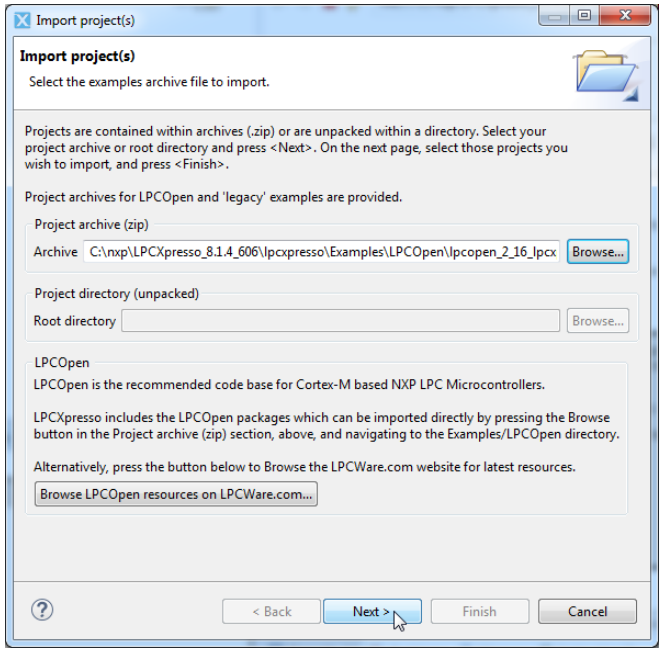
- One application project per CPU
- Links Master (M4) project to Slave (M0) project(s)
  - Master pulls in binary from Slave(s) to create single image to download
- Parallel debugging of all projects
  - Start Master debug connection first,
    then attach to Slave(s)
- Multiple instances of views
- Can pause / start CPUs in parallel (IDE not in hw)



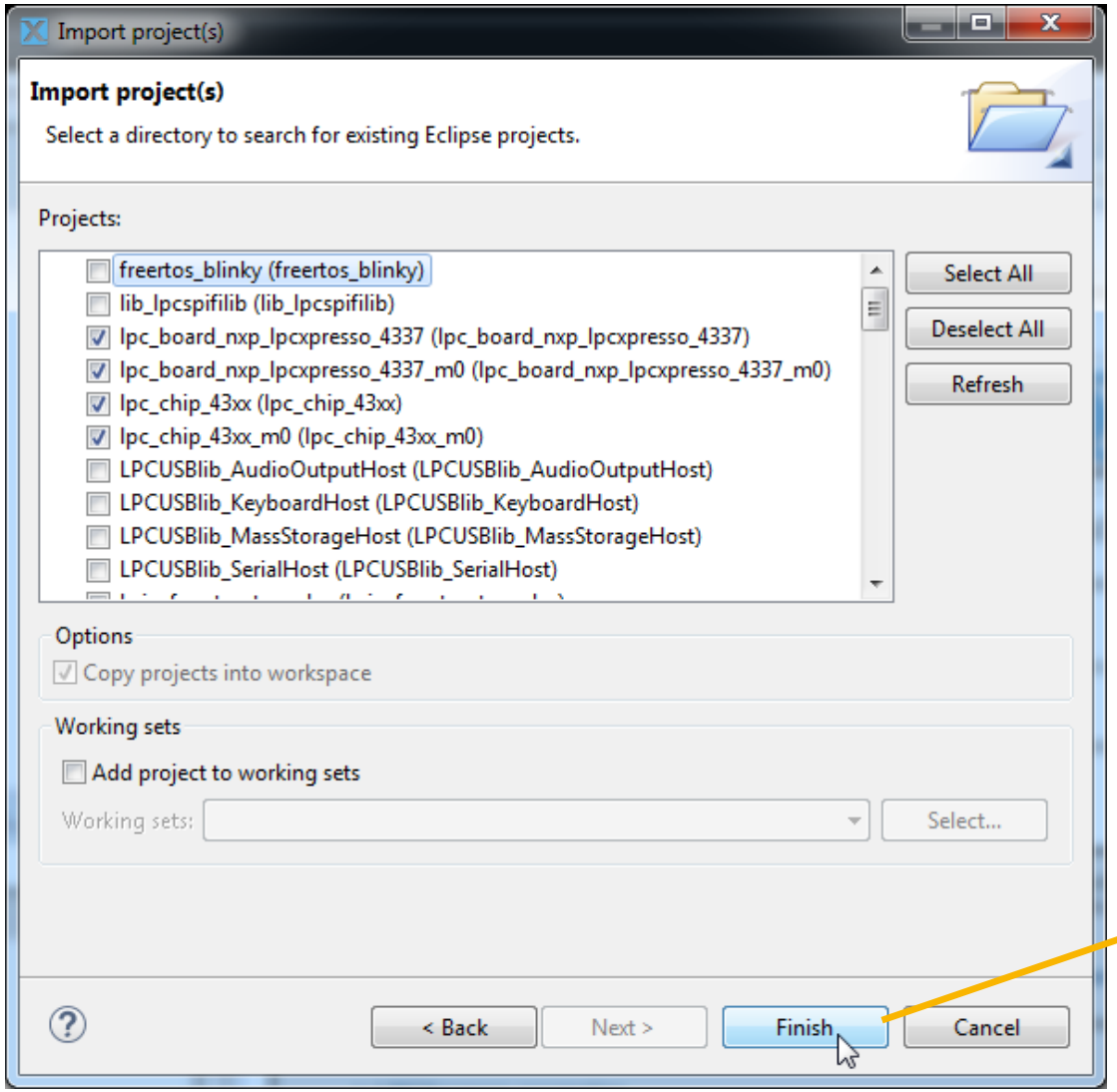
# Run Import Projects

- Use “File -> Switch workspace” menu to restart LPCXpresso IDE in a fresh workspace
  - Suggest C:\LPCX_FTF\workspaceMC
- Quickstart -> Import project(s) -> Project Archive, select ZIP, then click Next
  - `C:\nxp\LPCXpresso_8.1.4_606\lpcxpresso\Examples\LPCOpen`
    - `lpcopen_2_16_lpcxpresso_nxp_lpcxpresso_4337.zip`

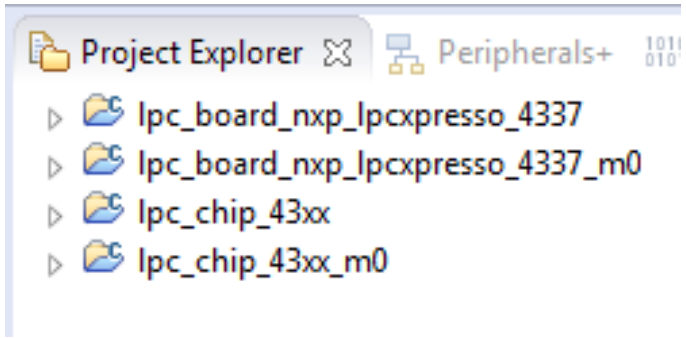


# Import CM4 and CM0 LPCOpen Library Projects

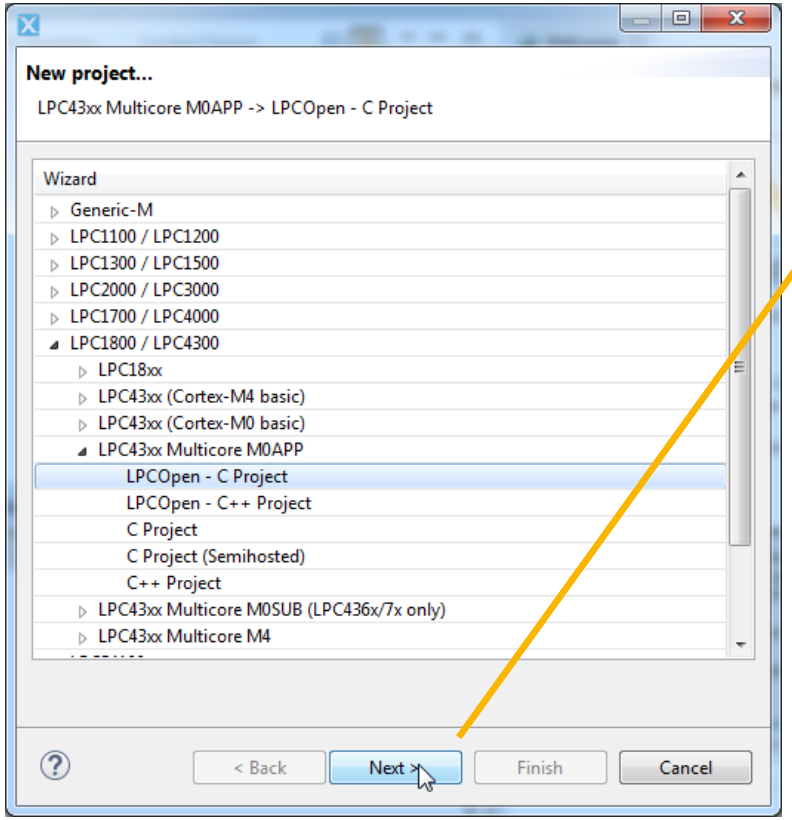


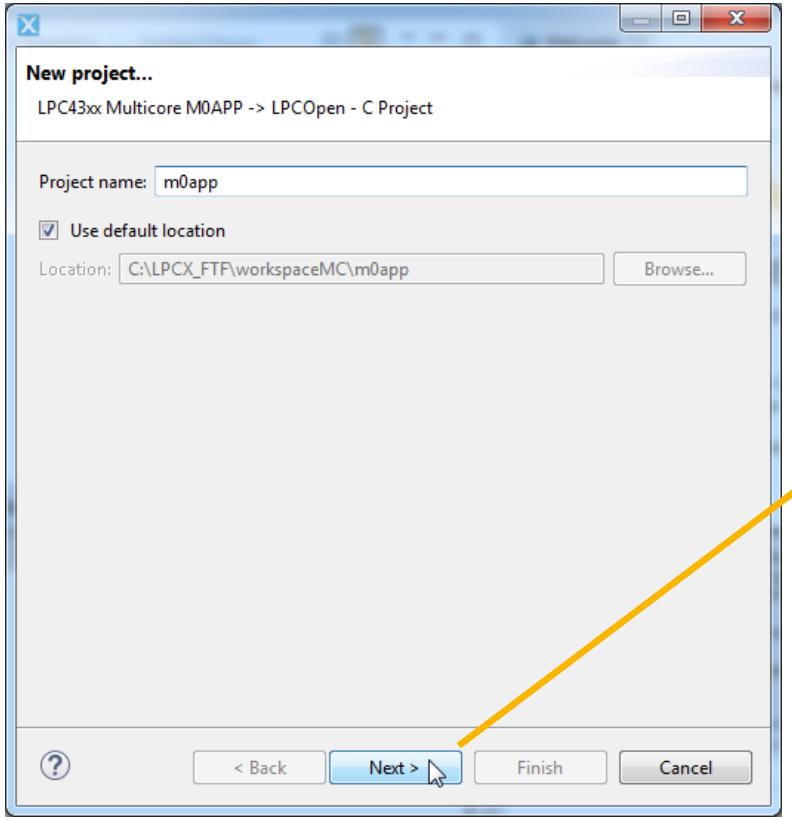

- Use the “Deselect All” option, then explicitly select required projects from scrollable list, then click “Finish”
  - `lpc_board_nxp_lpcxpresso_4337`
  - `lpc_board_nxp_lpcxpresso_4337_m0`
  - `lpc_chip_43xx`
  - `lpc_chip_43xx_m0`

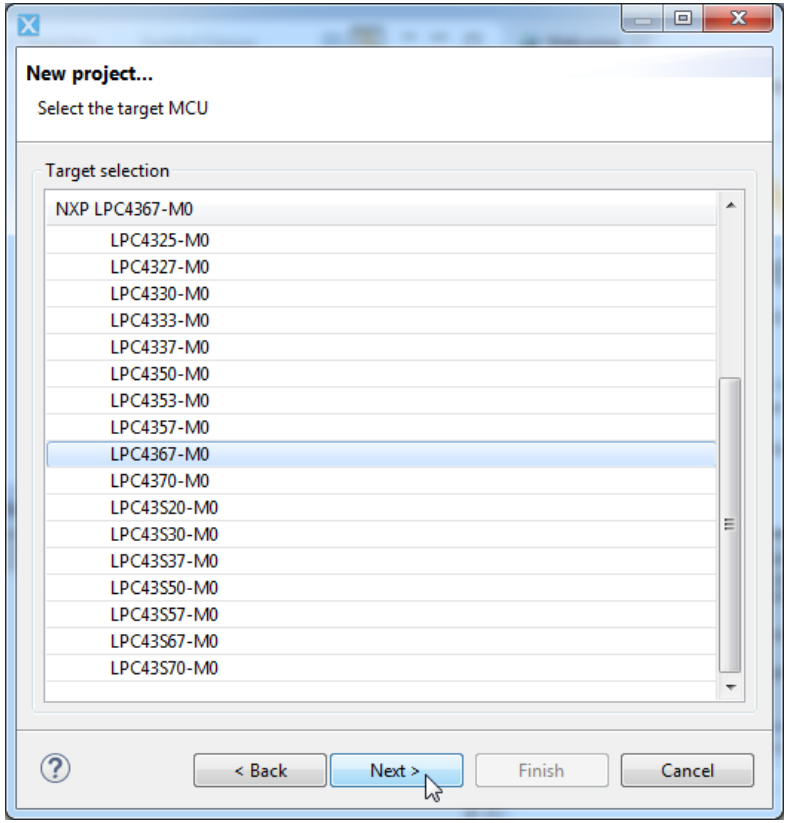


# Create Slave Project

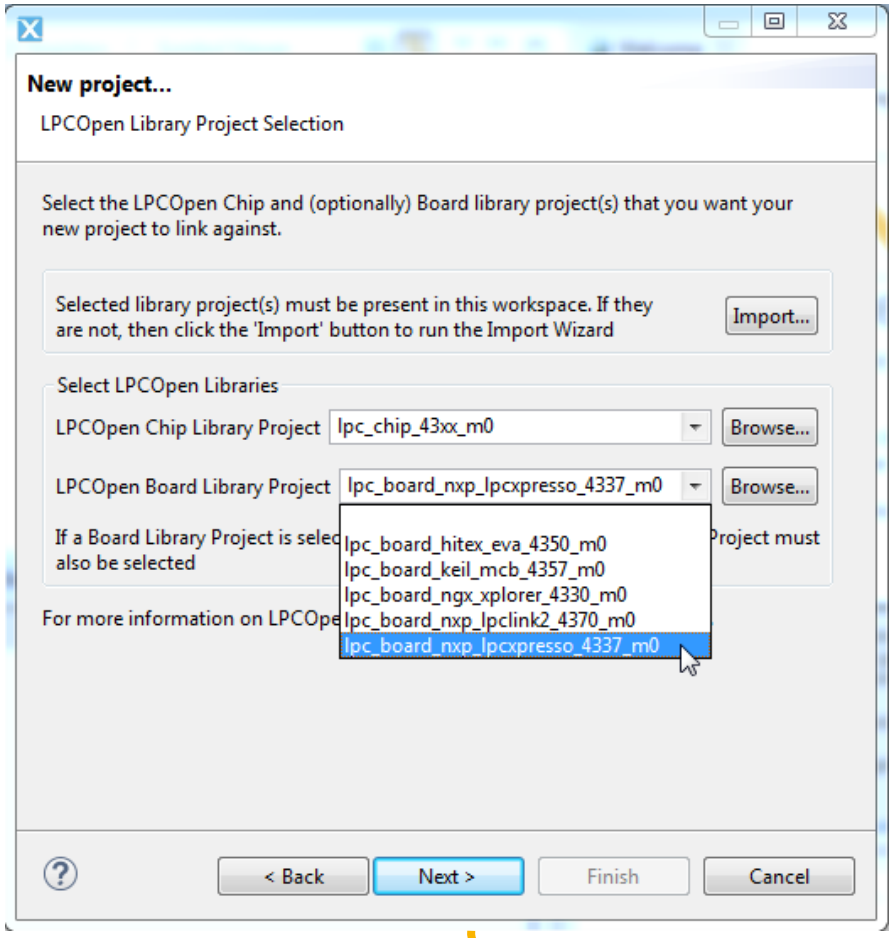
- Quickstart -> New project -> LPC1800/4300 -> LPC43xx Multicore M0App



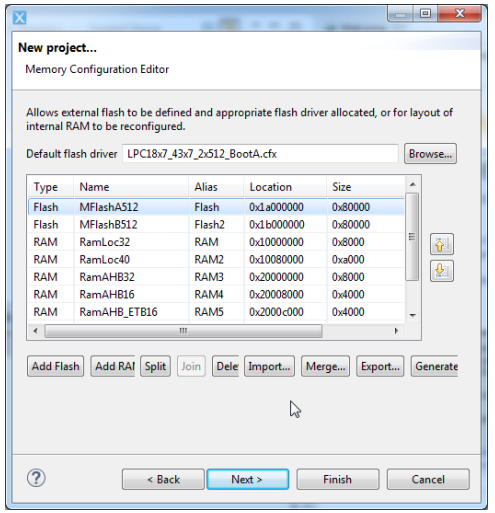



# Select LPCOpen Library Projects

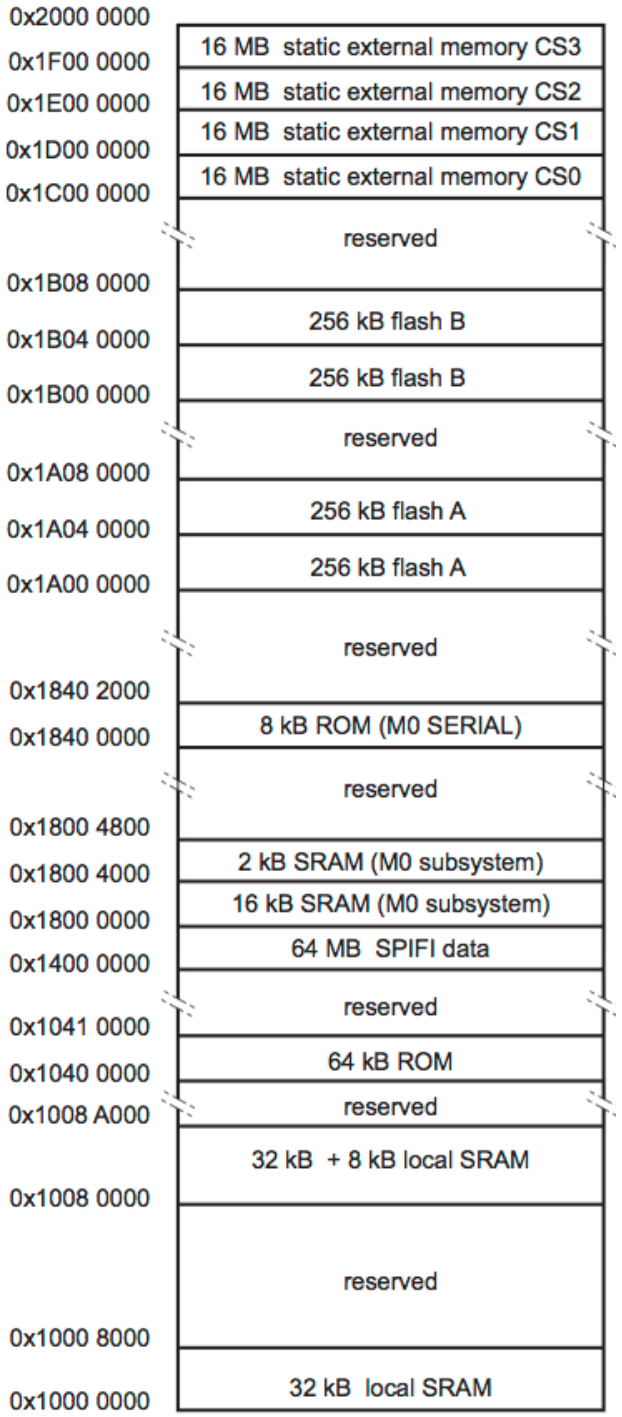


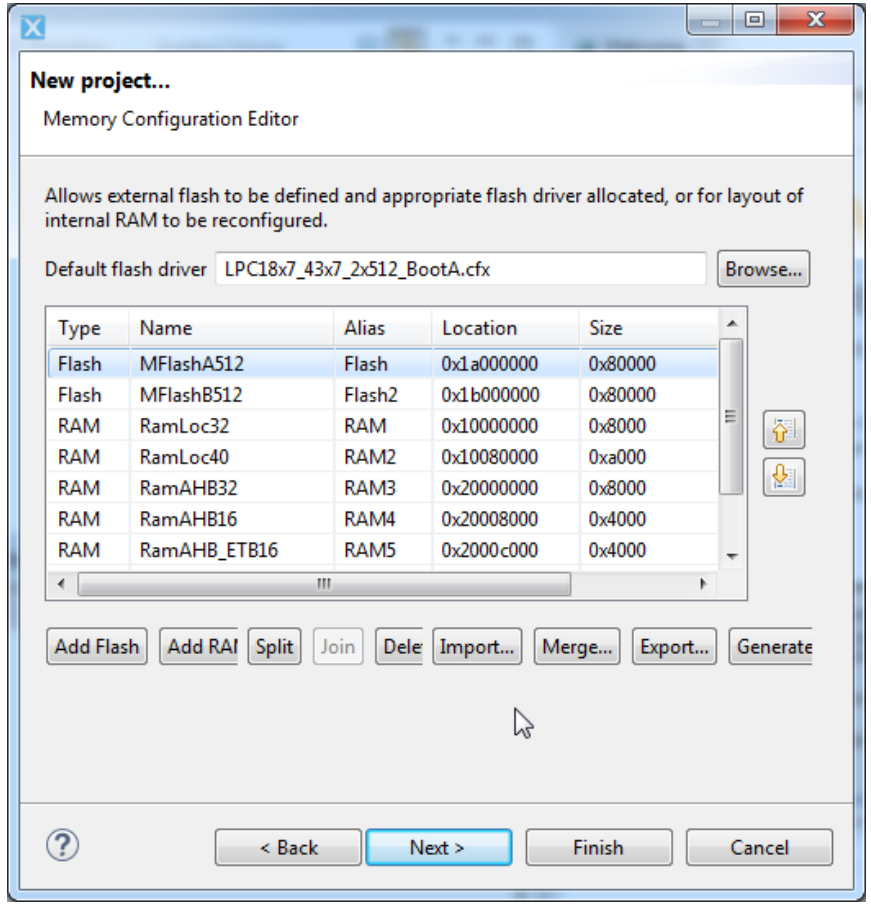

- Select the LPCOpen Chip and Board libraries that will be linked against
  - Chip Library automatically selected from MCU selection
  - Board Library : Needs to be selected based on your target board
    - For M0 cpu on LPCXpresso4367 board, select `lpc_board_nxp_lpcxpresso_4337_m0`
- Now click through the next few wizard pages until you reach the Memory Configuration Editor…

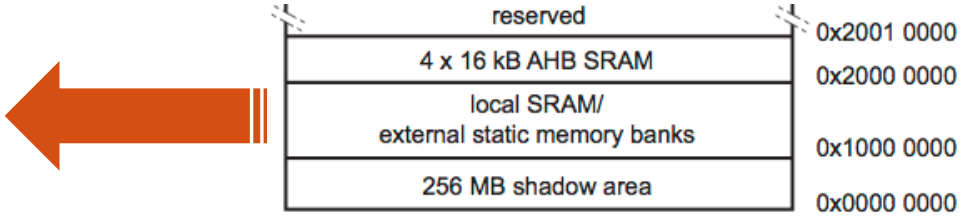


# LPC4367 Memory Layout

- Multiple blocks of memory in system, which can be accessed by all three CPUs if required
- Need to allocate blocks to store code+data for each CPU
  - Avoid bus contention
- This is the most complex part of setting up multicore projects!



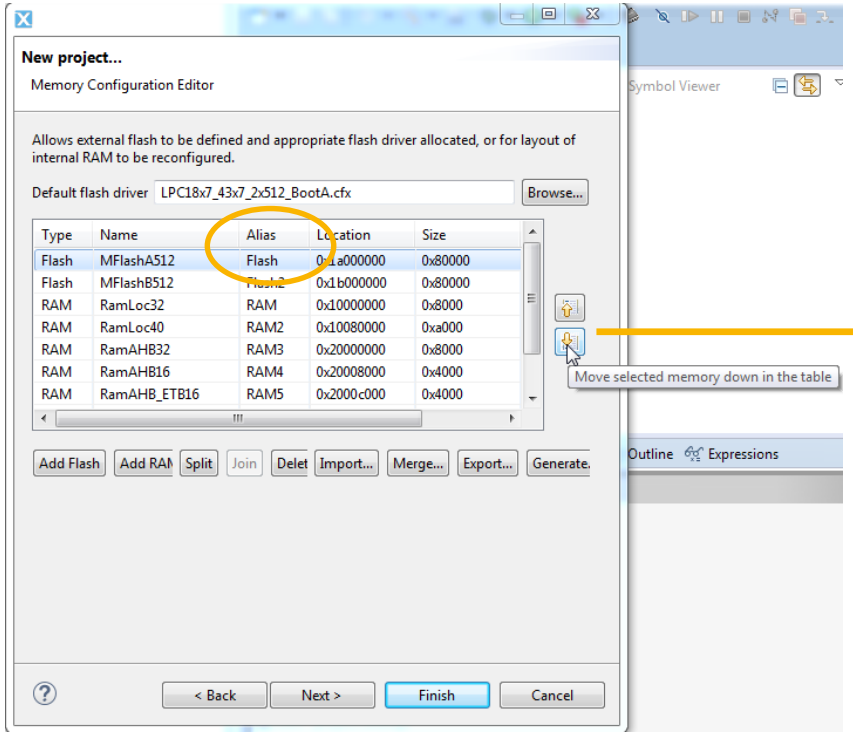



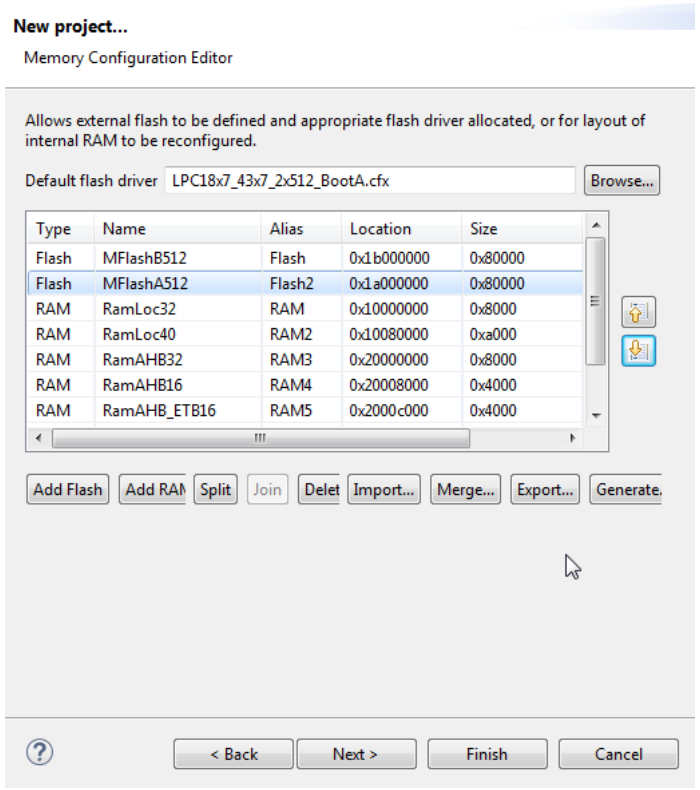



# Select MFlashB512 Flash Bank for Cortex-M0 Code

- Two banks of flash
  - Bank shown first in Memory Configuration Editor will be used by this project
  - Use one for CM4 code, one for CM0 code, so lets swap the order …

Highlight "Flash" in Alias column of table, then click on Down arrow

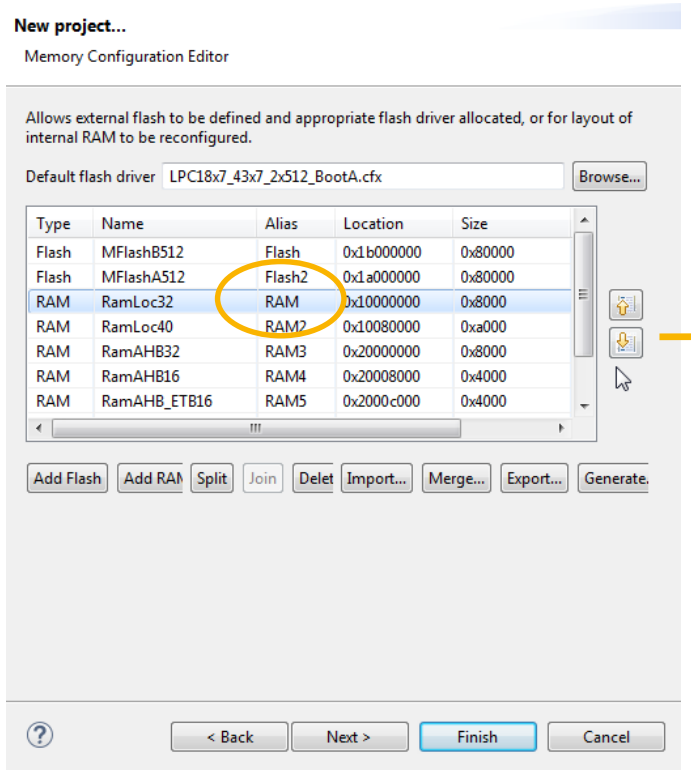


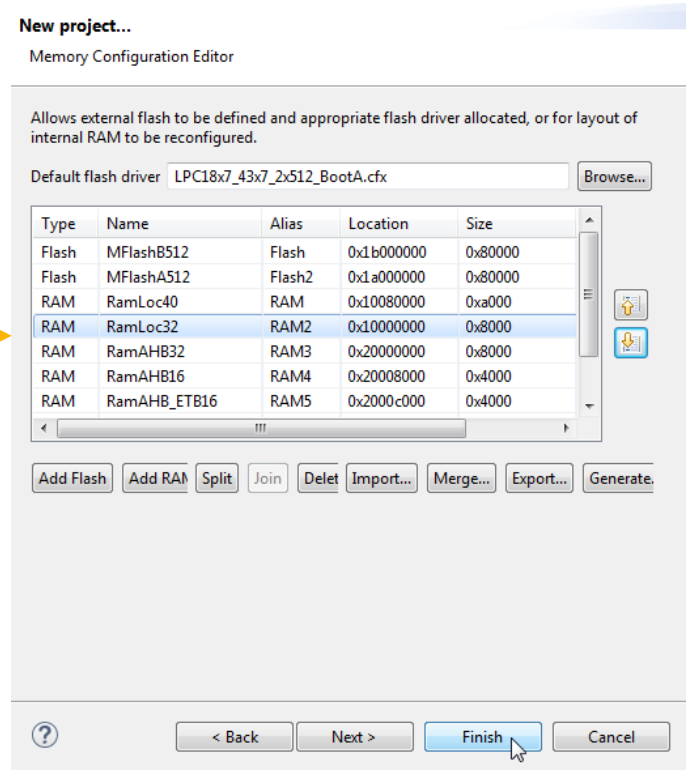



# Select RamLoc40 RAM Bank for Cortex-M0 Data

- Multiple banks of RAM
  - Bank shown first in Memory Configuration Editor will be used by default by application for data/heap/stack

Highlight "RAM" in Alias column of table, then click on Down arrow



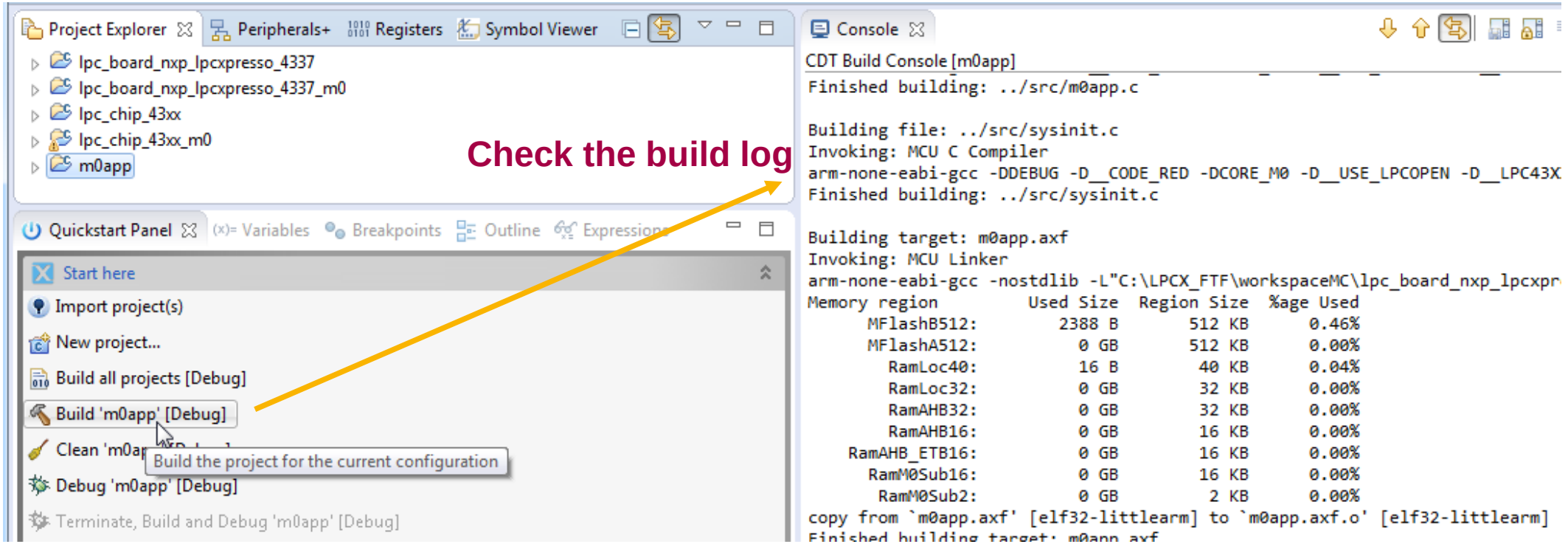




Then click on ***Finish***

# Check the M0App Project Builds

Select Project

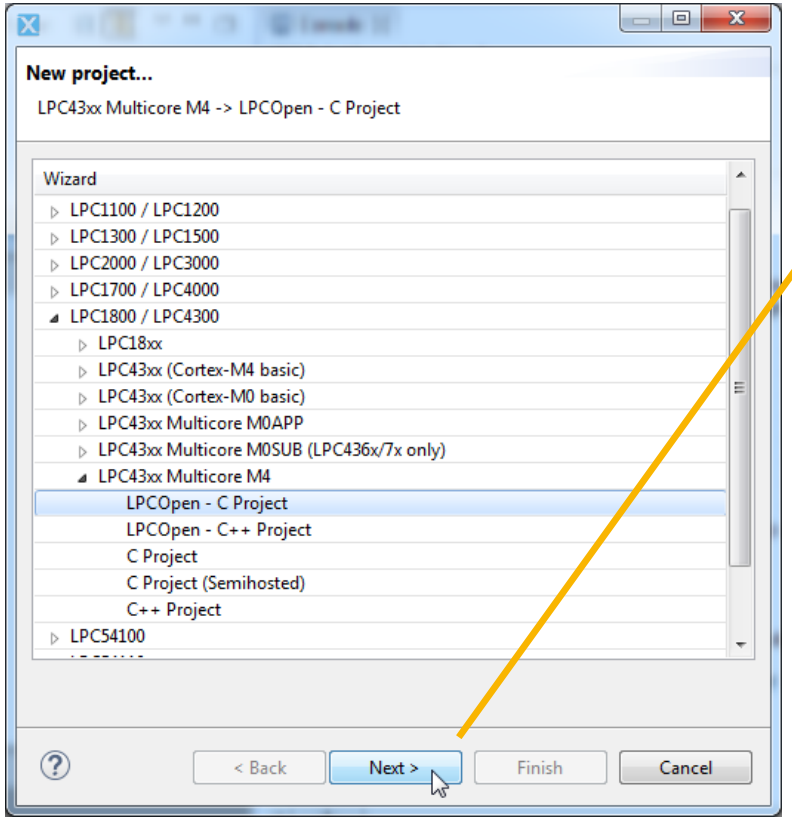
Build it

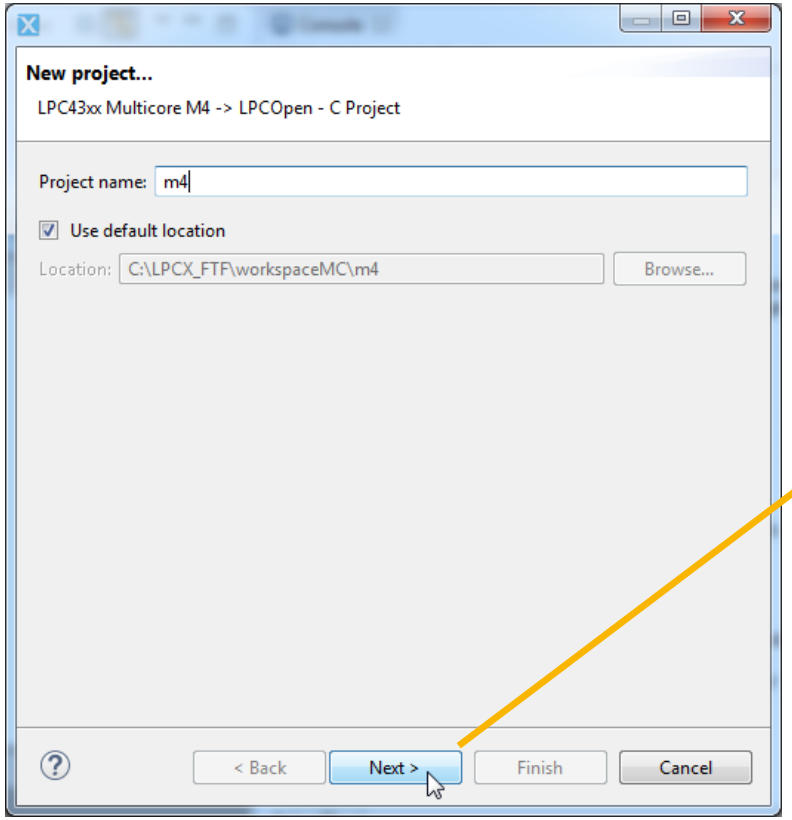


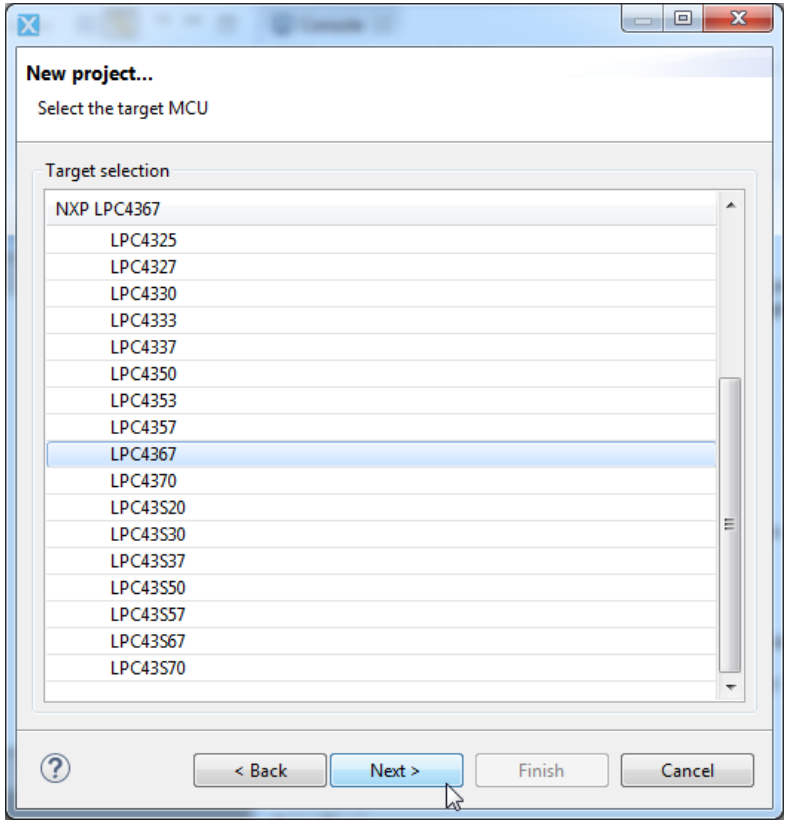

Note the memory blocks used for placing code/data

# Create Master Project

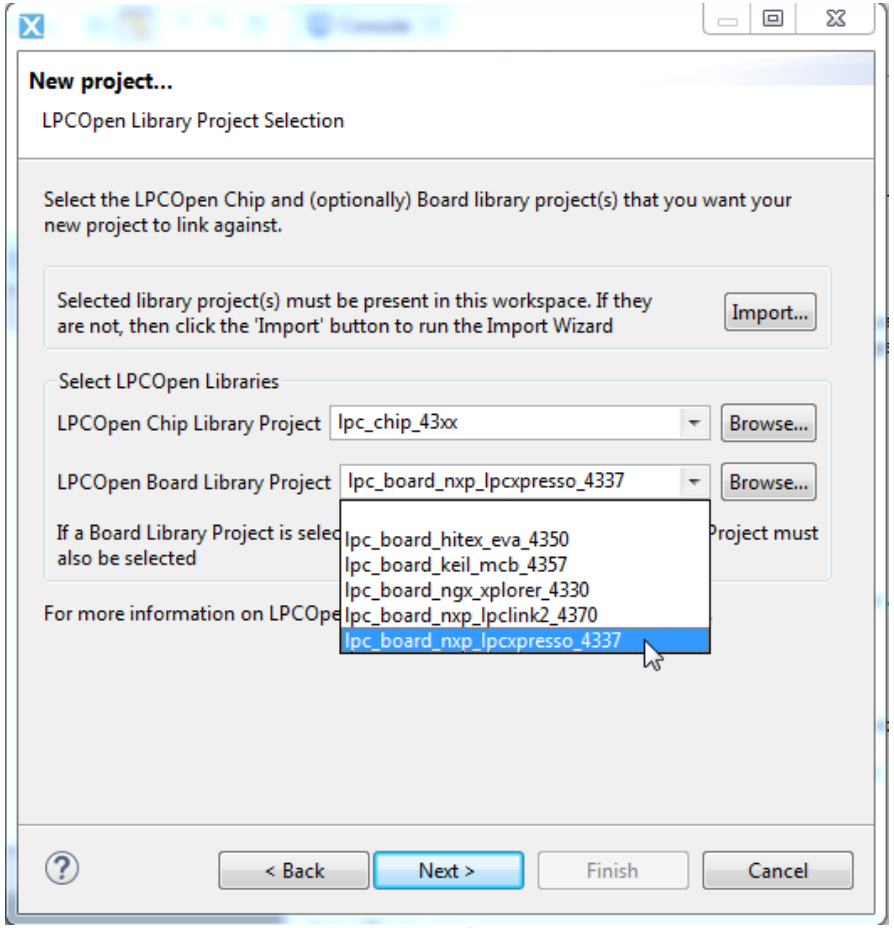
- Quickstart -> New project -> LPC1800/4300 -> LPC43xx Multicore M4



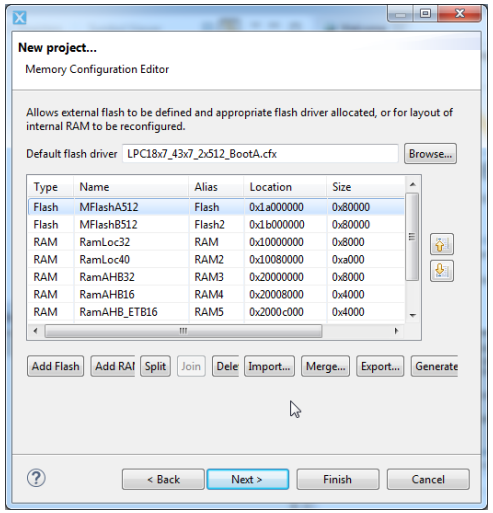



# Select LPCOpen Library Projects

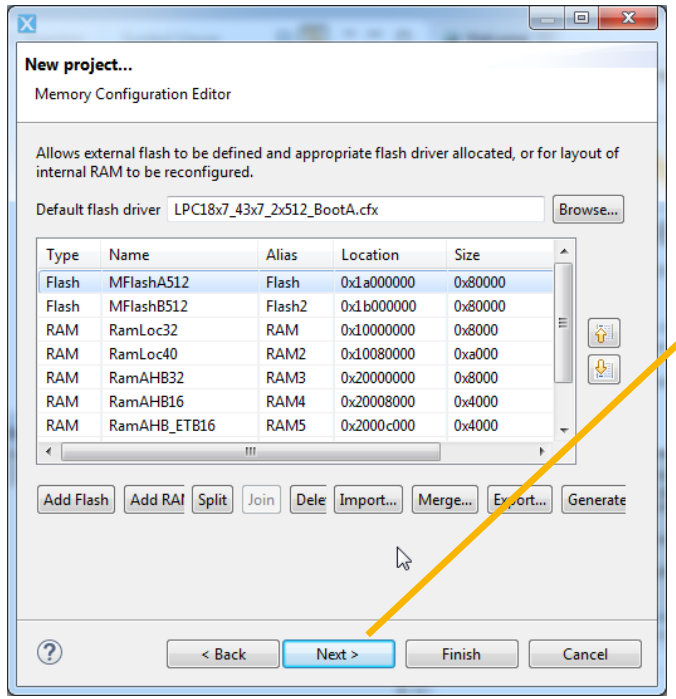



- Select the LPCOpen Chip and Board libraries that will be linked against
  - Chip Library automatically selected from MCU selection
  - Board Library : Needs to be selected based on your target board
    - For CM4 cpu on LPCXpresso4367 board, select `lpc_board_nxp_lpcxpresso_4337`
- Now click through the next few wizard pages until you reach the Memory Configuration Editor…

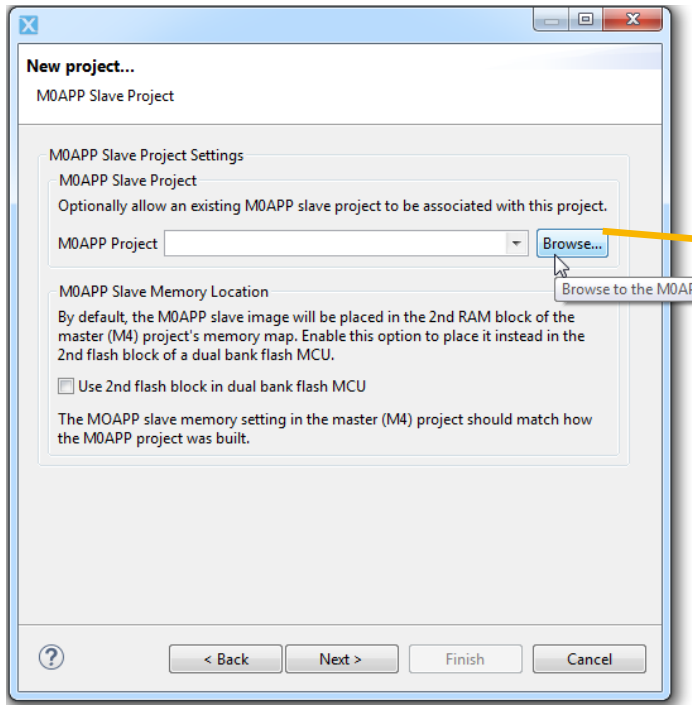


# Memory Layout

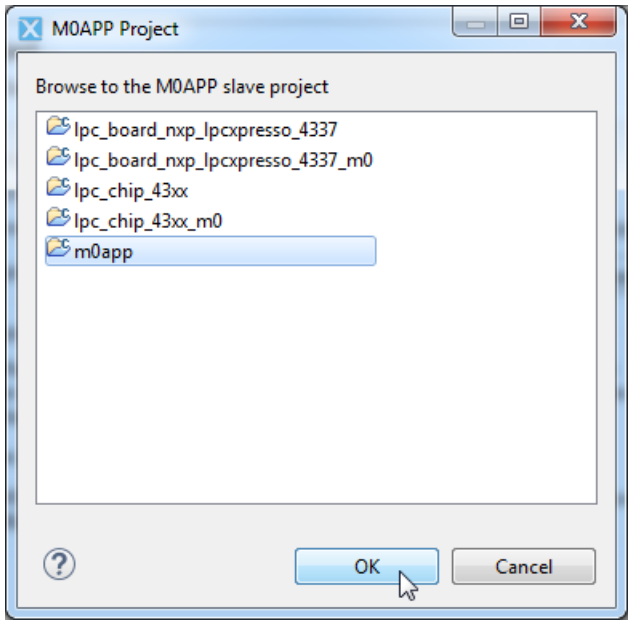
- This time the default memory layout is OK
- But need to configure the CM0 image to be used and where to place it




Just click Next in Memory Configuration Editor

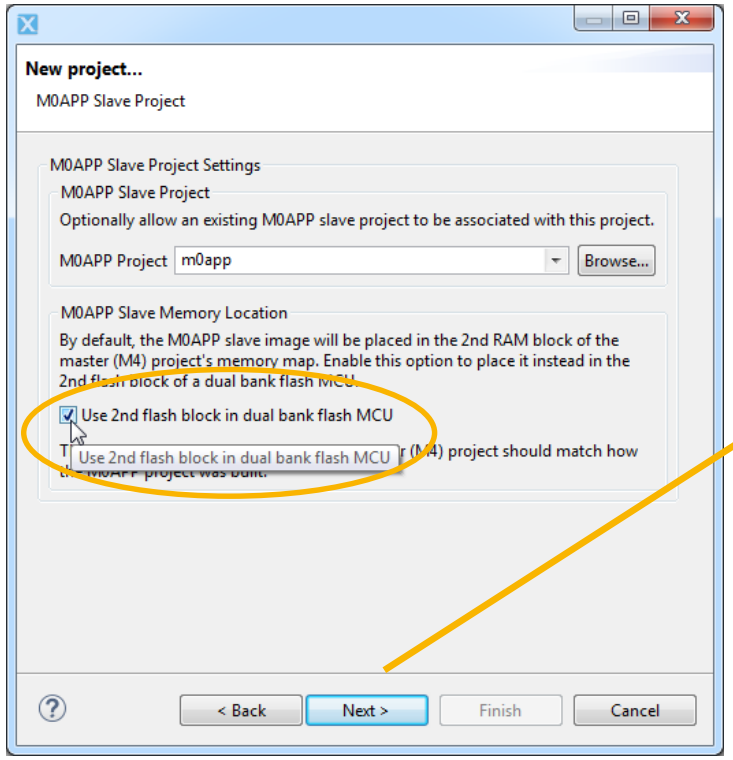



Click on “Browse” to select M0App project from within workspace

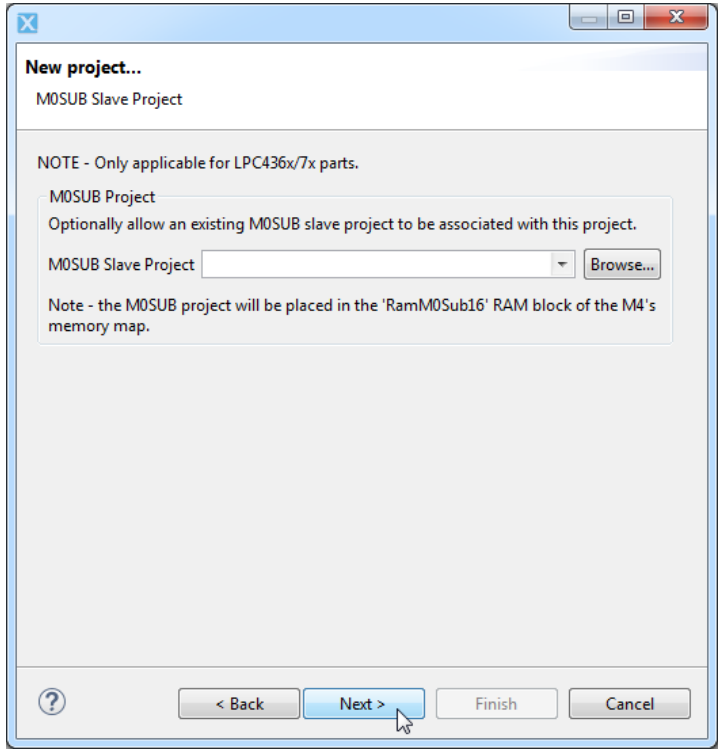



Select the “m0app” project, then click on ***OK***

# Place in Flash




Tick the box to place the M0App project into the 2nd flash bank, then click Next



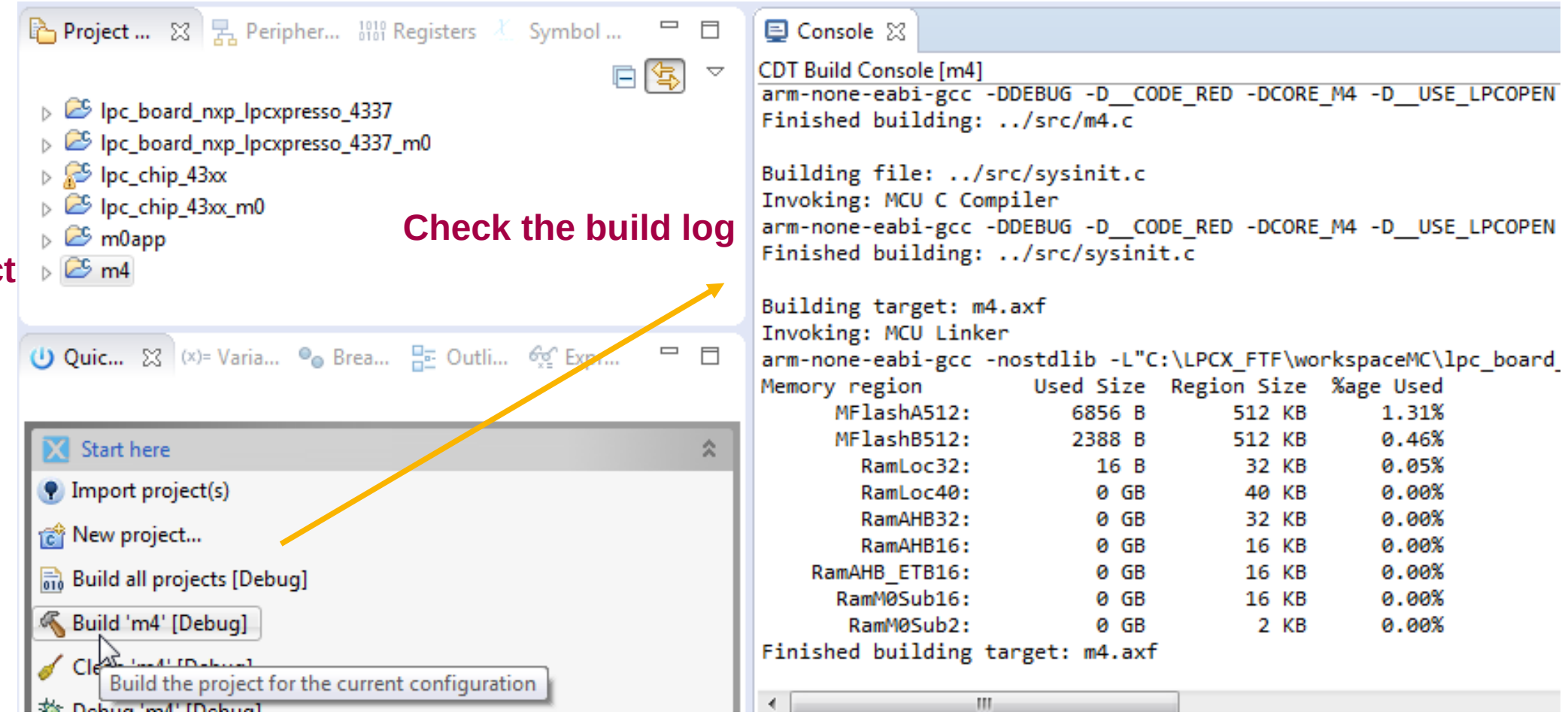

No M0Sub project used in this example, so leave blank and click Next, then click Finish on the next page to create the project

# Check the M4 Master Project Builds

Select Project

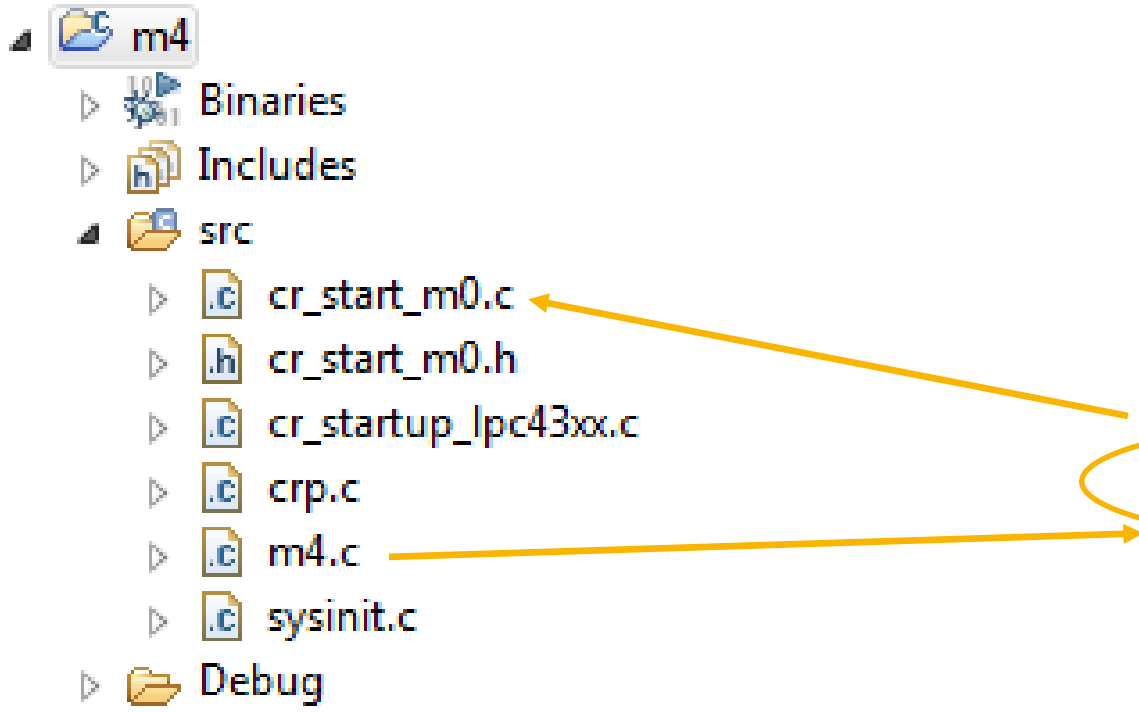
Build it

Check the build log




Note that MFlashB512 contains the M0App image

# CM4 Master Project – Additional Contents




cr_start_m0.c | m4.c

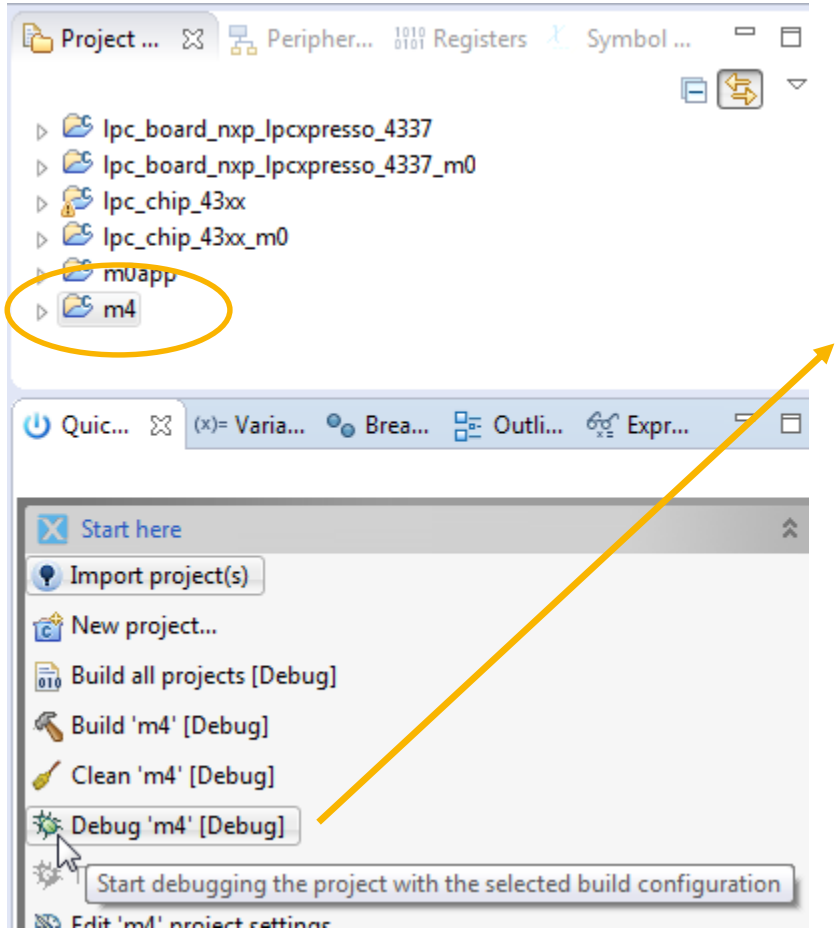
```
29 int main(void) {
30
31 #if defined (__USE_LPCOPEN)
32     // Read clock settings and update SystemCoreClock variable
33     SystemCoreClockUpdate();
34 #if !defined(NO_BOARD_LIB)
35 #if defined (__MULTICORE_MASTER) || defined (__MULTICORE_NONE)
36     // Set up and initialize all required blocks and
37     // functions related to the board hardware
38     Board_Init();
39 #endif
40     // Set the LED to the state of "On"
41     Board_LED_Set(0, true);
42 #endif
43 #endif
44
45     // Start M0APP slave processor
46 #if defined (__MULTICORE_MASTER_SLAVE_M0APP)
47     cr_start_m0(SLAVE_M0APP,&__core_m0app_START__);
48 #endif
49
50     // Start M0SUB slave processor
51 #if defined (__MULTICORE_MASTER_SLAVE_M0SUB)
52     cr_start_m0(SLAVE_M0SUB,&__core_m0sub_START__);
53 #endif
54
55     // TODO: insert code here
56
57     // Force the counter to be placed into memory
58     volatile static int i = 0 ;
59     // Enter an infinite loop, just incrementing a counter
60     while(1) {
```

# Make a Minor Modification to M4 Main()

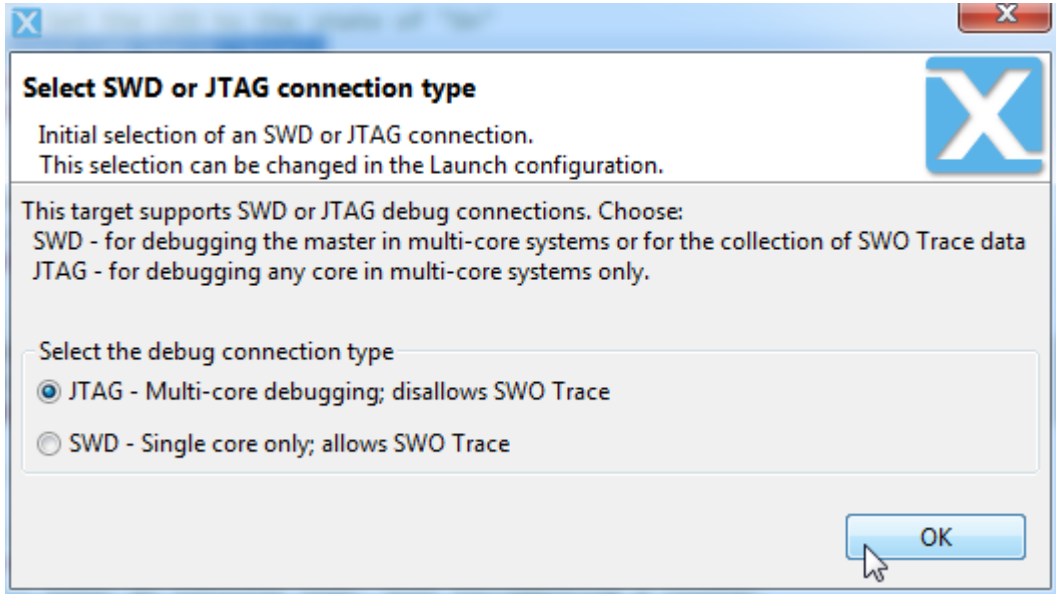
- Change
  - Board_LED_Set(0, true);
- To
  - Board_LED_Set(1, true);

- This means the CM4 and the CM0 applications will turn on different LED colors!

# Starting a Multicore Master Debug Session

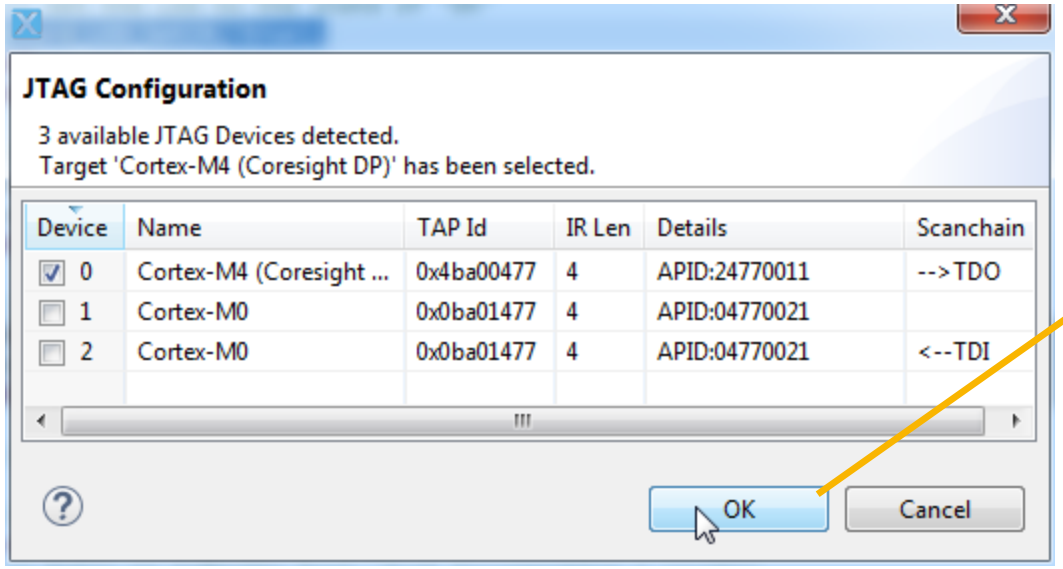


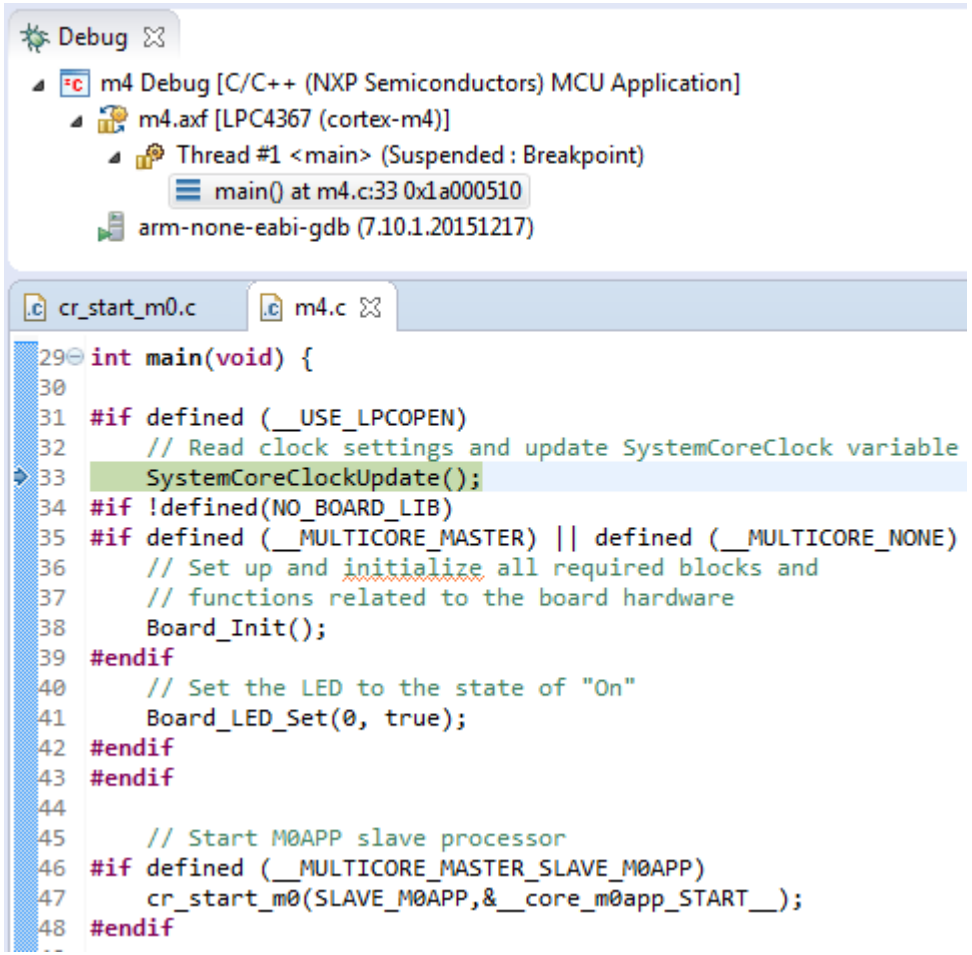

Probe boots
Select LPC-Link2




For LPC43xx multicore debugging, you must select a JTAG debug connection, then click OK

# Making the Multicore Master Debug Connection

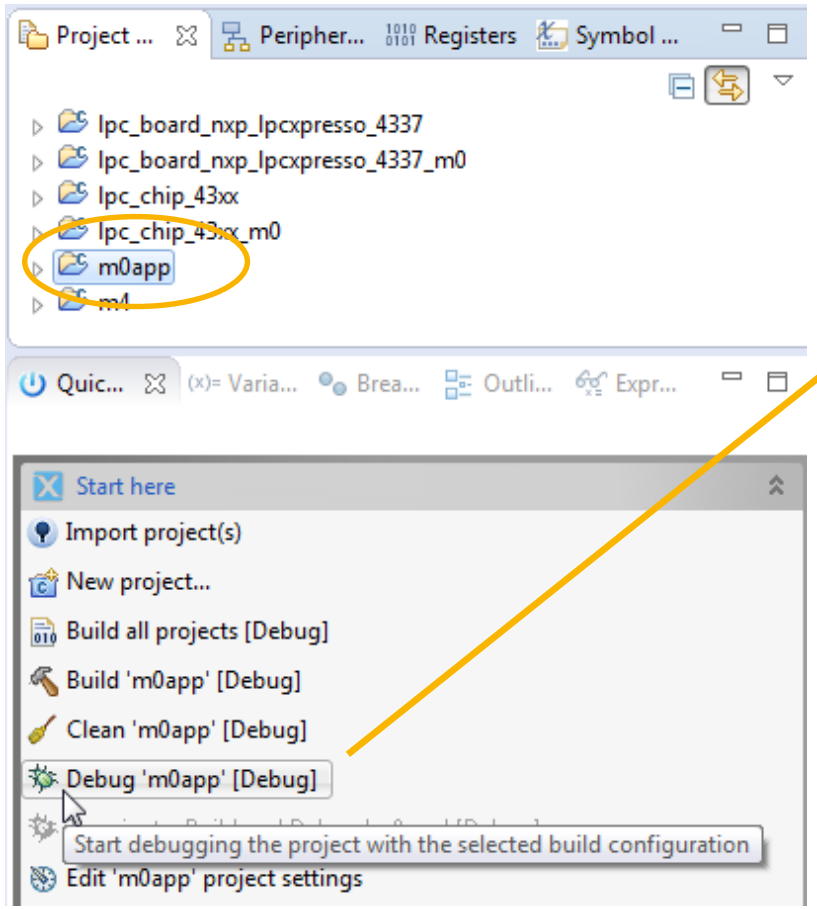


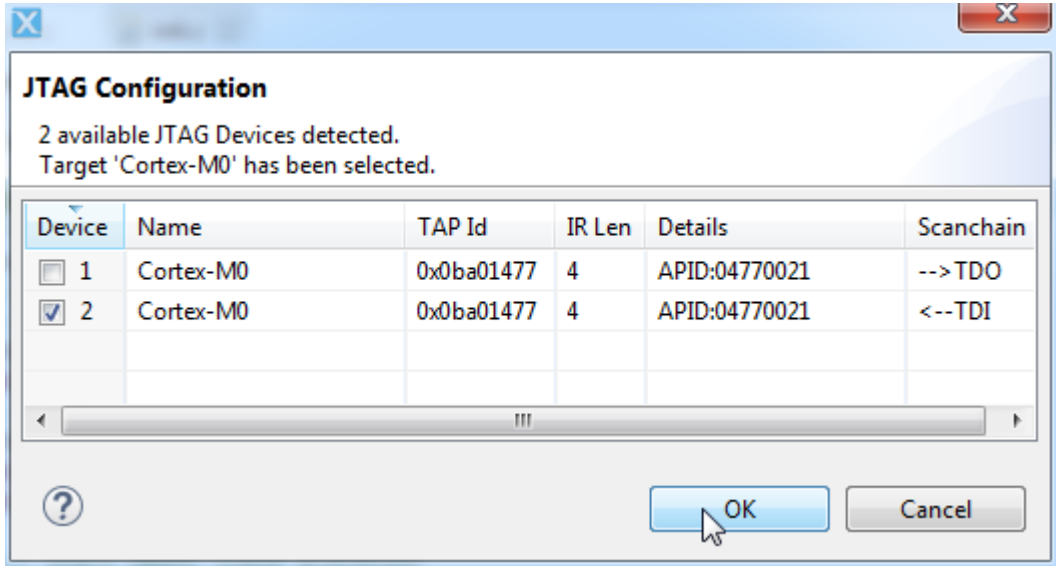

Select the appropriate CPU in the JTAG configuration.

Device closest to TDO is the M4 CPU (device 0 here). Make sure you select the right one!

# Starting a Multicore Slave Debug Session




Select LPC-Link2

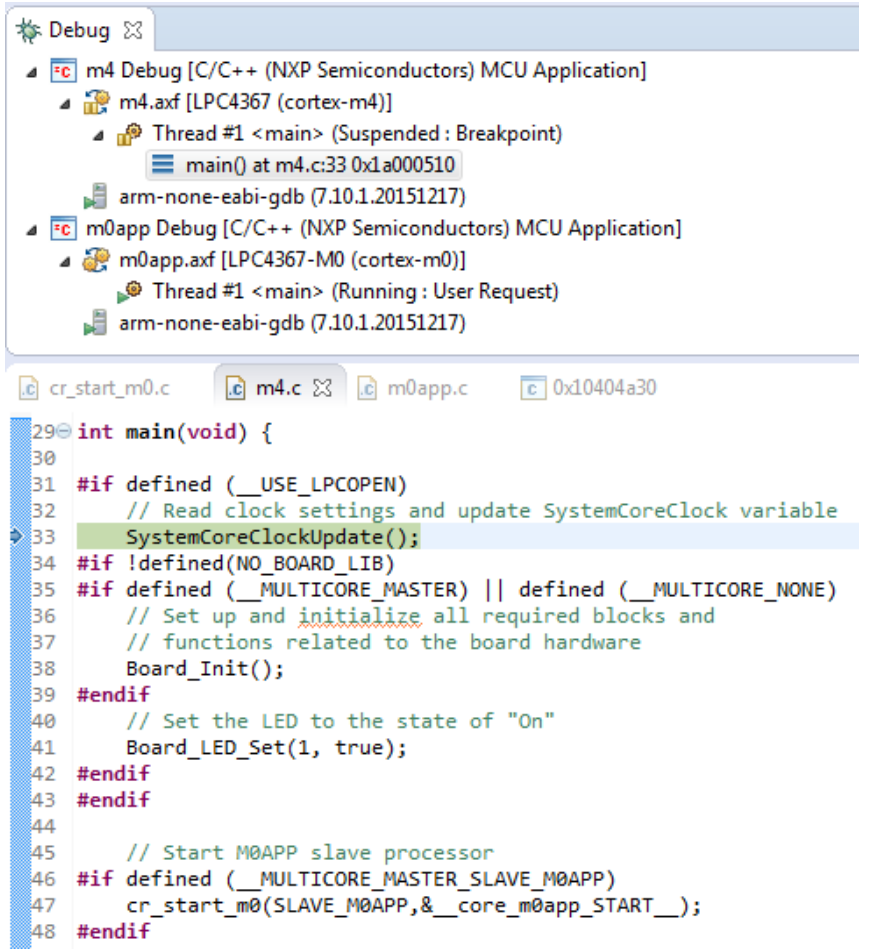



Select the appropriate CPU in the JTAG configuration. Device closest to TDI is the M0APP CPU (device 2 here).
Make sure you select the right one!

The M0APP debug session is an “attach only” connection – no code is downloaded

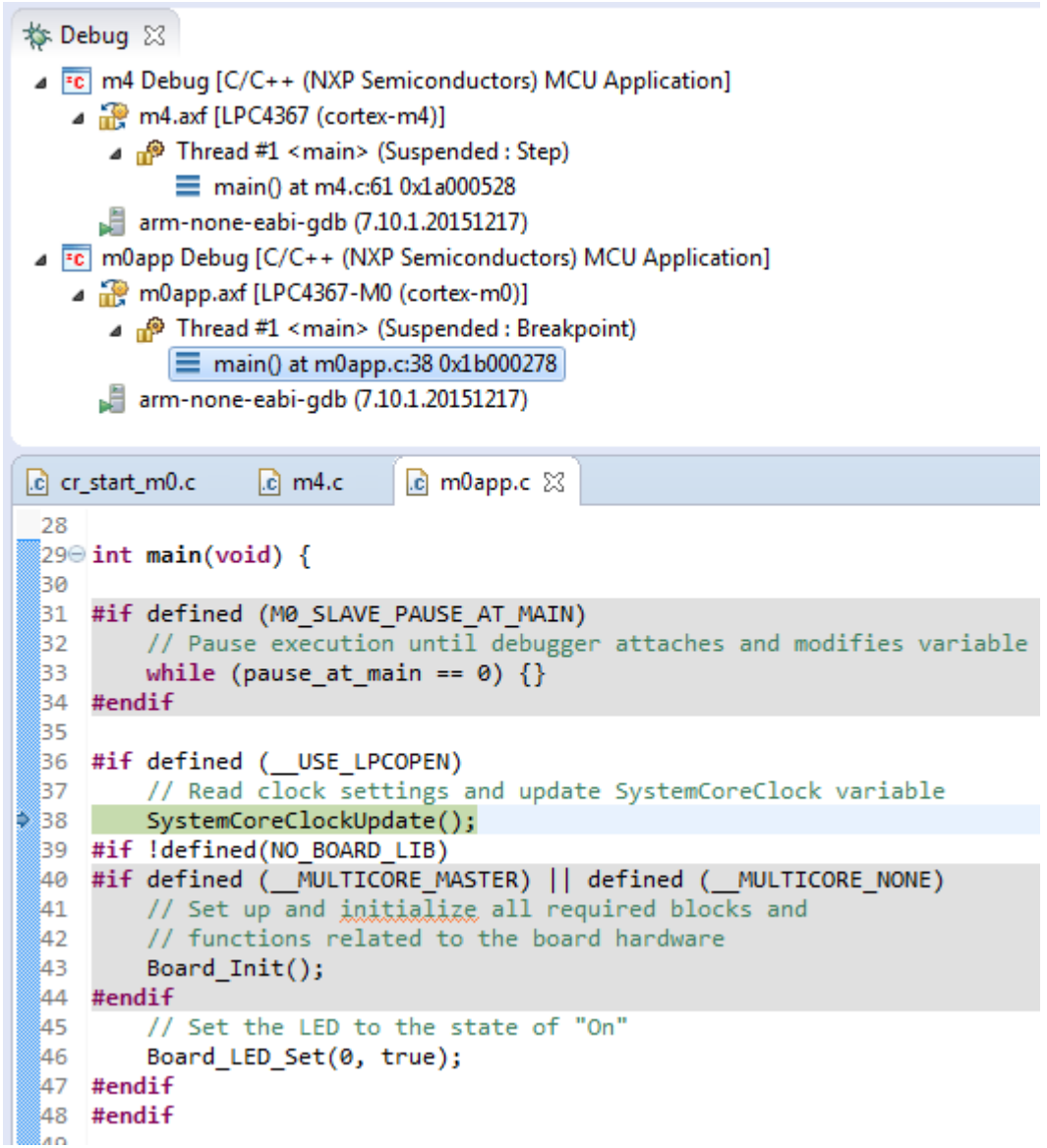
# Master and Slave Debug Sessions Started

- The IDE is now debugging both cpus
- But the M0APP CPU is still held in reset
- Ensure the m4 is highlighted in the Debug View, then use “Step over” to step through the M4 code until you reach
  - `Board_Led_Set (1, true);`
- Step this statement and the Green LED should come on
- Then “Step over” the next statement
  - `cr_start_m0(SLAVE_M0APP, &__core_m0app_START__);`

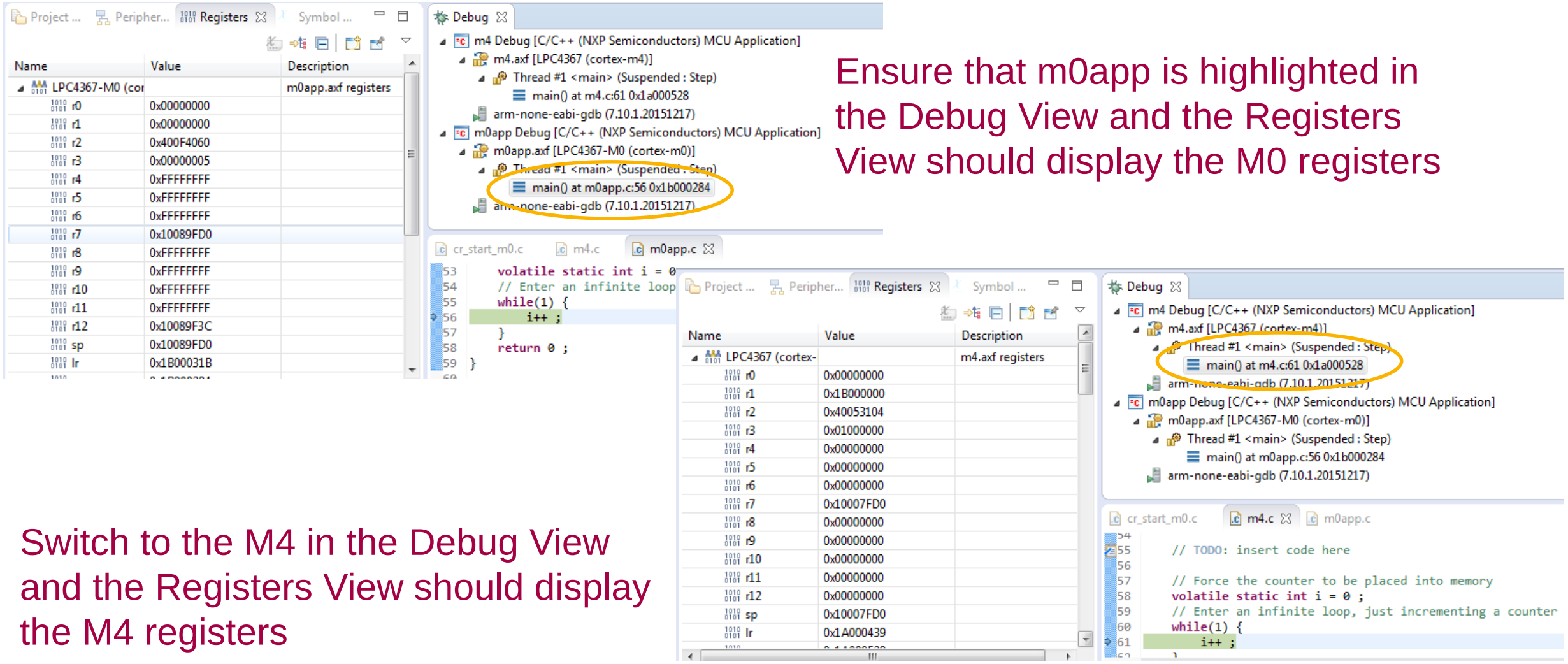


# M0APP Is Now Running

- The M4 has now released the M0APP from reset and it has now hit its default breakpoint on main().
- Ensure that m0app is highlighted in the Debug View, then use “Step over” to step through the M0 code until you reach
  - `Board_Led_Set (0, true);`
- Step this statement and the Red LED should come on (combining with the Green LED turned on by the M4 to give a Blue color).



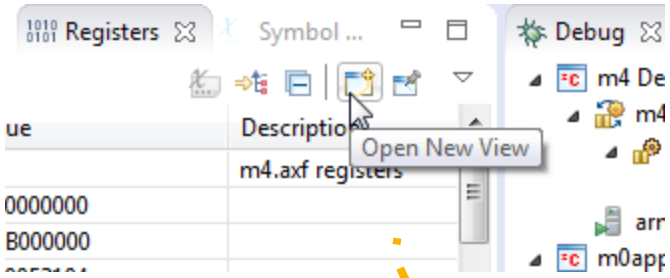
# Switch to the Registers View

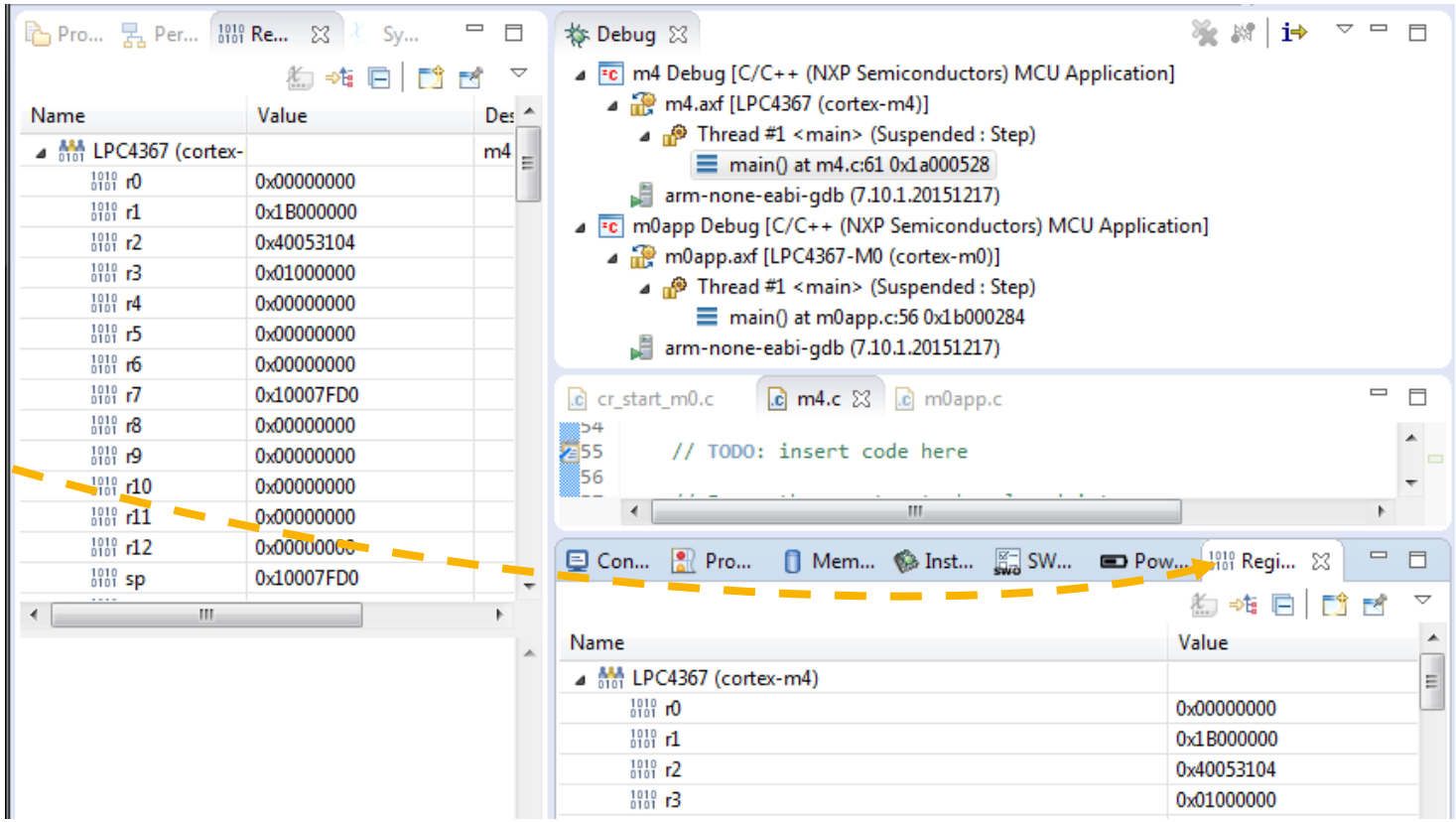



Ensure that m0app is highlighted in the Debug View and the Registers View should display the M0 registers

Switch to the M4 in the Debug View and the Registers View should display the M4 registers

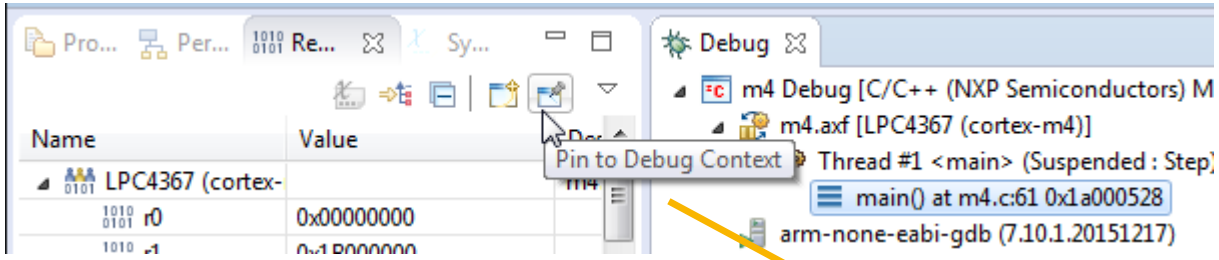
# Display Both CPUs' Registers – Create 2nd Register View



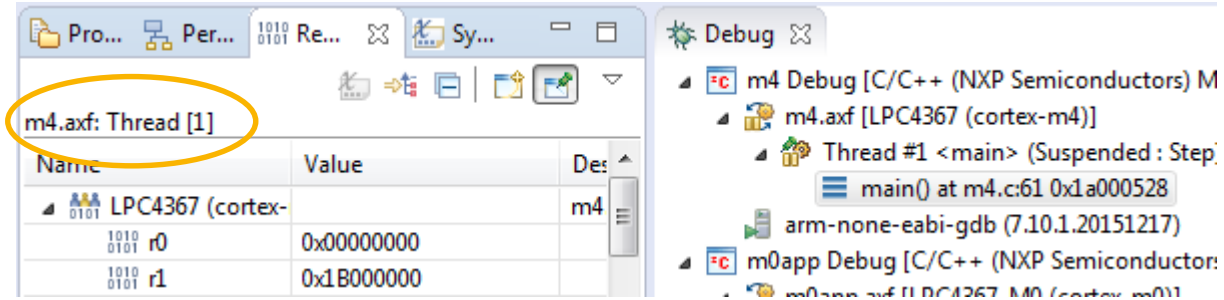




Click on "Open New View" in the original Registers View, and a 2nd view will open. By default this will be in the bottom right of the IDE, but you can move it to where you want it afterwards

# Display Both CPUs' Registers – Pin to Debug Context – 1st View

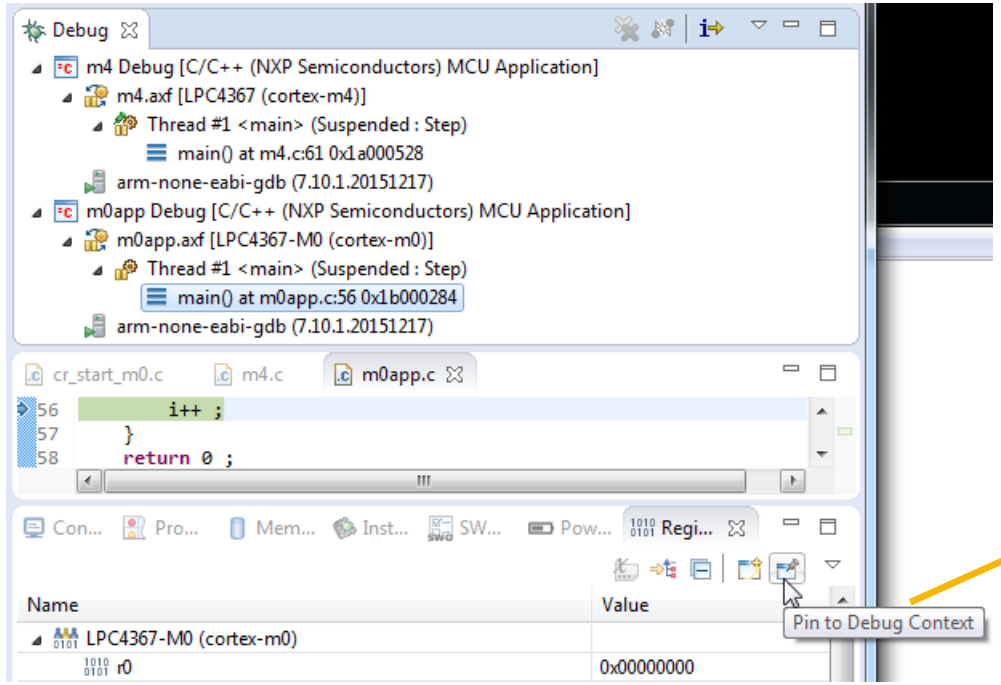


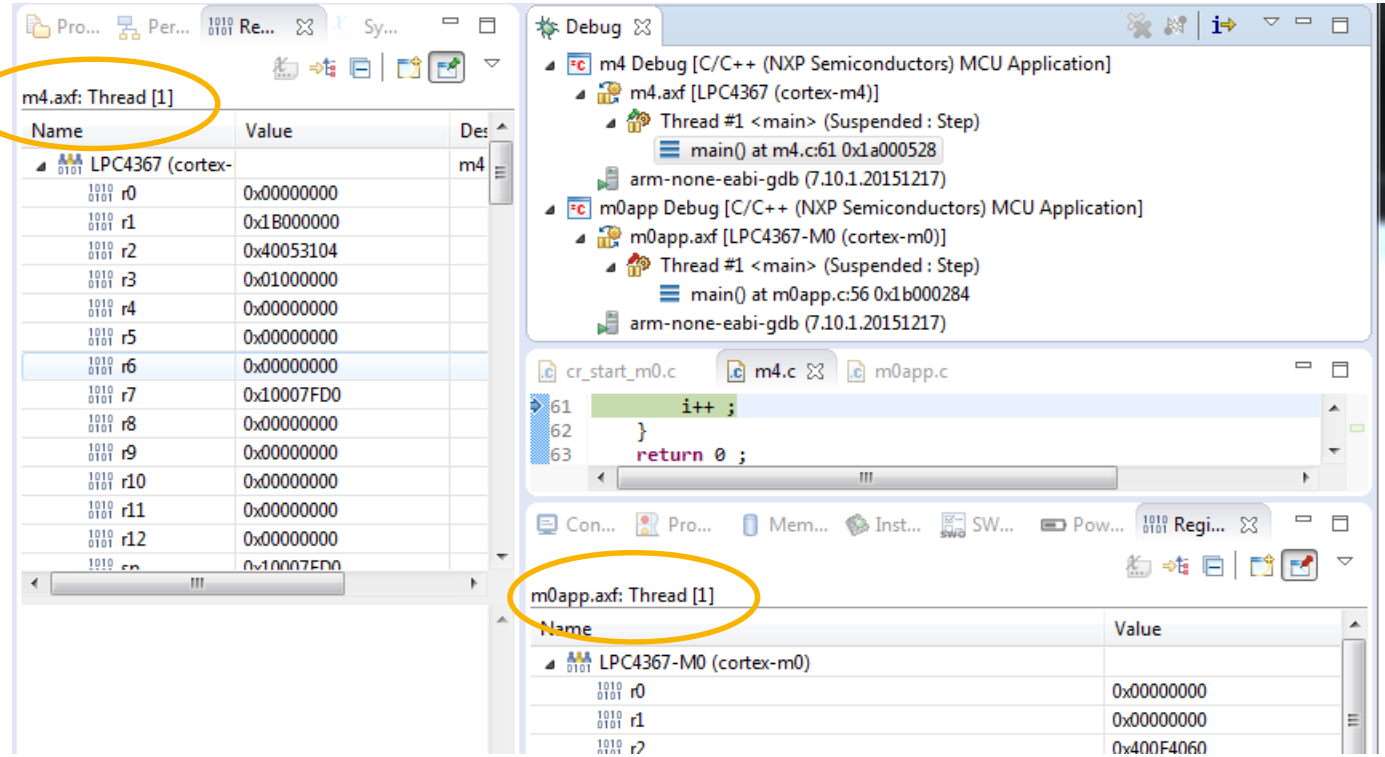

Switch to the M4 in the Debug View, then use "Pin to Debug Context" in the original Register View




The original Register View is now locked to the M4 CPU

# Display Both CPUs' Registers – Pin to Debug Context – Both Views



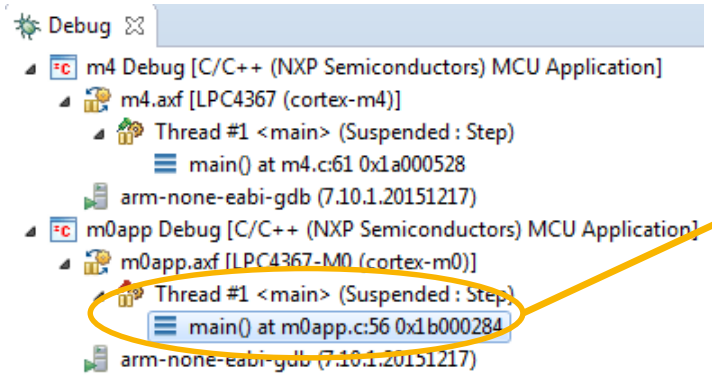

Switch to the M0APP in the Debug View, then use "Pin to Debug Context" in the second Register View

Each Register View is now locked to a specific CPU

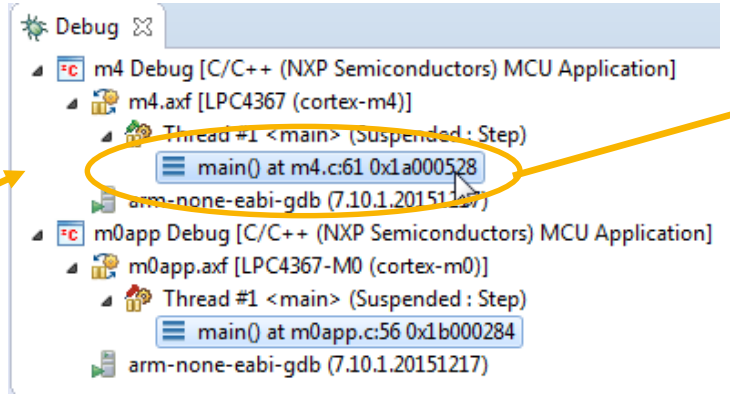
The same can also be done with the Variables and Expressions Views

# Run Control

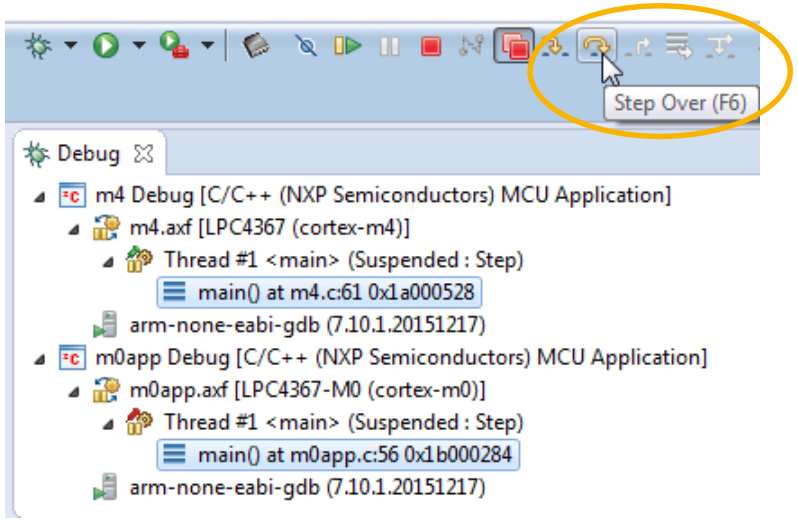
- By default, the Run Control options work on the currently selected CPU.
- But you can select multiple CPUs in the Debug View using CTRL-Click, and then use Resume/Step/Pause on all CPUs at the same time
  - This will be made simpler in the next LPCXpresso IDE release




Select 1st CPU




Ctrl-Click to select 2nd CPU

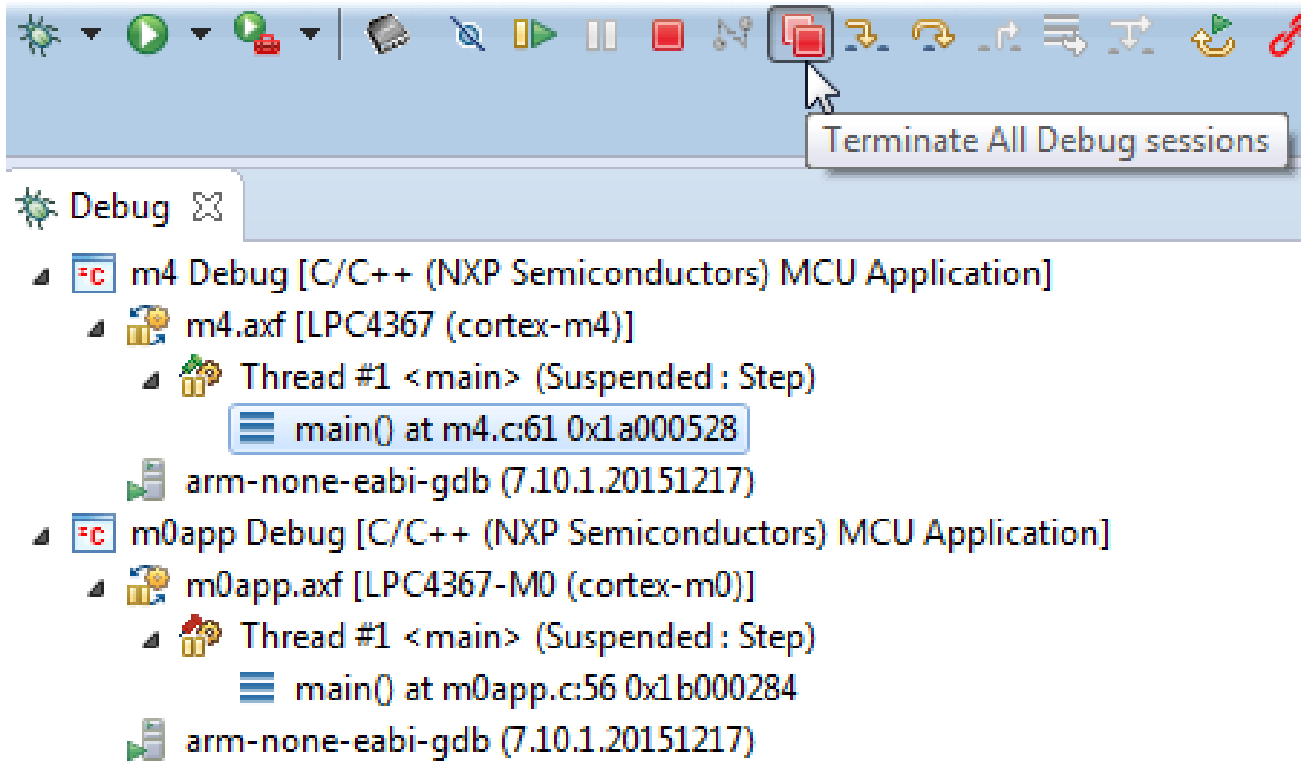



Can now step both at the same time

Note : Synchronization here is done by the debug tools, not in hardware

# Terminating a Multicore Connection

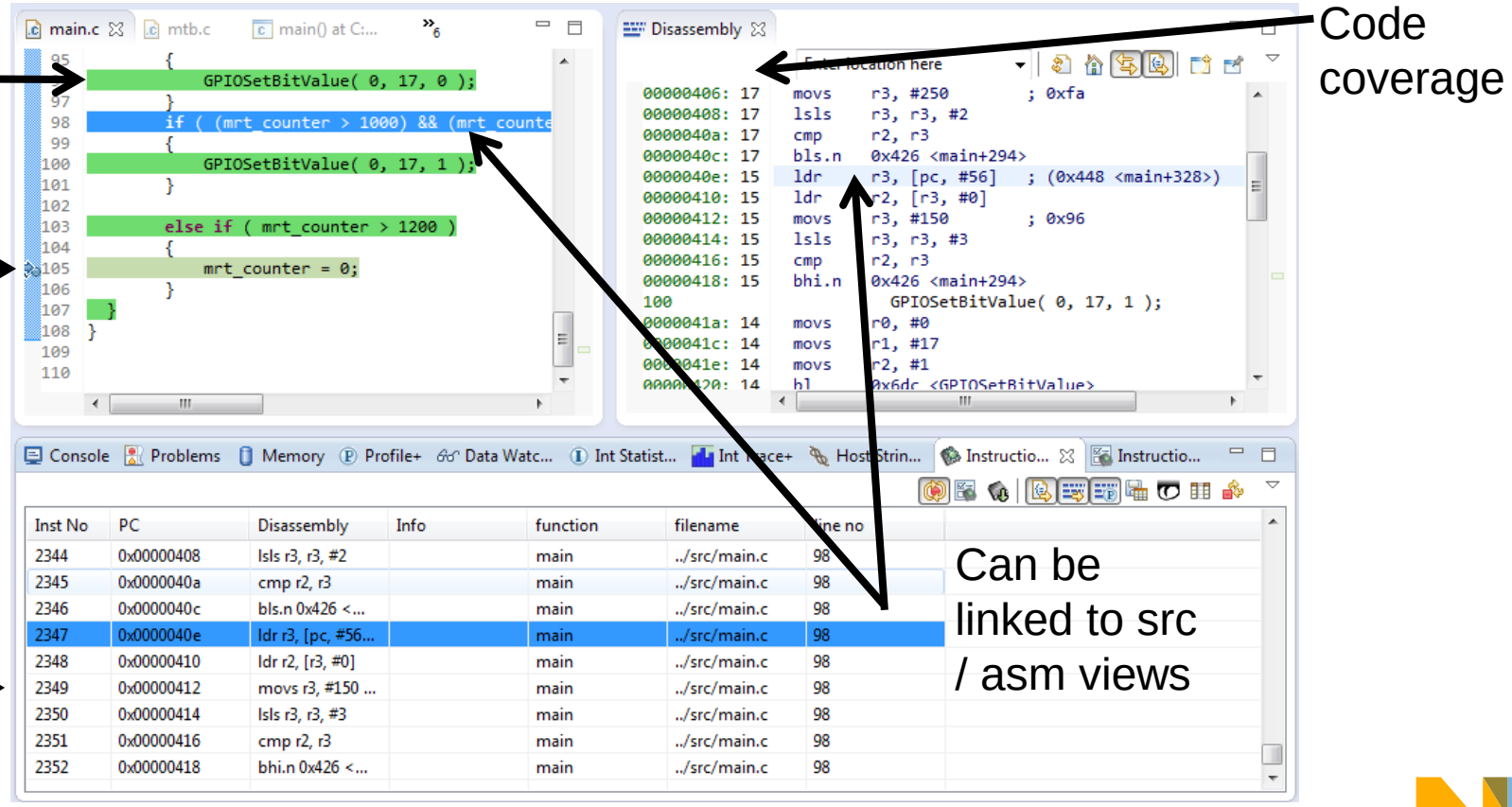
- When you have finished debugging, you can use “Terminate/Disconnect All” button on the icon bar to disconnect the debug connections to both the Master and Slave CPUs at the same time



# INSTRUCTION TRACE

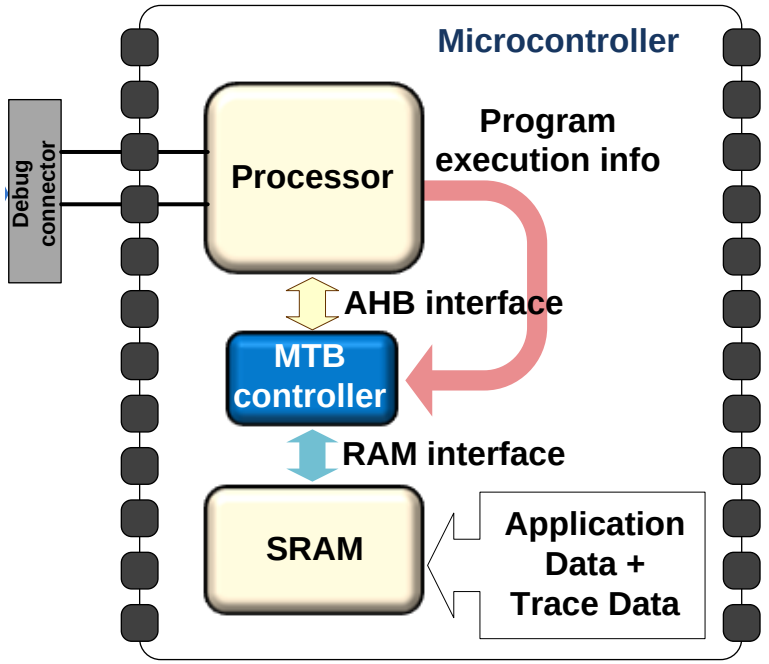
# Instruction Trace

- Collects details of instructions being executed
- Allows complex program flow problems to be examined
  - Gives insight into what happening in system before a fault was encountered



# Instruction Trace – Supported Targets

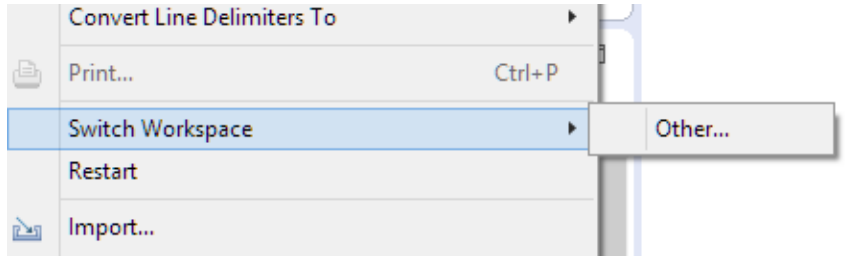
- Cortex M3/M4 LPC MCUs
  - Target MCU must implement both an Embedded Trace Macrocell (ETM) AND an Embedded Trace Buffer (ETB).
    - ✓ LPC18xx and LPC43xx parts
    - ✗ LPC17xx parts
      - ✗ Do not implement an ETB
      - ⌘ Third party tools allow trace using buffer implemented via debug probe
    - ✗ LPC13xx parts
      - ✗ Do not implement ETM or ETB
- Cortex M0+ LPC MCUs
  - Target MCU must implement a Micro Trace Buffer (MTB)
    - ✓ LPC8xx parts
    - ✓ LPC11U6x/11E6x parts



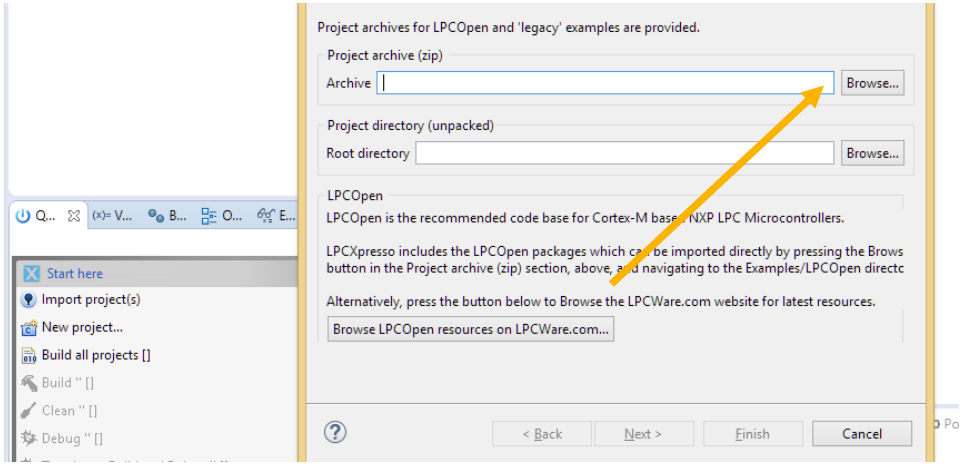

Instruction Trace in LPCXpresso can be carried out via any supported debug probe

# Instruction Trace Exploration #1

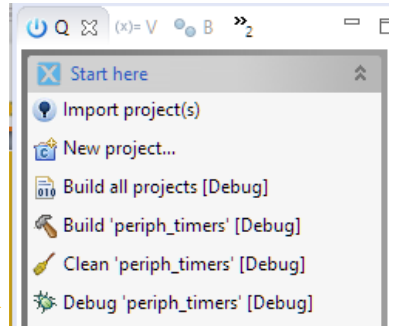
- To begin: Create a new workspace
  - From within the IDE select File -> Switch Workspace
    - enter e.g. C:\LPCX_FTF\InstructionTrace




- Import from the LPCXpresso4337 LPCOpen examples:
  - The Chip Library - lpc_chip_43xx
    - contains library support code for the features of the MCU
  - The Board Library - lpc_board_nxp_lpcxpresso_4337
    - contains library support code for features of the board
  - The Project – periph_timers
    - contains our example code




- Build and Debug the Application:
  - Click on the imported project in project explorer view to select it
  - Click ‘Debug’ in Quickstart panel to both build and launch a debug session
    - The example will now run to an automatic Breakpoint on Main

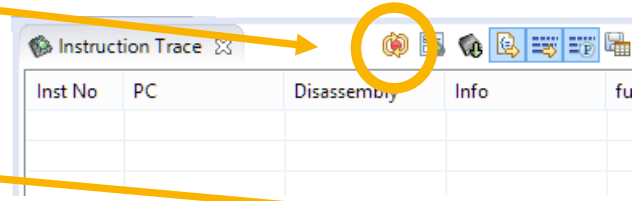


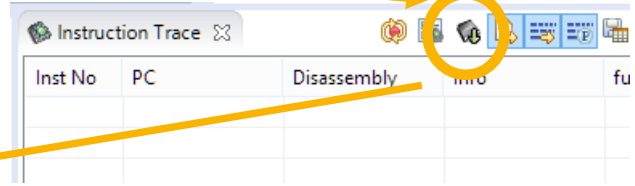
# Instruction Trace Exploration #2

- In this example we will trace through the initialization code from the beginning of main()
  - To limit the amount of code we collect, we can trace from the start of main to a break point
- In the editor view, locate the source file timers.c
  - Click on line 98 to place a Breakpoint i.e. before the while(1) loop is entered

```
92    Chip_TIMER_Enable(LPC_TIMER1);
93
94    /* Enable timer interrupt */
95    NVIC_EnableIRQ(TIMER1_IRQn);
96    NVIC_ClearPendingIRQ(TIMER1_IRQn);
97
98    DEBUGSTR("Blinky example using timer 1!\r
99    DEBUGOUT("Timer 1 clock     = %d Hz\r\n",
100   DEBUGOUT("Timer 1 tick rate = %d Hz\r\n",
101
102   while (1) {
```

- In the Console view, locate and select the Instruction trace view:
  - Click the ‘record continuously’ button to enable simple trace capture
  - Click ‘Resume’ to run from main() to our breakpoint
    - Instruction trace data will have been captured to on chip ETB buffer
  - Click ‘Download Trace buffer’ to download and decode the compressed data



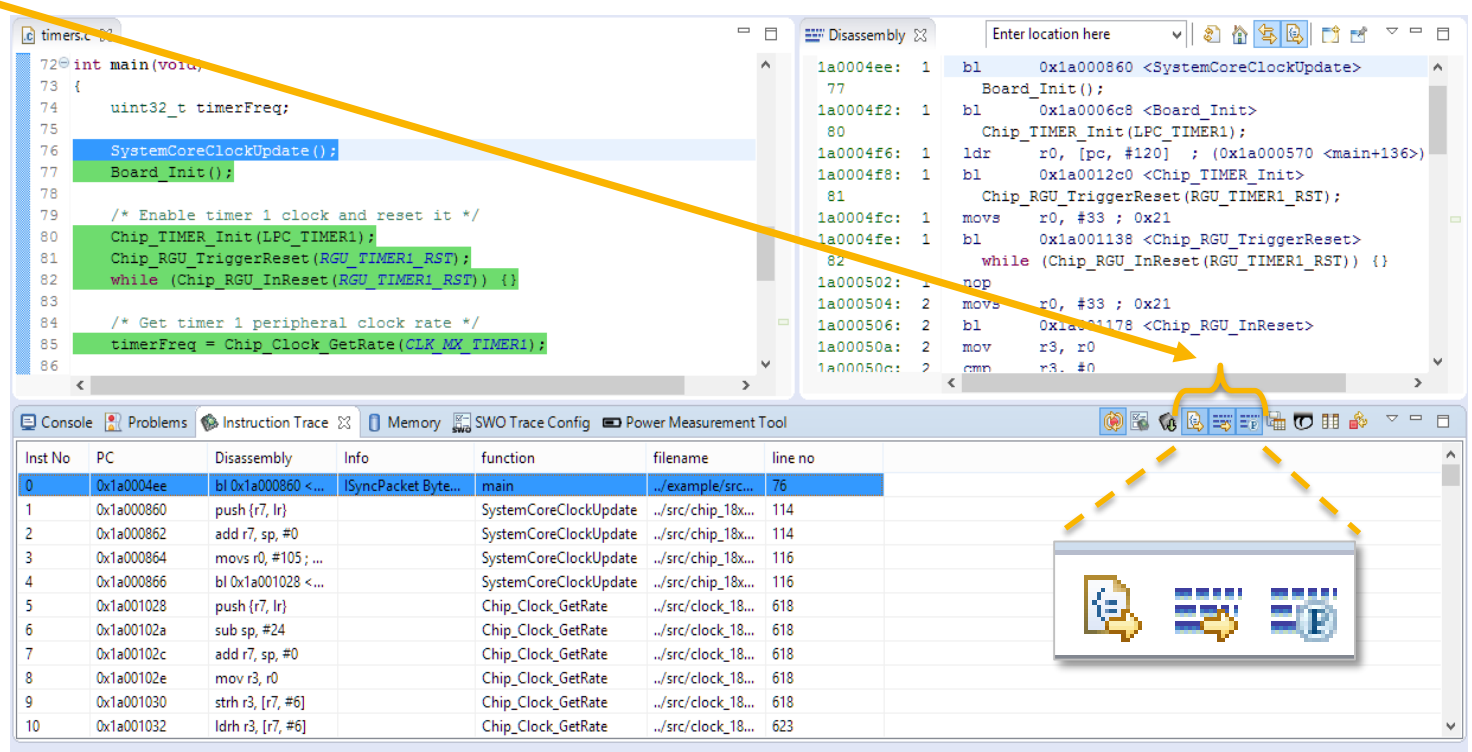




Instruction Trace

| Inst No | PC | Disassembly | Info | function | filename | line no |
|---|---|---|---|---|---|---|
| 0 | 0x1a0004ee | bl 0x1a000860 <... | ISyncPacket Byte... | main | ../example/src... | 76 |
| 1 | 0x1a000860 | push {r7, lr} | | SystemCoreClockUpdate | ../src/chip_18x... | 114 |
| 2 | 0x1a000862 | add r7, sp, #0 | | SystemCoreClockUpdate | ../src/chip_18x... | 114 |
| 3 | 0x1a000864 | movs r0, #105 ; ... | | SystemCoreClockUpdate | ../src/chip_18x... | 116 |
| 4 | 0x1a000866 | bl 0x1a001028 <... | | SystemCoreClockUpdate | ../src/chip_18x... | 116 |
| 5 | 0x1a001028 | push {r7, lr} | | Chip_Clock_GetRate | ../src/clock_18... | 618 |
| 6 | 0x1a00102a | sub sp, #24 | | Chip_Clock_GetRate | ../src/clock_18... | 618 |
| 7 | 0x1a00102c | add r7, sp, #0 | | Chip_Clock_GetRate | ../src/clock_18... | 618 |
| 8 | 0x1a00102e | mov r3, r0 | | Chip_Clock_GetRate | ../src/clock_18... | 618 |
| 9 | 0x1a001030 | strh r3, [r7, #6] | | Chip_Clock_GetRate | ../src/clock_18... | 618 |
| 10 | 0x1a001032 | ldrh r3, [r7, #6] | | Chip_Clock_GetRate | ../src/clock_18... | 623 |

# Instruction Trace Exploration #3

- Code in the Instruction Trace view can be linked back to its Source and Disassembly
  - Click the three buttons shown to:
    - Link to (and open) source file(s)
      - Source will be shown with a Blue bar
      - File are also located in the Project Explorer view
    - Link to and spawn the disassembly view
      - Instructions will be shown with a Light blue bar
    - Show profile (code contained in capture)
      - Green bars
  - Click in the Instruction Trace view
    - Use cursor keys to scroll through the Trace
      - Note the code called to perform the initializations



# SWO TRACE

# SWO Trace

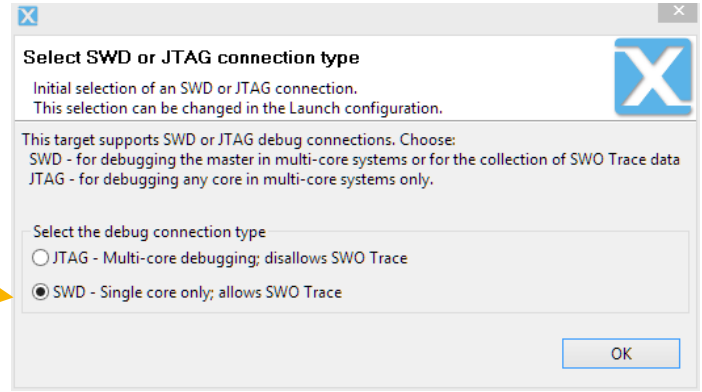
- Serial Wire Output (SWO) Trace
  - Hardware blocks within the Cortex-M CPU that can output CPU activity information via a minimal 1 pin serial connection
  - Also referred to as Serial Wire Viewer (SWV)

- Supported on any Cortex M3 / M4 LPC MCU
  - via LPC-Link2 with NXP CMSIS-DAP firmware

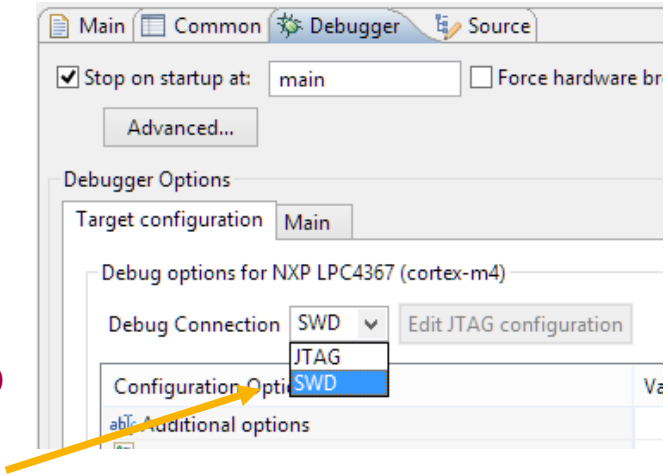
- Requires additional SWO pin from MCU to debug connector
  - Some parts also require pinmux configuration and/or trace clock enabling code
    - Typically provided in pinmux setup of LPCOpen board library / LPCXpresso startup code

# SWO Trace Debug Connection

- SWO trace features require an SWO Debug connection
  - Be sure to select SWD when making your debug connection




- Debug connection protocols are stored within a project launch configuration
  - These are automatically created when a project is first debugged
    - And will be re-used for subsequent Debug operations
  - To edit/change an existing connection protocol
    - Right click on a project and select the Launch Configuration:
      - Edit current -> Debugger ->Target configuration -> Debug Connection -> SWO



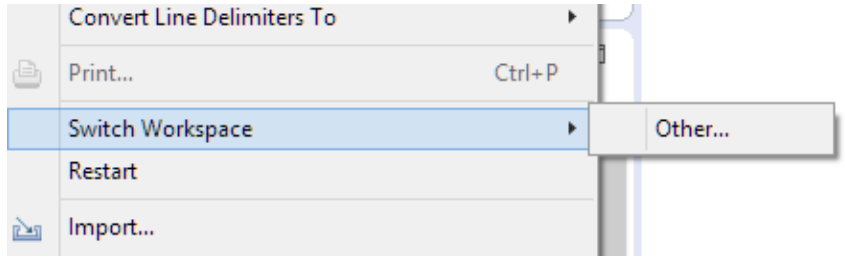
# SWO Trace Exploration #1

- To demonstrate the features of SWO Trace within LPCXpresso IDE, we need some code that incorporates :
  - some Functions (Profiling)
  - some Exceptions (Interrupts)
  - some Global variables (Data Watch)
  - and a Printf statement (ITM)

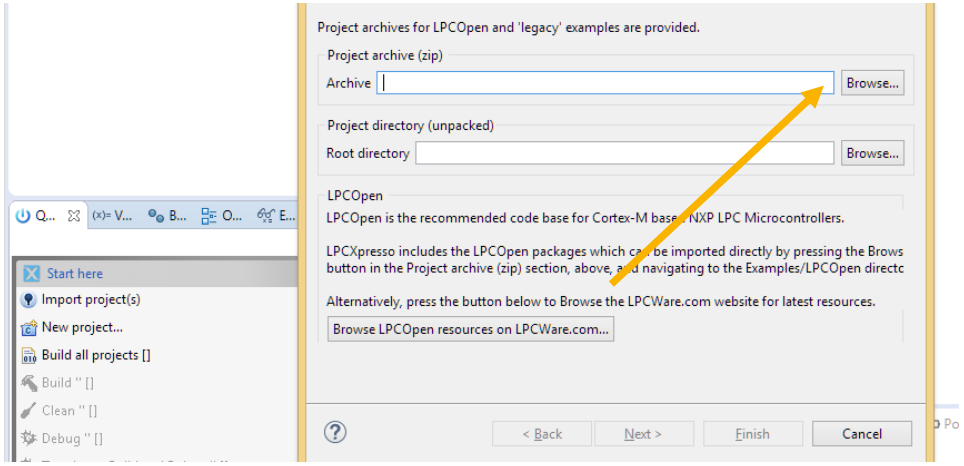
- The LPC4367_SWO_Example project incorporates these features :
  - Configuration of 5 exceptions (4 Timers and 1 SysTick)
  - Handler functions for each exception that updates some global variable(s)
  - Flashing LED in Main timed from SysTick
  - Print of the current LED colour (disabled in the initial build)

# SWO Trace Exploration #2

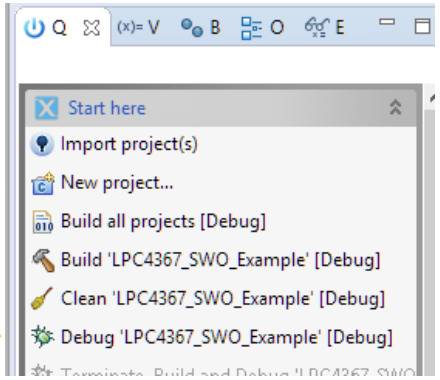
- To begin: Create a new workspace
  - From within the IDE select File -> Switch Workspace
    - enter e.g. C:\LPCX_FTF\swo




- Import the supplied example: LPC4367_SWO_Example
  - Found in C:\LPCX_FTF
  - This import consists of 3 parts:
    - The Chip Library - lpc_chip_43xx
      - contains library support code for the features of the MCU
    - The Board Library - lpc_board_nxp_lpcxpresso_4337
      - contains library support code for features of the board
    - The Project - LPC4367_SWO_Example
      - contains our example code

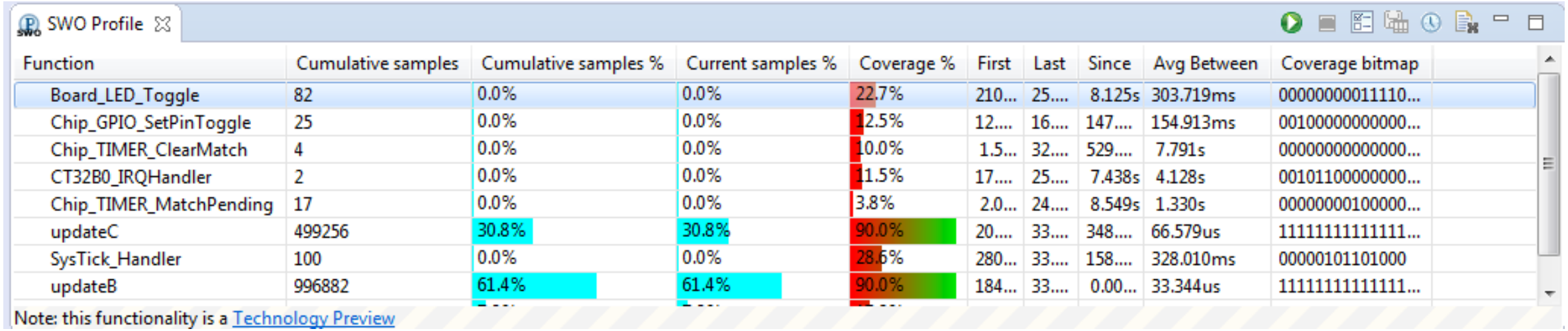



- Build and Debug the Application:
  - Click on the imported project to select it
  - Click ‘Debug’ to both build and launch a debug session
    - The example will now run to an automatic Breakpoint on Main



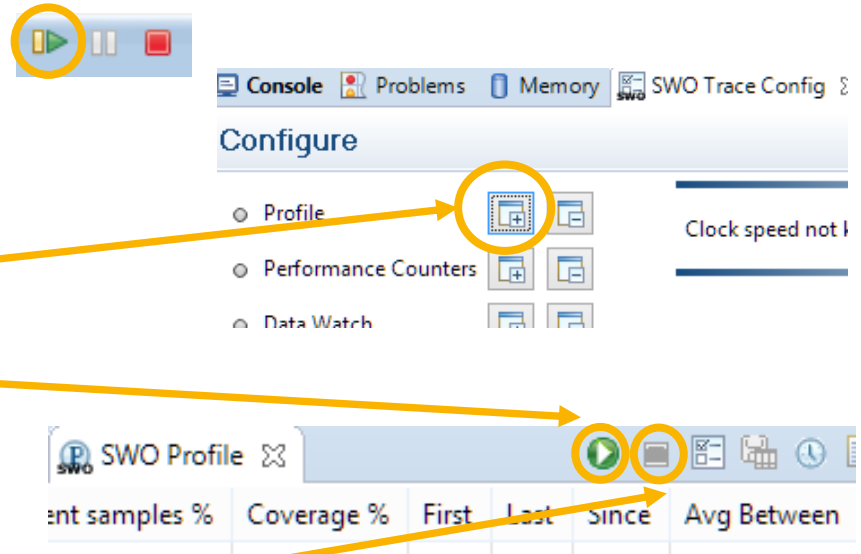
# SWO Trace – Profile View

- Displays statistical profile of application activity
  - Based on PC sampling, typically at ~50kHz
  - Non-intrusive – does not affect application
- Benefits – Identify hotspots



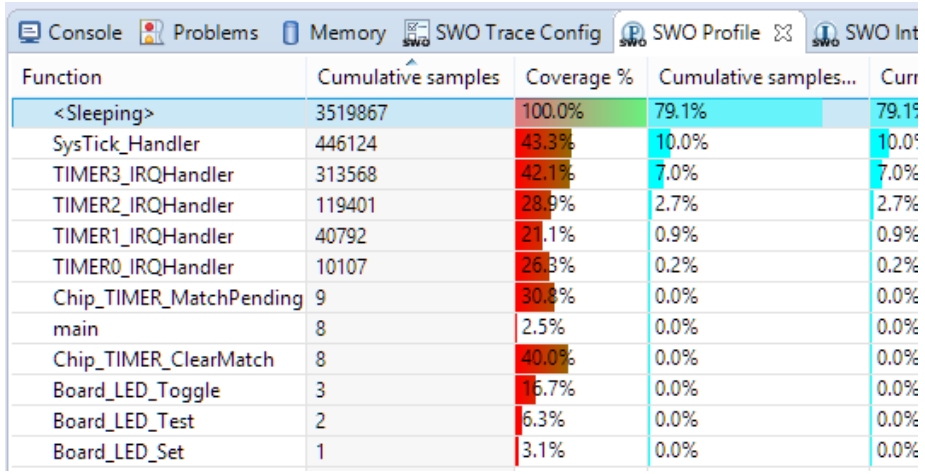
# SWO Profiling Example #1

- To start profiling the example application:
  - First click the application Resume button to start execution from Main
  - From the Console view, select the SWO Trace Config
  - Click the Profile+ button to create and go to the SWO Profile view
  - Start Profiling by clicking the Profile Run button
    - The first time a project is started, the MCU clock speed is required
      - If available, this will be automatically read from the symbol:
        - 'SystemCoreClock' and stored in the project .settings folder
  - After a few seconds click the Profile Stop button



# SWO Profiling Example #2

- In this example, interrupt handlers dominate the output
  - Some view features:
    - to look at the code corresponding to a function, just click the function name
    - to sort the columns, just click the column headings
    - to auto select the width, just click between the column headings




| Function | Cumulative samples | Coverage % | Cumulative samples... | Curr |
|---|---|---|---|---|
| <Sleeping> | 3519867 | 100.0% | 79.1% | 79.1 |
| SysTick_Handler | 446124 | 43.3% | 10.0% | 10.0 |
| TIMER3_IRQHandler | 313568 | 42.1% | 7.0% | 7.0% |
| TIMER2_IRQHandler | 119401 | 28.9% | 2.7% | 2.7% |
| TIMER1_IRQHandler | 40792 | 21.1% | 0.9% | 0.9% |
| TIMER0_IRQHandler | 10107 | 26.3% | 0.2% | 0.2% |
| Chip_TIMER_MatchPending | 9 | 30.8% | 0.0% | 0.0% |
| main | 8 | 2.5% | 0.0% | 0.0% |
| Chip_TIMER_ClearMatch | 8 | 40.0% | 0.0% | 0.0% |
| Board_LED_Toggle | 3 | 16.7% | 0.0% | 0.0% |
| Board_LED_Test | 2 | 6.3% | 0.0% | 0.0% |
| Board_LED_Set | 1 | 3.1% | 0.0% | 0.0% |

- SWO Profiling information is generated by capturing the PC value at a timed interval and identifying the function running from this particular code address
  - The greater proportion of time a particular function executes, the more PC samples will be captured from the function
    - Profiling provides visibility of code hotspots and coverage and it increases in accuracy the longer the duration of the run

- Looking at the Cumulative samples for the Timer Handlers we can see that:
  - Handling Timer3's exceptions consumed approximately 30x the runtime of Timer0's handler
    - However from this view we cannot see whether this was due to the frequency the function was called or its size and/or complexity
    - … SWO Interrupt Trace will provide much more information on the behavior of exceptions

# SWO Bandwidth Issues

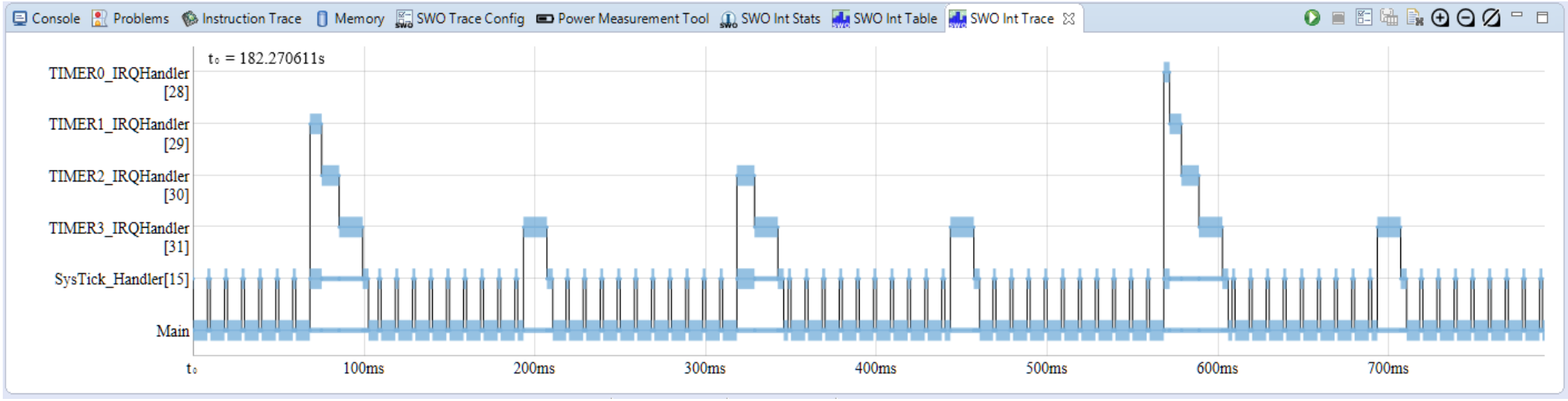
- SWO Trace captures data from the MCU and is non intrusive to the running application
- However when capturing high frequency events, data sizes become large and bandwidth issue can arise within the debug chain
- Therefore for best results:
  - Stop capture when data is no longer required
  - Avoid running high bandwidth operations at the same time
    - For example: Simultaneous Profiling and Interrupt capture should be avoided

# SWO Trace – Interrupt Views – Pro Edition

SWO Int Table

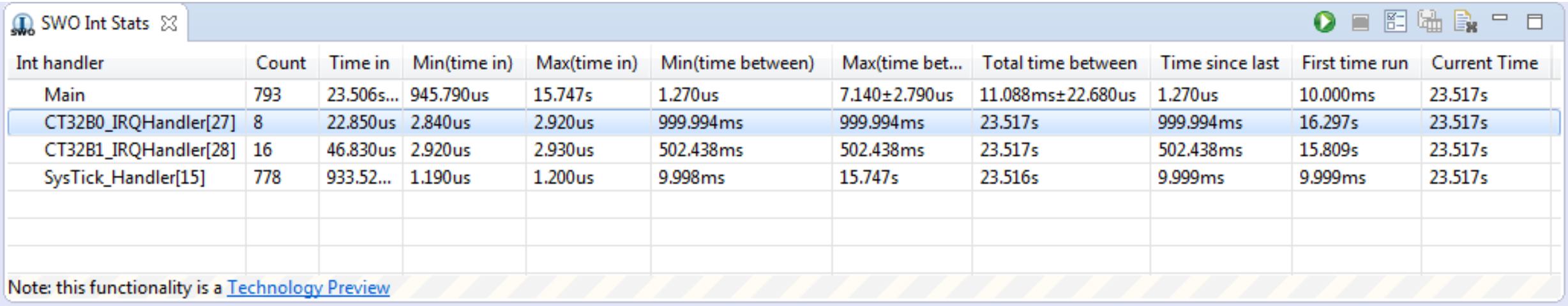
| Index | ID | Event | Handler | Time | Ticks |
|---|---|---|---|---|---|
| 3098 | 35 | EXIT | I2C1_IRQHand... | 16.048s | 3273746073 |
| 3097 | 35 | ENTRY | I2C1_IRQHand... | 16.048s | 3273745862 |
| 3096 | 0 | RETURN | | 16.048s±3.294us | 3273740833 |
| 3095 | -3 | OVERFLOW | SWO Overflow | 16.048s±3.294us | 3273740833 |
| 3094 | 0 | RETURN | | 16.048s±3.294us | 3273740833 |
| 3093 | 15 | EXIT | SysTick_Handler | 16.048s | 3273740833 |
| 3092 | 15 | ENTRY | SysTick_Handler | 16.048s | 3273740817 |
| 3091 | 31 | EXIT | TIMER3_IRQH... | 16.048s | 3273740810 |
| 3090 | 31 | ENTRY | TIMER3_IRQH... | 16.034s | 3270940679 |

Interrupt Table and Graph views available in Pro Edition only



# SWO Trace – Interrupt Views – Free and Pro Editions

- Interrupts Stats
  - Continuous count (and other stats) of all interrupts
- Benefits
  - Determine time spent in interrupt handlers
  - Optimization of interrupt handlers

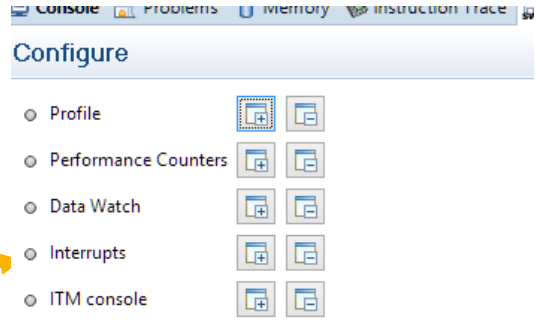


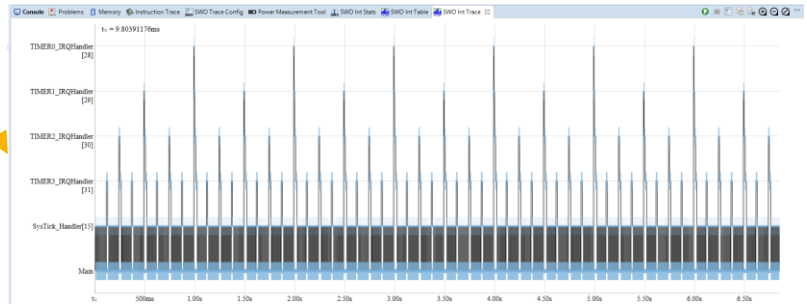

| Int handler | Count | Time in | Min(time in) | Max(time in) | Min(time between) | Max(time bet... | Total time between | Time since last | First time run | Current Time |
|---|---|---|---|---|---|---|---|---|---|---|
| Main | 793 | 23.506s... | 945.790us | 15.747s | 1.270us | 7.140±2.790us | 11.088ms±22.680us | 1.270us | 10.000ms | 23.517s |
| CT32B0_IRQHandler[27] | 8 | 22.850us | 2.840us | 2.920us | 999.994ms | 999.994ms | 23.517s | 999.994ms | 16.297s | 23.517s |
| CT32B1_IRQHandler[28] | 16 | 46.830us | 2.920us | 2.930us | 502.438ms | 502.438ms | 23.517s | 502.438ms | 15.809s | 23.517s |
| SysTick_Handler[15] | 778 | 933.52... | 1.190us | 1.200us | 9.998ms | 15.747s | 23.516s | 9.999ms | 9.999ms | 23.517s |

Note: this functionality is a Technology Preview

# SWO Interrupt Trace #1

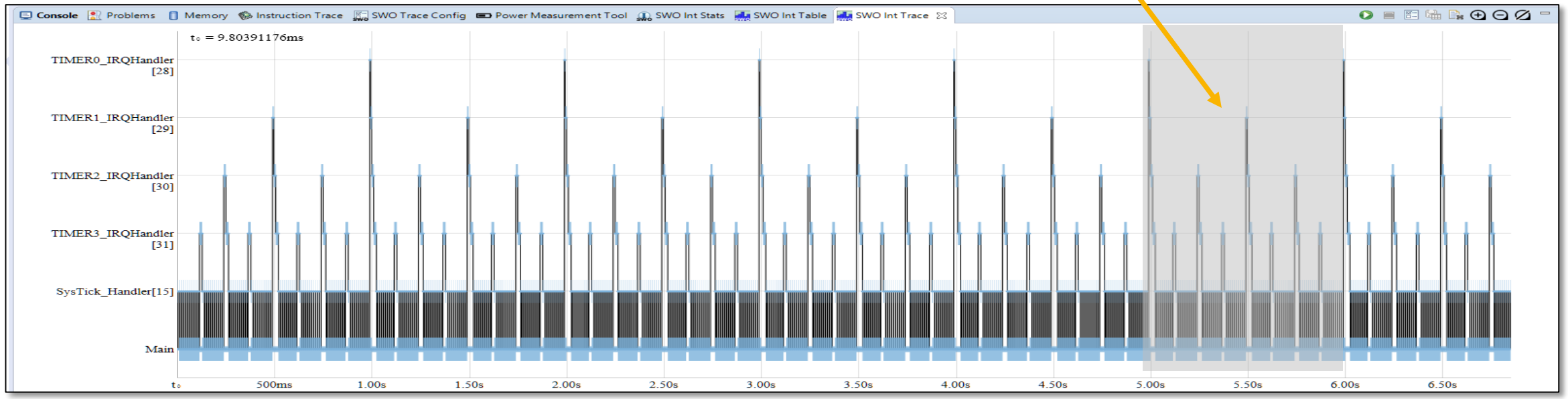
- To start Interrupt Trace on the example application:
  - Click the run control Resume button to start execution (if required)
    - Note: Ensure Profiling capture is stopped!
  - From the Console view, select the SWO Trace Config
    - Click the Interrupts + button to create and go to the SWO Interrupts view
    - Start collecting data by clicking the Interrupts Run button
    - After a few seconds click the Interrupts Stop button
      - Large amounts of data will be captured so limiting capture time to < 10 seconds is recommended
    - Initially only a 4.5 ms view will be displayed, click reset Zoom to see all
      - To zoom in use the + button
      - To zoom out use the - button
      - To investigate a section, drag the mouse pointer to select a block to zoom
    - The display will show a timeline graph showing time spent in each exception



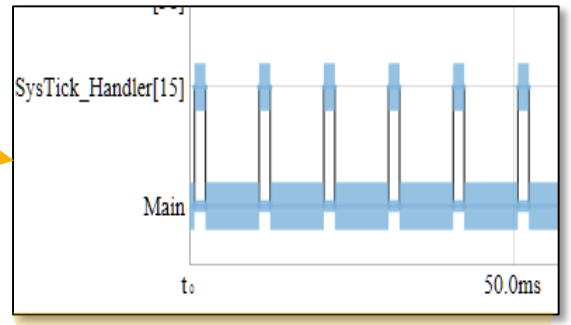
# SWO Interrupt Trace #2

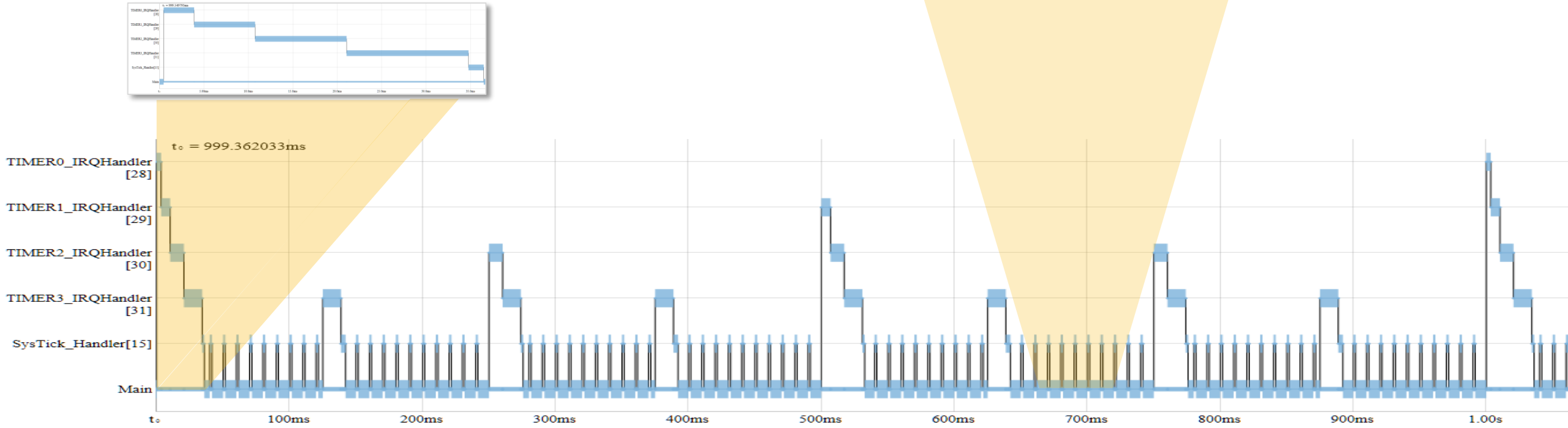
- Five exceptions are recorded, and we can observe:
  - Timer0 exception occurs once per second
  - Timer1 exception occurs twice per second
  - Timer2 exception occurs 4 times per second
  - Timer3 exception occurs 8 times per second
  - SysTick exception occurs more often than can be explored from this view
- To explore further - select drag with the mouse across a 1 second period (next slide)



# SWO Interrupt Trace #3

- Systick is now clearly visible as a 10ms event
  - We can also begin to see the time spent within each handler
  - Also the proportion of time ‘free’ to our main application
- If we zoom in further onto the display
  - We can accurately measure the time spent within a handler (next slide)

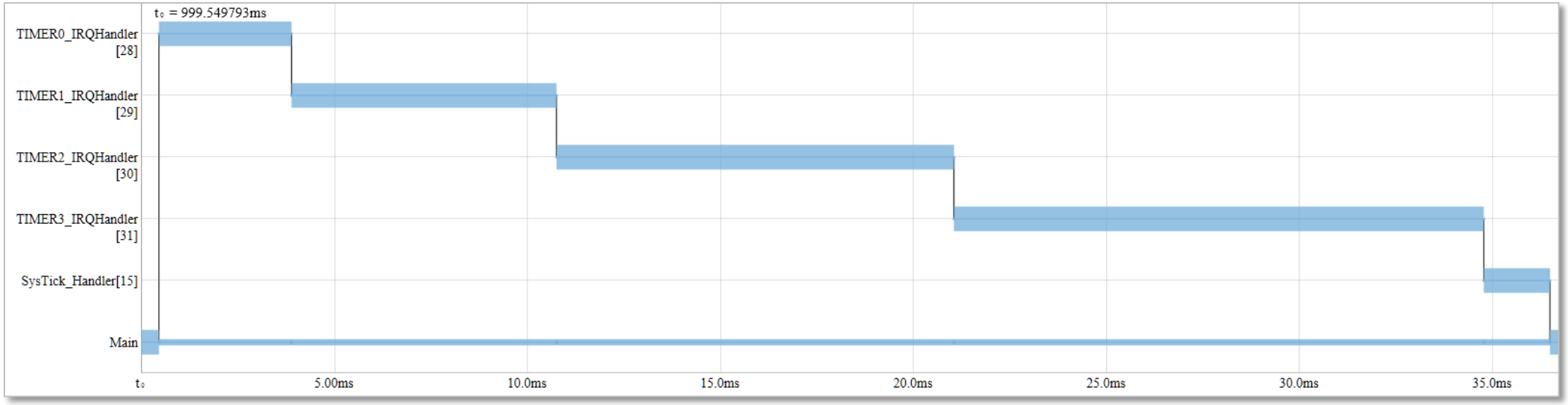


# SWO Interrupt Trace #4

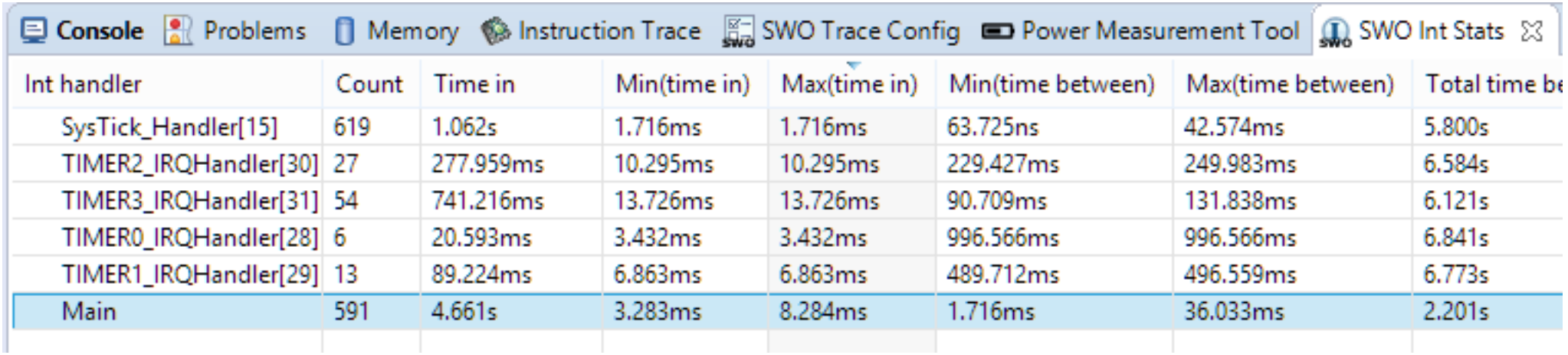
- Now measure the times spent in each handler:
  - Timer0 handler executes in ~3ms (3.44ms with more careful measurement)
  - Use the mouse to identify vertical locations on the graph, time information will be displayed in the top right corner of the view
  - Alternatively, hold <shift> and drag the display with the mouse to accurately set the origin
  - Timer1 handler executes in 6.9ms, Timer2 handler executes in 10.3ms etc.
    - Systick handler in 1.7ms
- Note how these measurements relate to the delay loop size built into each handler
  - These loops are used to represent work in the handler and have been set to artificially large values for ease of visibility!

```
39 #define TIMER0_DELAY 100000
40 #define TIMER1_DELAY 200000
41 #define TIMER2_DELAY 300000
42 #define TIMER3_DELAY 400000
43 #define SYSTICK_DELAY 50000
```



# SWO Interrupt Stats

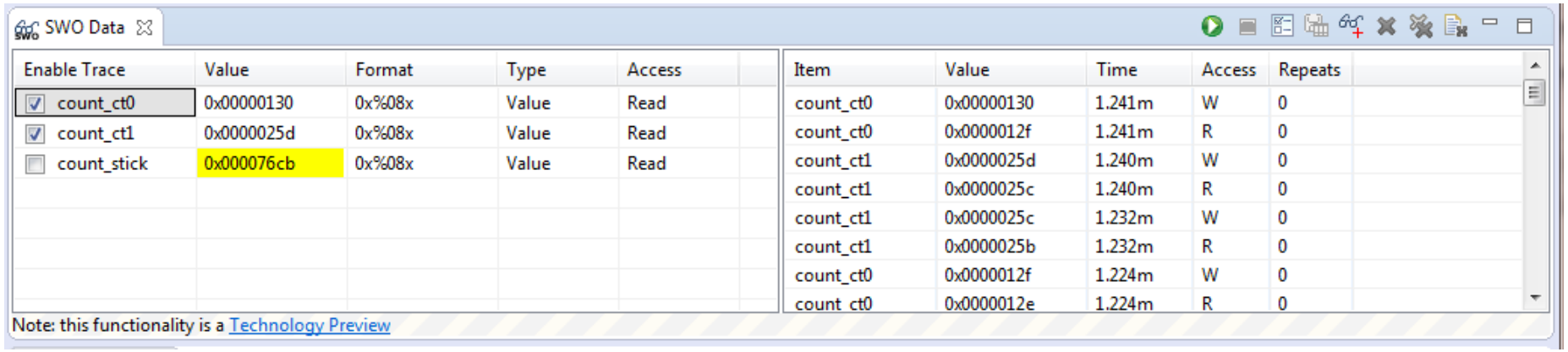
- Capturing Interrupt Trace also populates an SWO Int Stats View
- Switch to the SWO Int Stats View and compare the Stats with our measured values
  - Note that for linear functions the Min (time in) and Max (time in) will be the same
    - this is not true for time spent in Main (which is the interrupted code)
  - Note the Min (time between) and Max (time between) is not guaranteed to be same
    - This relates to interrupt default priorities and pre-emption




| Int handler | Count | Time in | Min(time in) | Max(time in) | Min(time between) | Max(time between) | Total time be |
|---|---|---|---|---|---|---|---|
| SysTick_Handler[15] | 619 | 1.062s | 1.716ms | 1.716ms | 63.725ns | 42.574ms | 5.800s |
| TIMER2_IRQHandler[30] | 27 | 277.959ms | 10.295ms | 10.295ms | 229.427ms | 249.983ms | 6.584s |
| TIMER3_IRQHandler[31] | 54 | 741.216ms | 13.726ms | 13.726ms | 90.709ms | 131.838ms | 6.121s |
| TIMER0_IRQHandler[28] | 6 | 20.593ms | 3.432ms | 3.432ms | 996.566ms | 996.566ms | 6.841s |
| TIMER1_IRQHandler[29] | 13 | 89.224ms | 6.863ms | 6.863ms | 489.712ms | 496.559ms | 6.773s |
| Main | 591 | 4.661s | 3.283ms | 8.284ms | 1.716ms | 36.033ms | 2.201s |

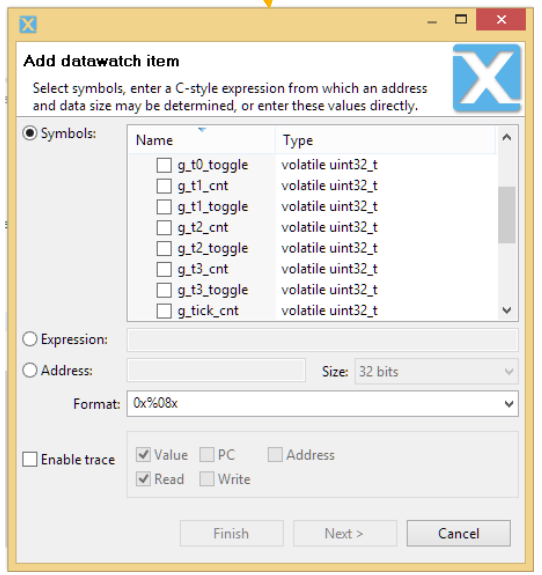
# SWO Trace – DataWatch

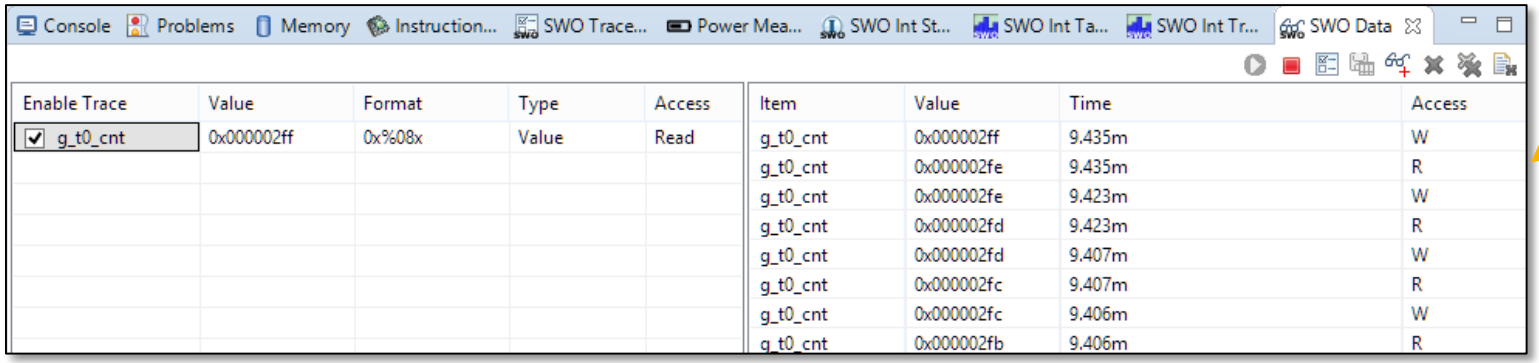
- Dynamic memory accesses
  - Read and write to target memory without stopping CPU
  - Non-intrusive
  - Reads done on periodic basis (by default)
  - Unlimited number of addresses
  - Allows modifications to parameters in real time
- Datawatch Trace
  - Capture all accesses to memory location, without stopping CPU
  - Can trace up to 4 locations concurrently
    - 1 only in Free Edition
- Benefits
  - Monitor and analyse memory accesses
  - Identify ‘rogue’ memory accesses



# SWO Trace – DataWatch Example #1

- To start Data Watch Trace on the example application:
  - Click the run control Resume button to start execution (if required)
    - Note: Ensure Profiling and Interrupt capture is stopped!
  - From the Console view, select the SWO Trace Config view (as before)
  - Click the Data Watch + button to create and go to the SWO Data view
  - To watch a global variable or address, click
    - In this example, select a variable g_t0_cnt and click Finish
    - Start monitoring by clicking the DataWatch run button
      - Log the values and access by checking the box (this counter is incremented by the Timer0 handler)



# SWO Trace – DataWatch Example #2

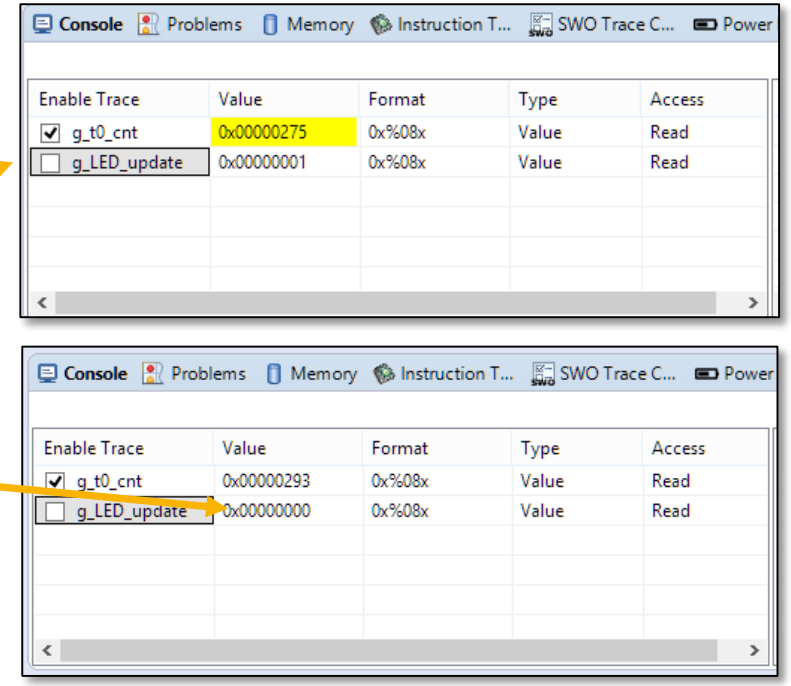
- DataWatch can also be used to write values to live global variables
- This can be used to changing program behavior while executing
  - or to force a particular condition

For example:

- Add the global variable g_LED_update to the DataWatch view
- To edit a value:
  - Click inside the value field
  - Write in a new value of 0
    - Observe the LED will stop updating on the LPCXpresso Board
      - Code in main checks this variable before performing an LED Update

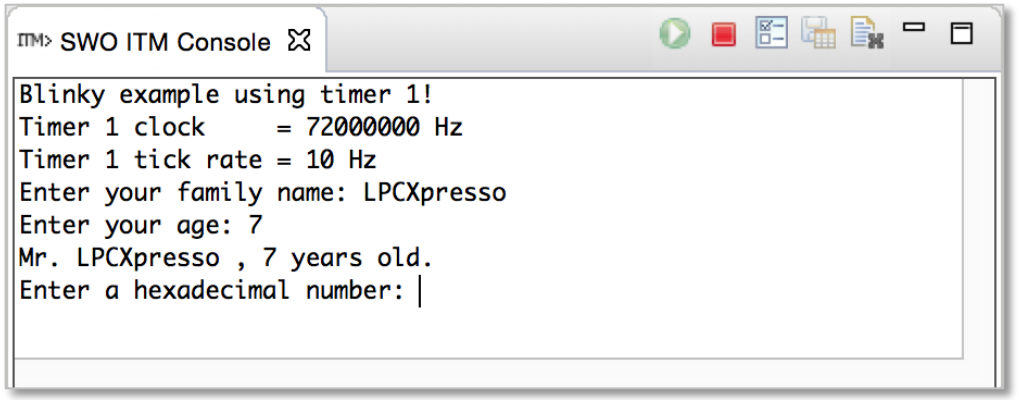
```
if (g_LED_update) {
    Board_LED_Set(0, (bool) (LED_colour & 1));
    Board_LED_Set(1, (bool) ((LED_colour >> 1) & 1));
    Board_LED_Set(2, (bool) ((LED_colour >> 2) & 1));
}
```

  - Now restore the value of the variable to 1
    - observe the LED flashing sequence restarts



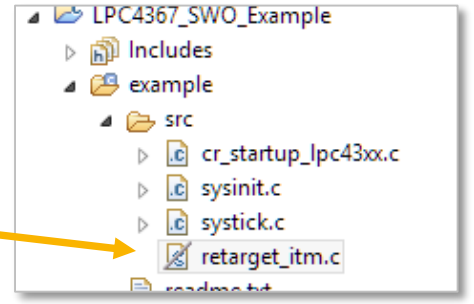
# SWO Trace – ITM Printf

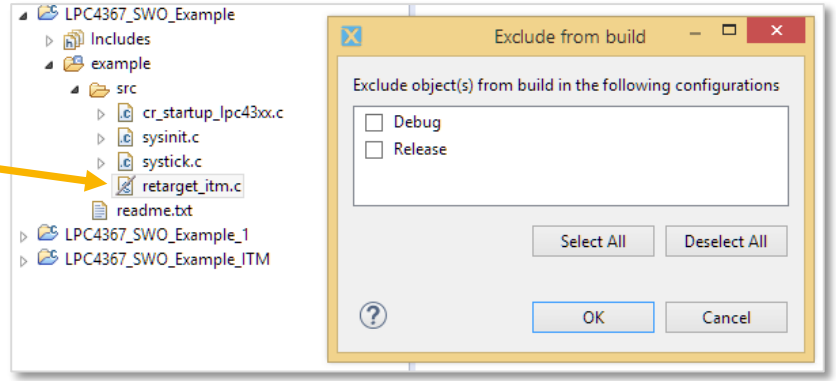
- Instrumentation Trace Macrocell (ITM) block provides a mechanism for sending data from your target to the debugger via the SWO trace stream
- LPCXpresso IDE allows user to redirect printf/scanf data by reimplementing low level Redlib function __sys_write / __sys_readc
  - Newlib reimplementation also possible
- Unlike semihosting, this scheme is both low bandwidth and does not halt the MCU to transfer data
  - https://www.lpcware.com/content/faq/lpcxpresso/how-use-itm-printf



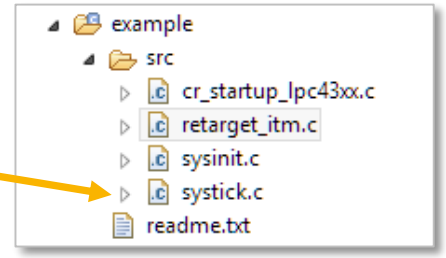
# SWO Trace – ITM Printf Example #1

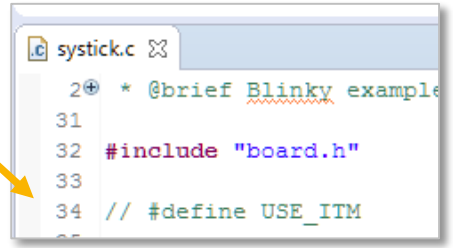
- To use ITM to support printf, some support code is required:
  - This is already included in the LPC4367_SWO_Example via the file ‘retarget_itm.c’
    - However it has been excluded from the build – to restore:
    - Right click on the file in the Project Explorer
    - Select Resource Configuration -> Exclude from Build
    - Uncheck Debug and Release and click OK



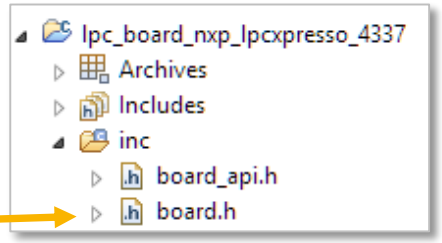

- Additionally, some edits are required to our project:
  - Ensure any existing debug session is Terminated
  - From the example project locate the source file systick.c
  - On line 34 - uncomment the statement - #define USE_ITM
    - This will enable a printf statement within the example to be compiled

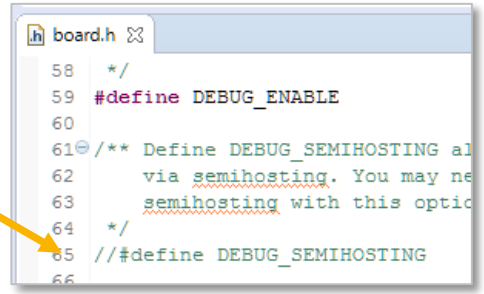


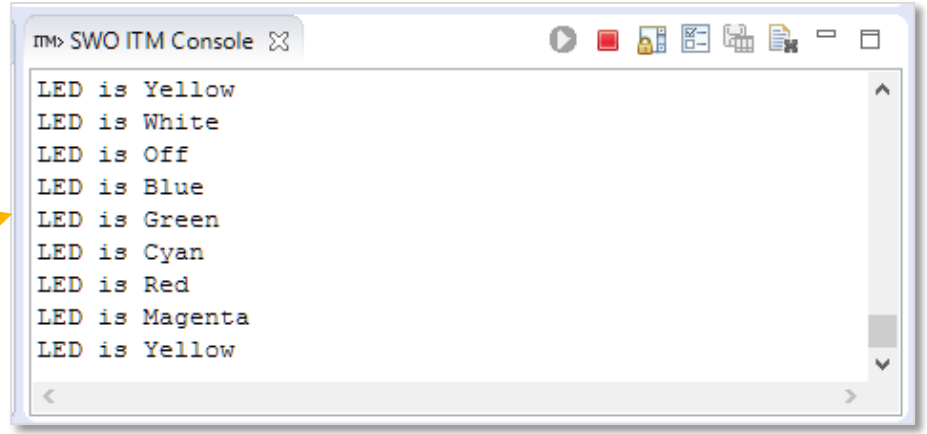
# SWO Trace – ITM Printf Example #2

- Finally we must tell the board Library code that we wish to use Semihosting
  - Locate the file board.h within the lpc_board_nxp_LPCXpresso4337 lib project
  - On line 65 – uncomment the statement - #define DEBUG_SEMIHOSTING
  - Now we can re-select our project within the Project Explorer view
  - Click Debug from the Quick Start view
    - Any unsaved files will be saved at this point




- To start ITM Printf on the example application:
  - Click the application Resume button to start execution (if required)
  - From the Console view, select the SWO Trace Config view (as previously)
  - Click the ITM Console + button to create and go to the ITM Console view
  - Click Start ITM Trace button to begin:
    - The current colour of the LED will be printed to the ITM console



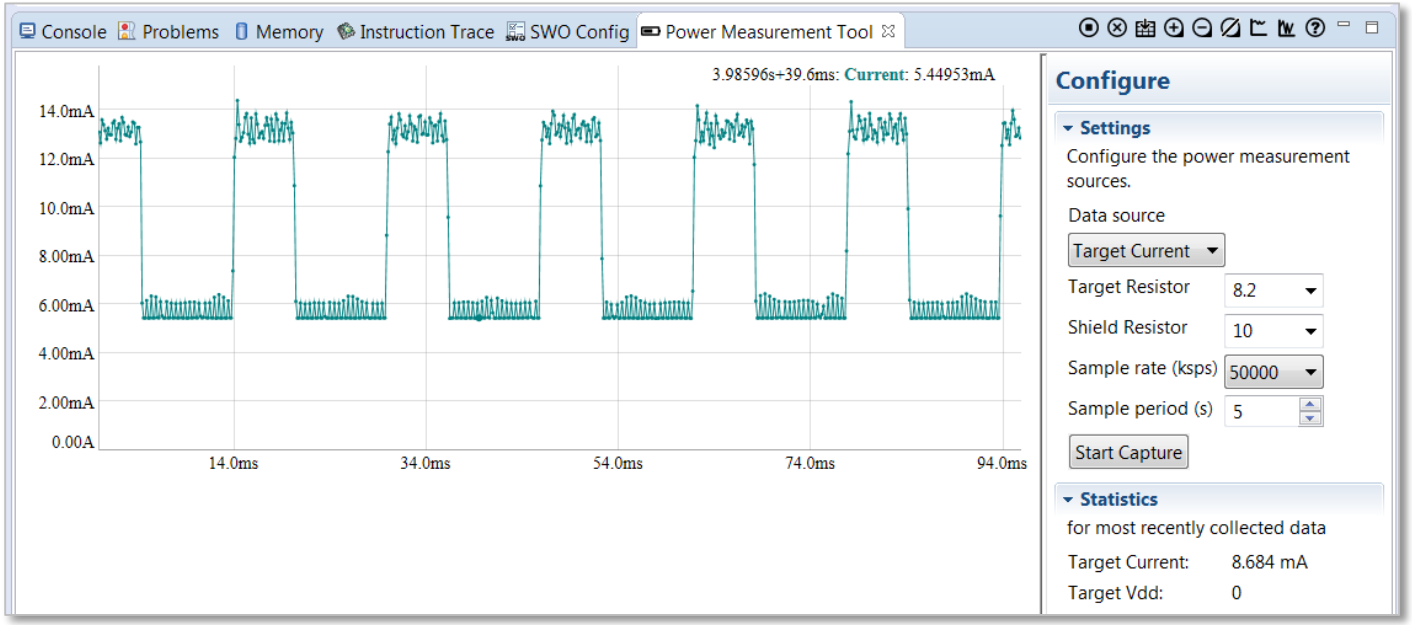

This scheme can be used to continuously generate status information that is only visible when debug tools are connected

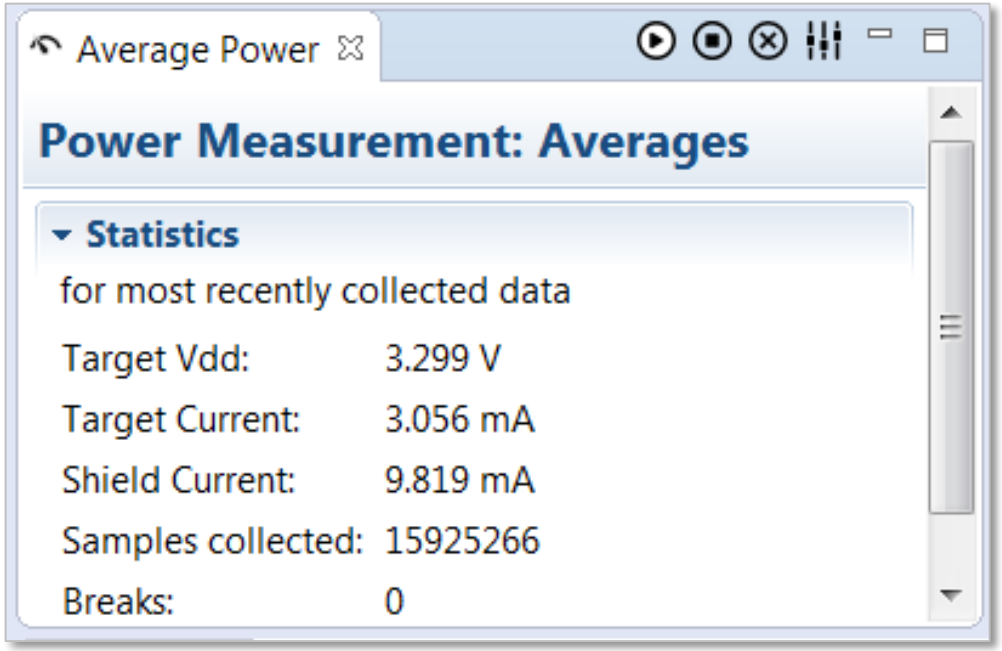
# POWER MEASUREMENT

# Power Measurement

- Most LPCXpresso V3 boards (and NXP LPC shields) provide support for power measurement
  - High speed ADC, measuring voltage across sense resistor via instrumentation amp

- LPC-Link2 CMSIS-DAP firmware provide USB channel for transmitting this data back to host – separately from debug channel

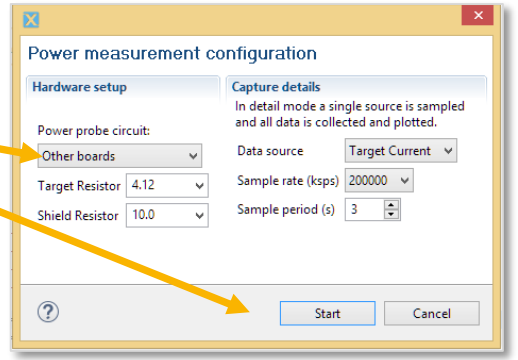


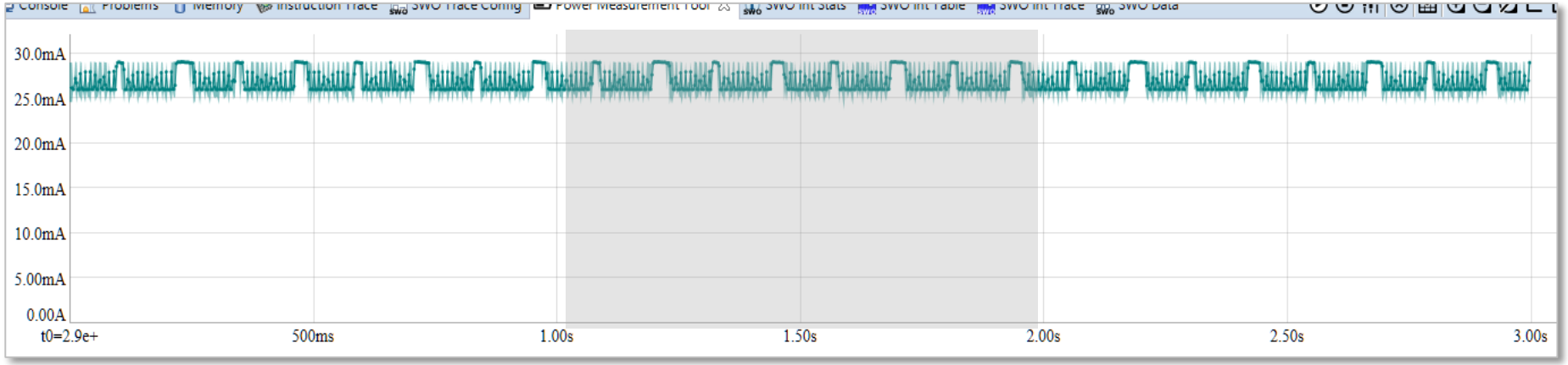
# Power Measurement – Example #1

- In this example we will explore the power consumed by the MCU executing our example code
  - First Select the Power Measurement Tool view (bottom right of IDE window)
  - Click the ‘configure the power probe’ button to generate the configuration dialogue

  - Select ‘Other boards’ from the Power Probe Circuit and Click start

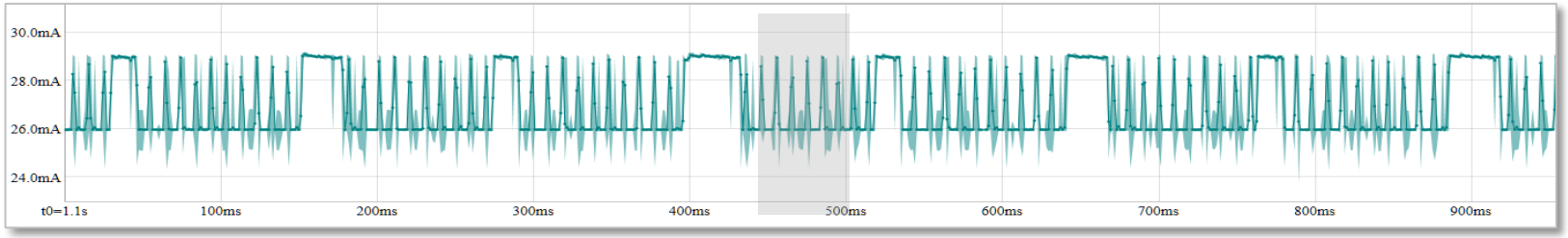



    - A default 3 second sample of MCU power usage will be captured
      - Mouse click drag horizontally to zoom in the X axis
      - Mouse click drag vertically to zoom in the Y axis
    - Now, explore a 1 second portion of the graph (next slide)

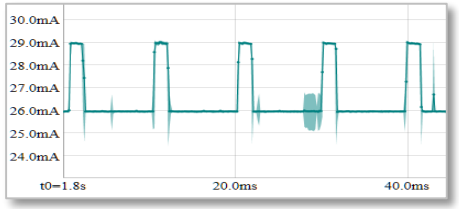


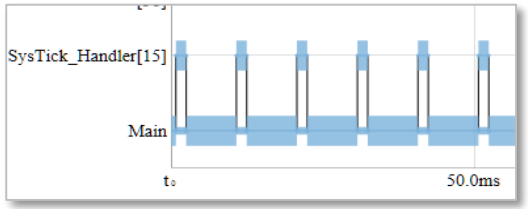
# Power Measurement – Example #2

- Observe the frequency and size of the peaks in the graph




  - Zoom in further and contrast this output with our exception trace graph recorded earlier (below)



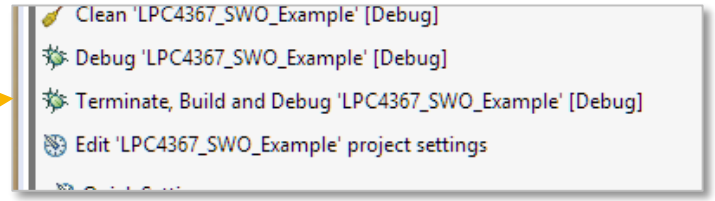

  - A clear match between our exceptions frequency and power consumption can be observed

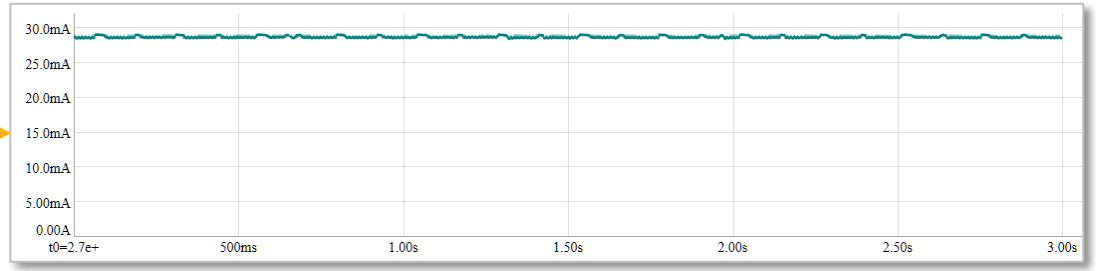
- This behavior is due to our main application entering a sleep state at the end of the loop in Main()
  - The CPU is woken when any exception occurs and this is reflected in the MCU power consumption

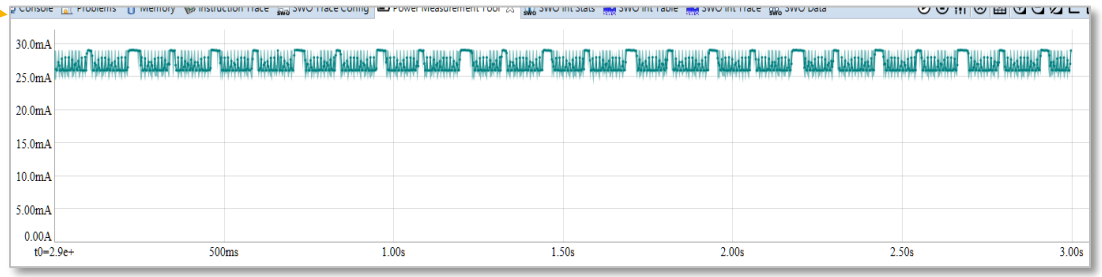
# Power Measurement – Example #3

- To see the effect of the WFI statement on power usage:
  - Locate the __WFI(); statement within the Systick.c file
  - Comment out the line and save the file
  - Click the Quick Start View line to ‘Terminate, Built and Debug’
    - This is a shortcut for help the edit code and retest cycle
  - re-capture the power consumption of the MCU
    - Contrast this result with the previous trace
    - Without __WFI();
    - With __WFI();

```
257             }
258             LED_colour++;
259             LED_colour &= 7;
260         }
261 //      __WFI();
262     }
```



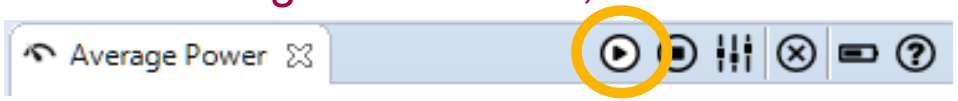




Note: in this example, the WFI command will put the MCU into the first of 4 power saving modes – Sleep. Other modes are:
- Deep Sleep
- Power Down
- Deep Power Down

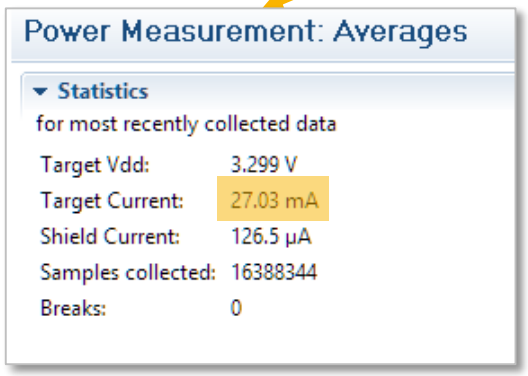
# Power Measurement – Example #4

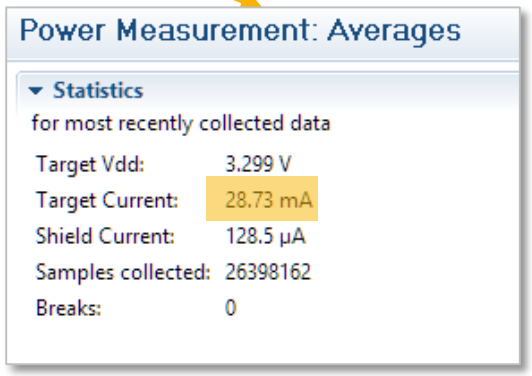
- Finally, we can also measure the average power consumption of an MCU running code.
  - Within the Power Measurement Tool View, click to launch the Power Measurement Averages view

  - Within the Average Power view, click the ‘Start Data Collection’ button




  - Compare the results achieved with __WFI() and without __WFI()



SECURE CONNECTIONS
FOR A SMARTER WORLD

# ATTRIBUTION STATEMENT