Zephyr Debugging with MCUXpresso IDE Lab

Rev2.1, August 18, 2022

Objectives

In this lab, you will learn

- How to debug a Zephyr application with MCUXpresso IDE
- How to configure Thread Awareness in your Zephyr project and see all threads through J-Link.

Pre-Requisites

This lab is written for Windows10, but Zephyr and this lab can easily be done in Linux or MacOS. This lab guide was written for Zephyr release v3.0.0.

- Follow Zephyr_Installation_Lab.pdf
- Download the latest MCUXpresso IDE, v11.6.0 or later required for some Zephyr OS features
- Terminal Program: MCUXpresso IDE integrates a terminal, or another program like PuTTY or Tera Term can be used
- Follow Zephyr_hello_world_Sample_Lab.pdf to prepare the hardware, verify environment variables, connect the terminal to the board
- Optional, review Zephyr_GDB_Lab.pdf if interested in comparing debugging in MCUXpresso IDE with GDB command line

Hardware Requirements

- MIMXRT1060-EVKB
- Micro-USB cable (may need two cables, see Zephyr_hello_world_Sample_Lab.pdf)

Running the lab

Prepare Command Line

Before using the West commands, a command window should be opened, python virtual environment activated, and environment variables set. These steps only need to be done once after a command window is opened. If already done in a previous lab, skip to the next section. See Zephyr_hello_world_Sample_Lab.pdf for more details.

1. Open a normal command prompt in Windows
2. Navigate to the zephyr directory:
   cd %userprofile%\zephyrproject\zephyr

3. Activate the python virtual environment. After activating, the prompt should now include (.venv):
   ..\.venv\Scripts\activate.bat

4. Set the environmental variables. Note, when using a command file like this to set the variables, the command file needs to be executed once anytime a new command window is opened:
   zephyr-env.cmd

5. If working on an NXP lab computer, you can restore the Zephyr source files that may have been modified in earlier labs using this Git command:
   git reset --hard

Build the Zephyr application

When building this sample application, we will add a Kconfig setting through command line to enable Zephyr Thread Awareness for debugging. In the command below, notice we add these West command arguments:

- `-DCONFIG_DEBUG_THREAD_INFO=y` : this argument overrides a Kconfig symbol at build time. The -D argument defines a symbol from the command line. The =y is selecting this Kconfig symbol CONFIG_DEBUG_THREAD_INFO to be true, which will add structures and hooks into the kernel to add Thread Awareness for debugging.
- `-DCONFIG_INIT_STACKS=y` : selecting this Kconfig symbol CONFIG_INIT_STACKS will initialize the stack memory with a known value. This enables the tools to determine the high water mark for stack usage in the application.

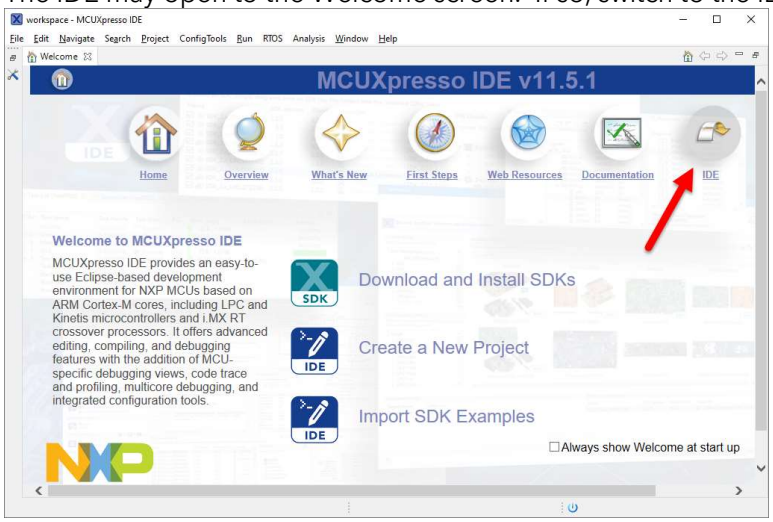
6. In the command window, build the synchronization sample application using west. NOTE: when pasting this line into the command window, be sure to get both lines below as a single command. You may need to paste the 2nd line separately to the end of the 1st line for the full command:
   west build -b mimxrt1060_evk -d ..\build-mcux samples\synchronization --pristine -- -DCONFIG_DEBUG_THREAD_INFO=y -DCONFIG_INIT_STACKS=y

Starting with MCUXpresso IDE

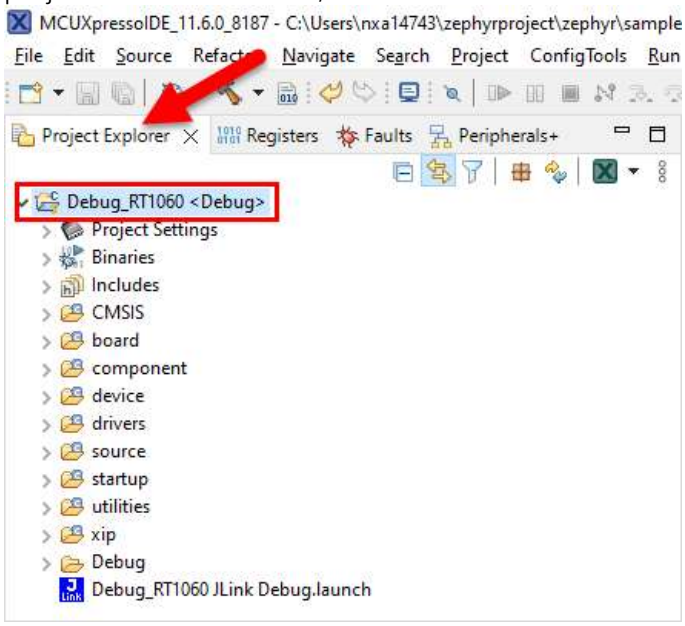
MCUXpresso IDE is NXP's free IDE for NXP's MCUs, and uses the Eclipse framework.

7. **Open MCUXpresso IDE**. If using a lab computer with multiple versions of MCUXpresso IDE installed, be sure to open the version for this lab **v11.6.0**

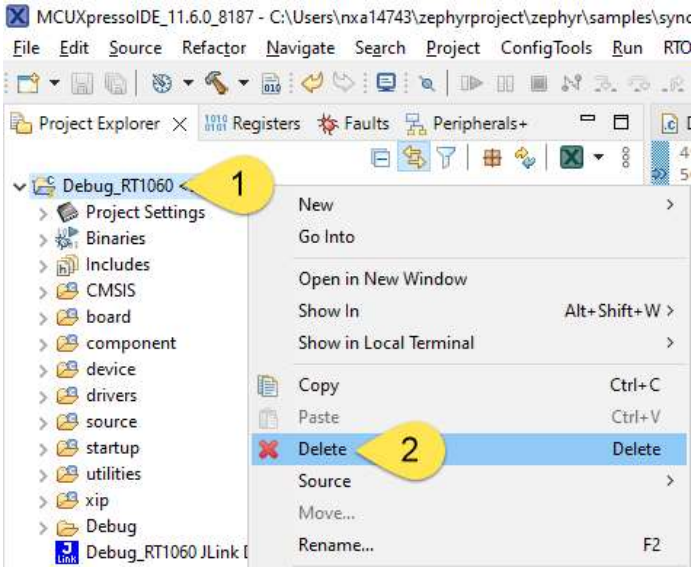
8. The IDE may open to the Welcome screen. If so, switch to the IDE.



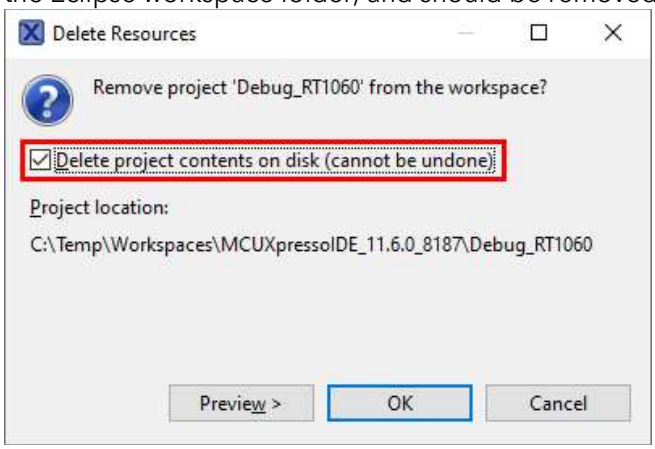
9. For an NXP lab computer that has been used before, the Debug project may have already been created. Click the **Project Explorer tab** to view the projects in the workspace. If the Debug_RT1060 project is listed like below, it should be deleted before the remaining lab steps.



10. To delete an existing project, right-click on the project name, and **select Delete**. Or use the Delete key on the keyboard.



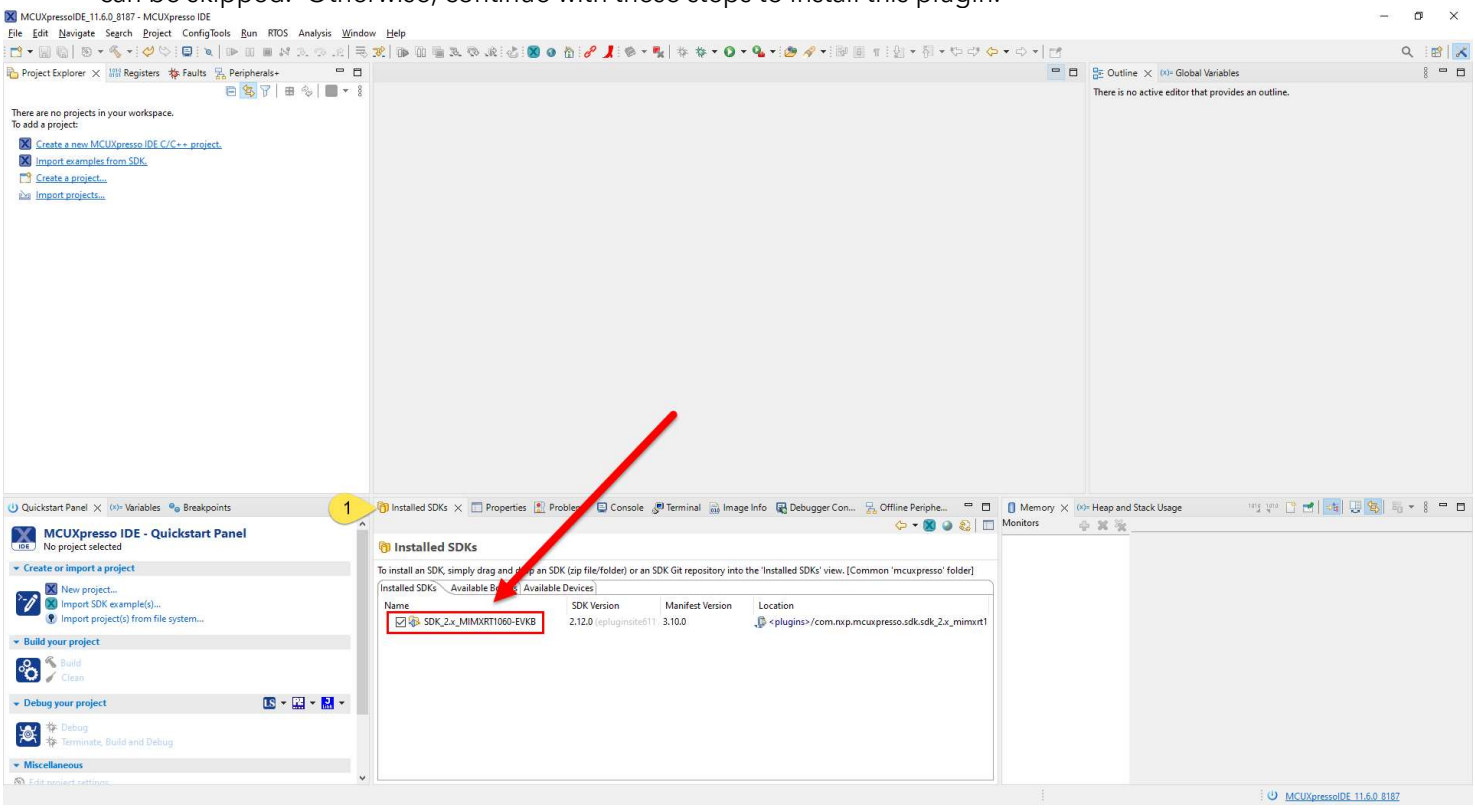
11. Be sure to check the box to "**Delete project contents on disk**", as this Debug project is included in the Eclipse workspace folder, and should be removed completely. Click OK.



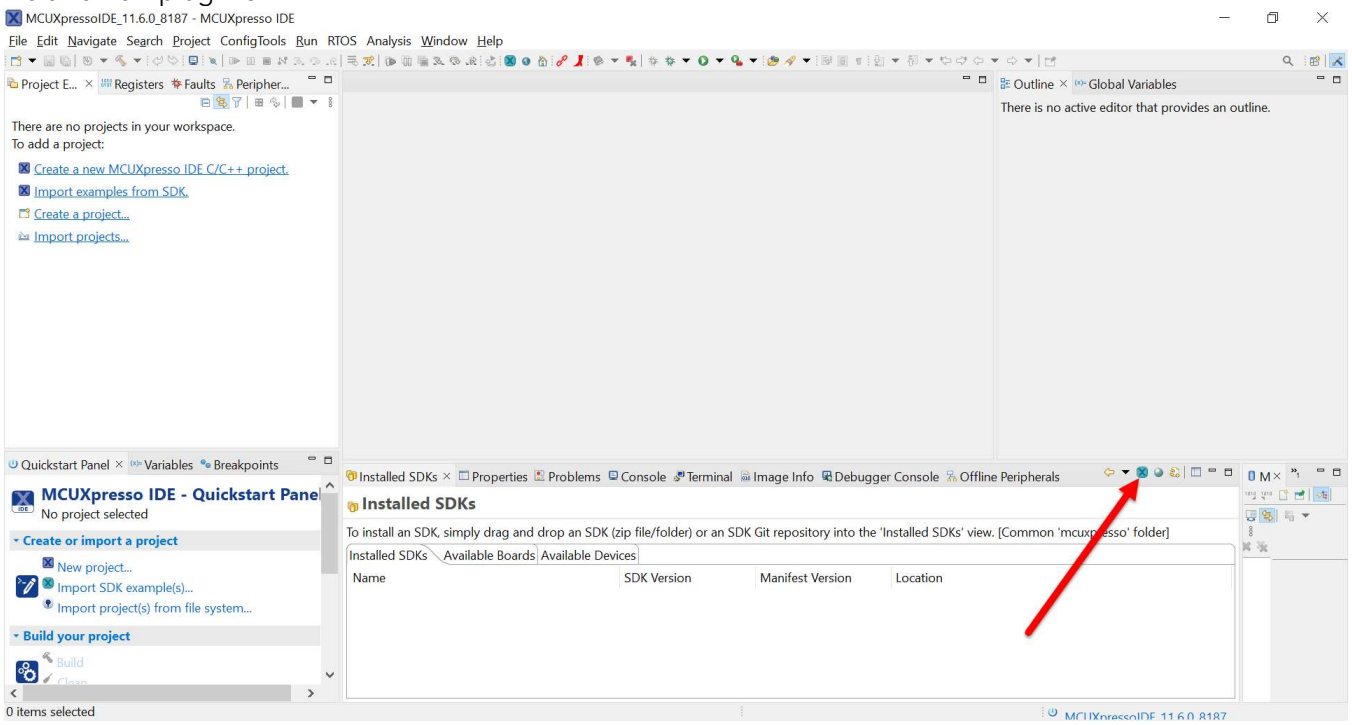
Install MCUXpresso SDK for the board/MCU

The IDE requires an SDK installed to create or debug projects for an MCU. If the required SDK is already installed, this section can be skipped. When using a lab computer, this SDK has already been installed.

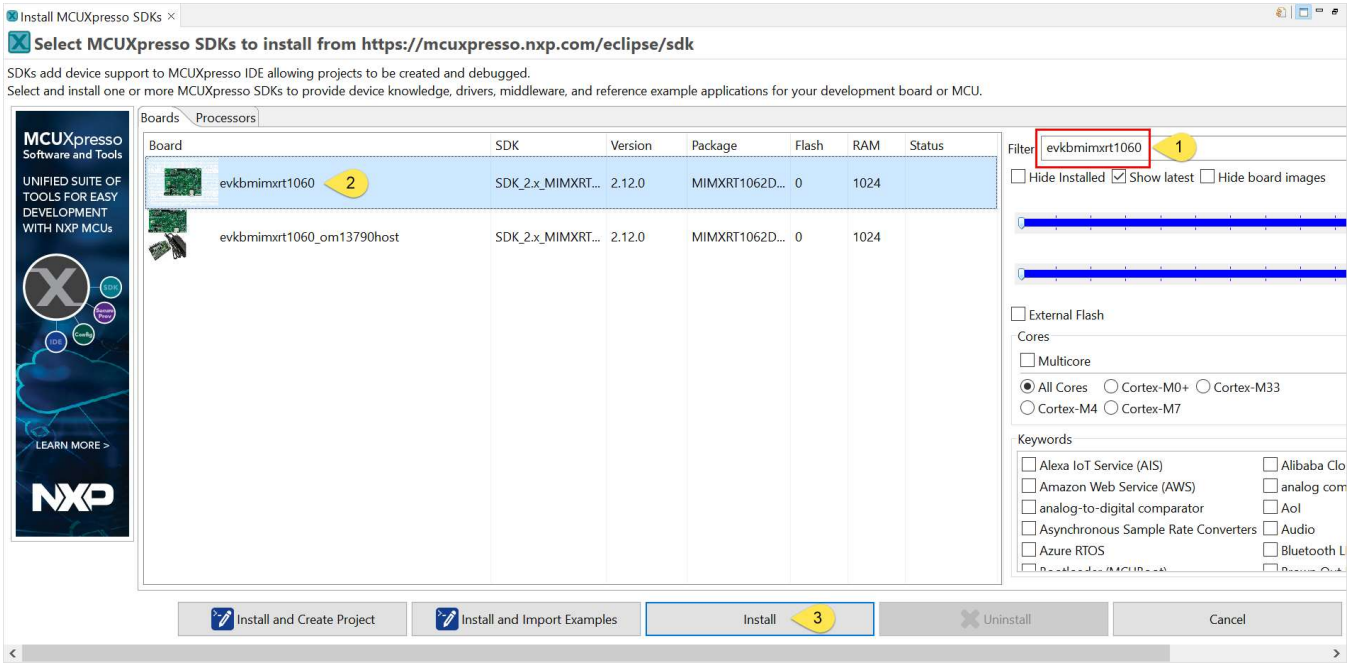
12. Confirm if the required SDK is already installed. In the IDE, find the tab for the Installed SDKs View, shown below. This lab requires the SDK for the board **MIMXRT1060-EVKB**. Note the SDK Version as well, for this lab v2.12.0 or newer is recommended. If this SDK plugin is already installed, this section can be skipped. Otherwise, continue with these steps to install this plugin.



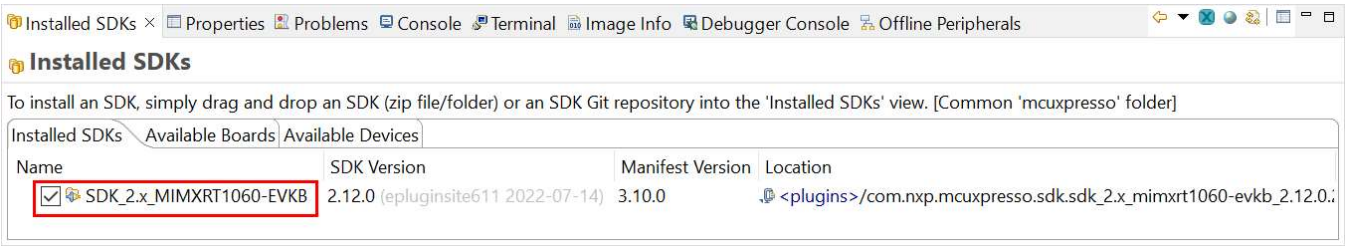
13. Install a new plugin SDK.



14. Select the SDK plugin to install for your board/device
    a. Filter for the board **evkbmimxrt1060**
    b. Select the board **evkbmimxrt1060**
    c. Click the **Install** button



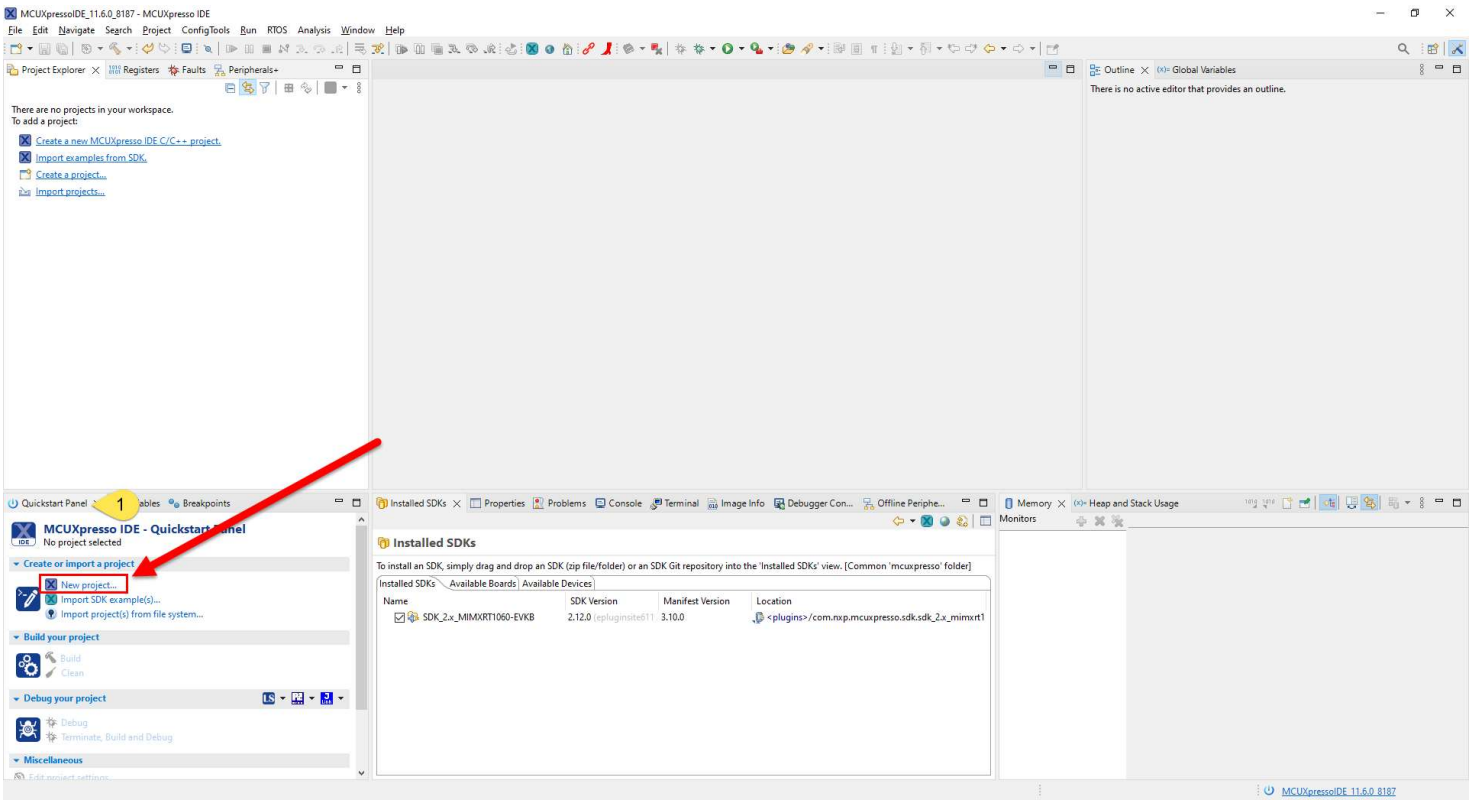
15. Please accept the licensing terms to install the SDK plugin. The plugin will be downloaded and installed in the IDE. Once finished, the SDK for the board will be shown in the Installed SDKs View like below:



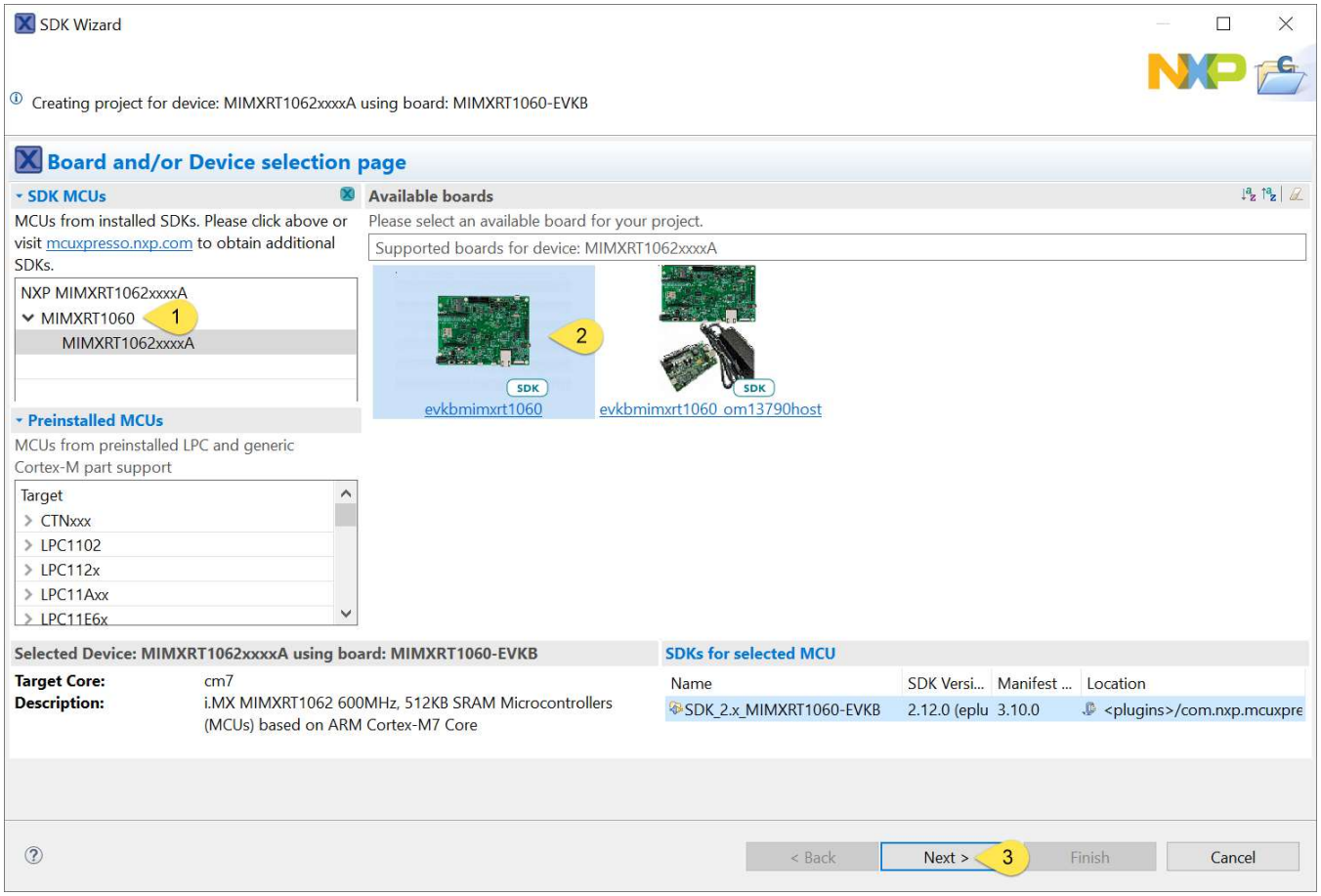
Create a project for MCUXpresso IDE

For the best debugging experience in MCUXpresso IDE, we recommend using the IDE to create a Debug project to help initialize/configure the IDE. This Debug project will not build the Zephyr application, but will be used to debug it. This method uses the IDE to generate the Debug Configuration for the MCU to gain all the debugger features from the IDE. Then the Debug Configuration is manually modified to use the Zephyr application image already built by West. The IDE can then download the Zephyr app to flash and debug it.

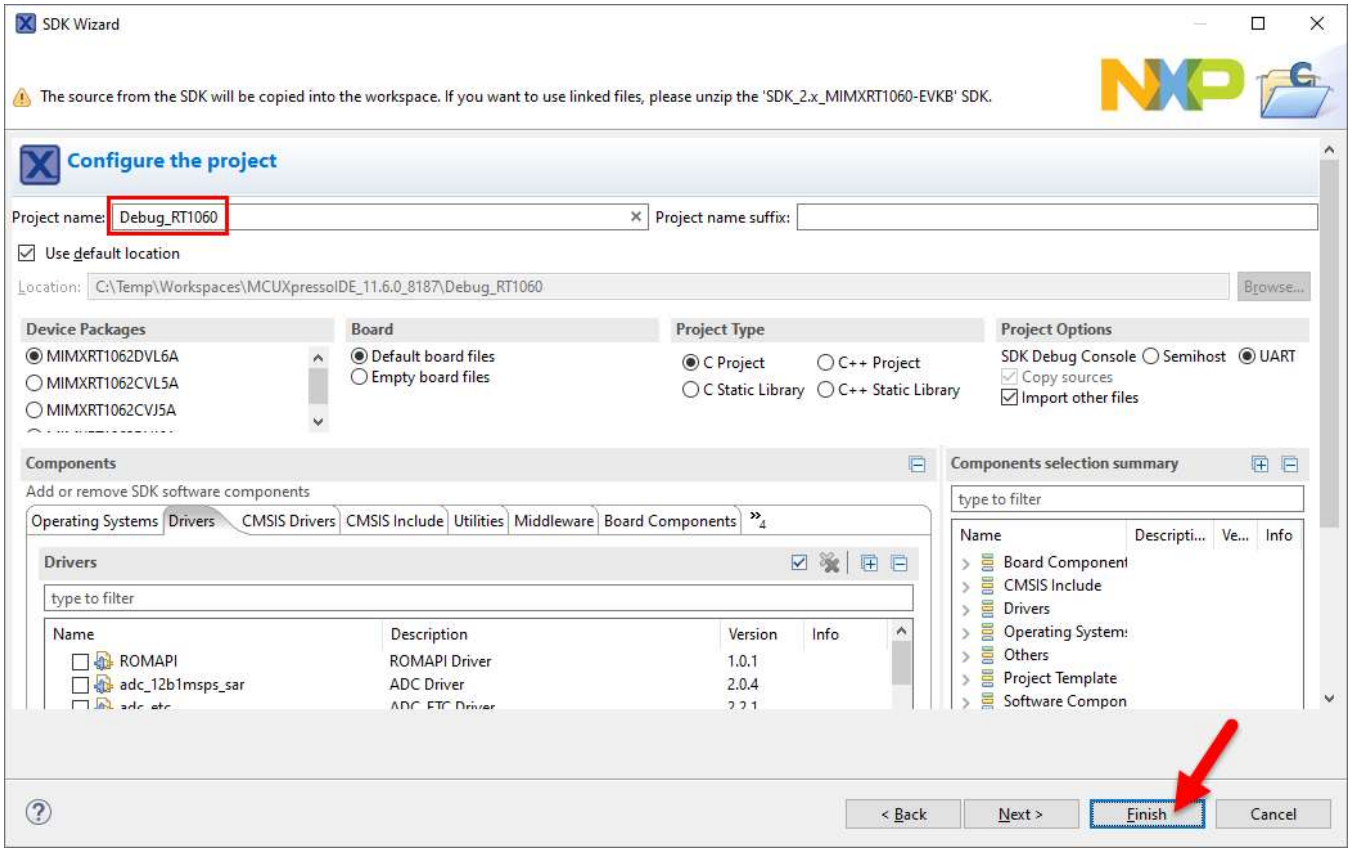
16. In the IDE, find the Quickstart Panel View, and click New Project



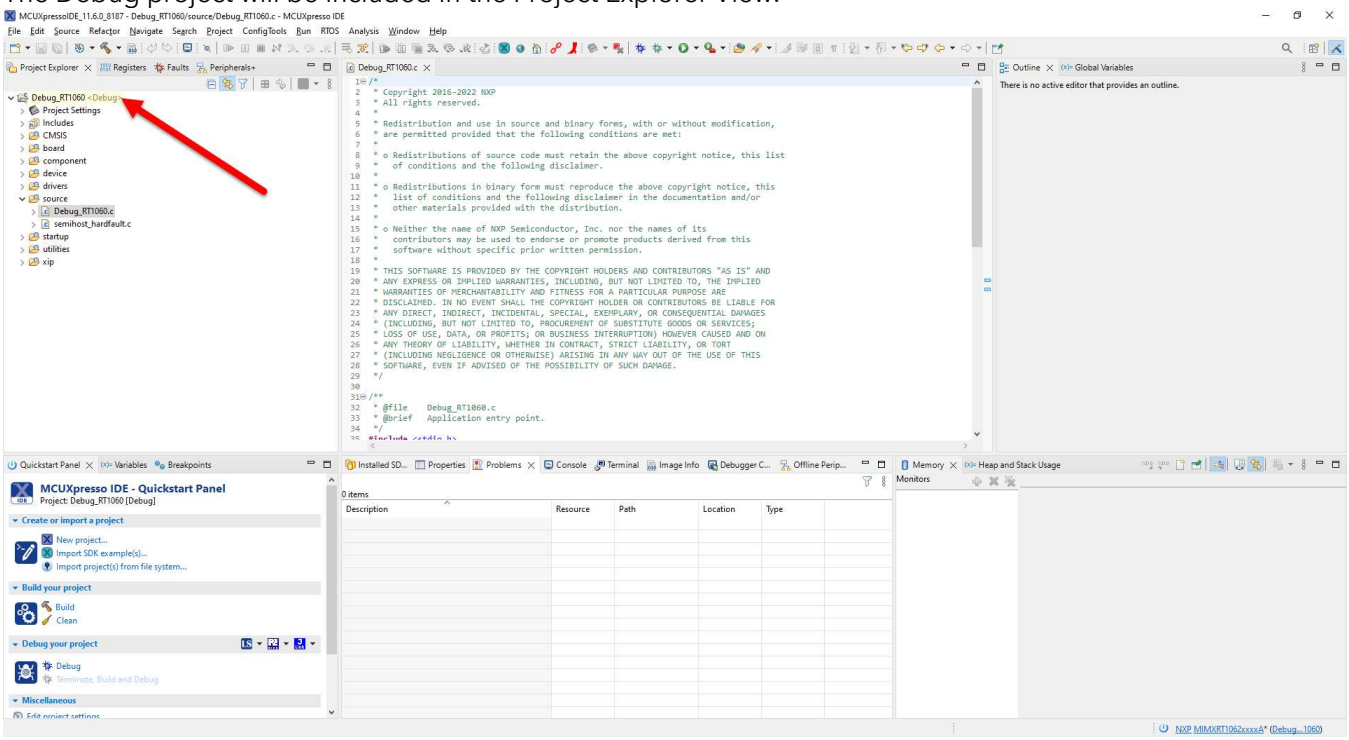
17. Select the board for the new project:
    a. It can help to select the MCU family RT1060 in the left window to filter down all the board options
    b. Select the board **evkbmimxrt1060**
    c. Click the Next button



18. Configure the new project:
    a. Name the project **Debug_RT1060**.
    b. No other changes are needed for the Debug project, and the default settings can be used. Click the Finish button.



The Debug project will be included in the Project Explorer View:

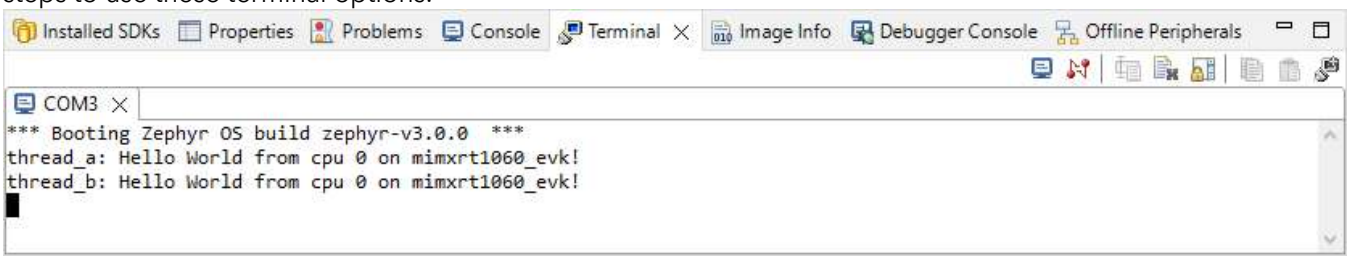


Modify the IDE project memory map

This section is not done with the RT1060, and not needed for this lab. For the RT1060, you can skip to the next section. This section is also not needed when using a Segger JLink debug probe. But other devices with a LinkServer (CMSIS-DAP) probe, it may require this step for the debugger to properly flash and debug the Zephyr application. One example where this step is required is when using a device with Cortex-M33 core, and has separate memory maps for Secure and Non-Secure code. When the memory map used by the Zephyr app differs from the default memory map in the IDE project, follow this additional guide to modify the memory map: **Zephyr_MCUXpresso_IDE_Memory_Map_Addendum.pdf**. That guide uses the i.MX RT500 MCU as an example on the MIMXRT595-EVK board.

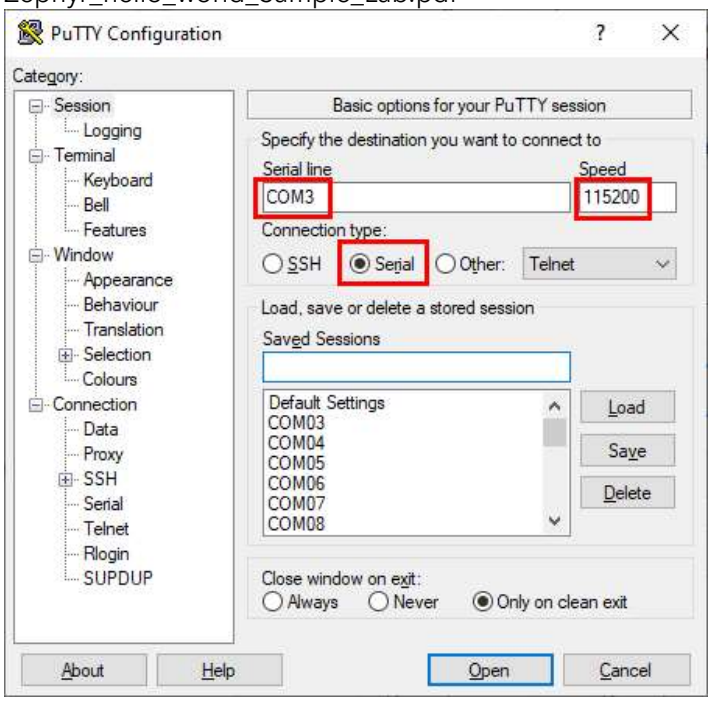
Connect a Terminal to the board

The previous lab guide Zephyr_hello_world_Sample_Lab.pdf gave steps for using PuTTY as a terminal. PuTTY can be re-used here, but Eclipse also offers a Terminal which is integrated in MCUXpresso IDE. Below are the steps to use these terminal options.

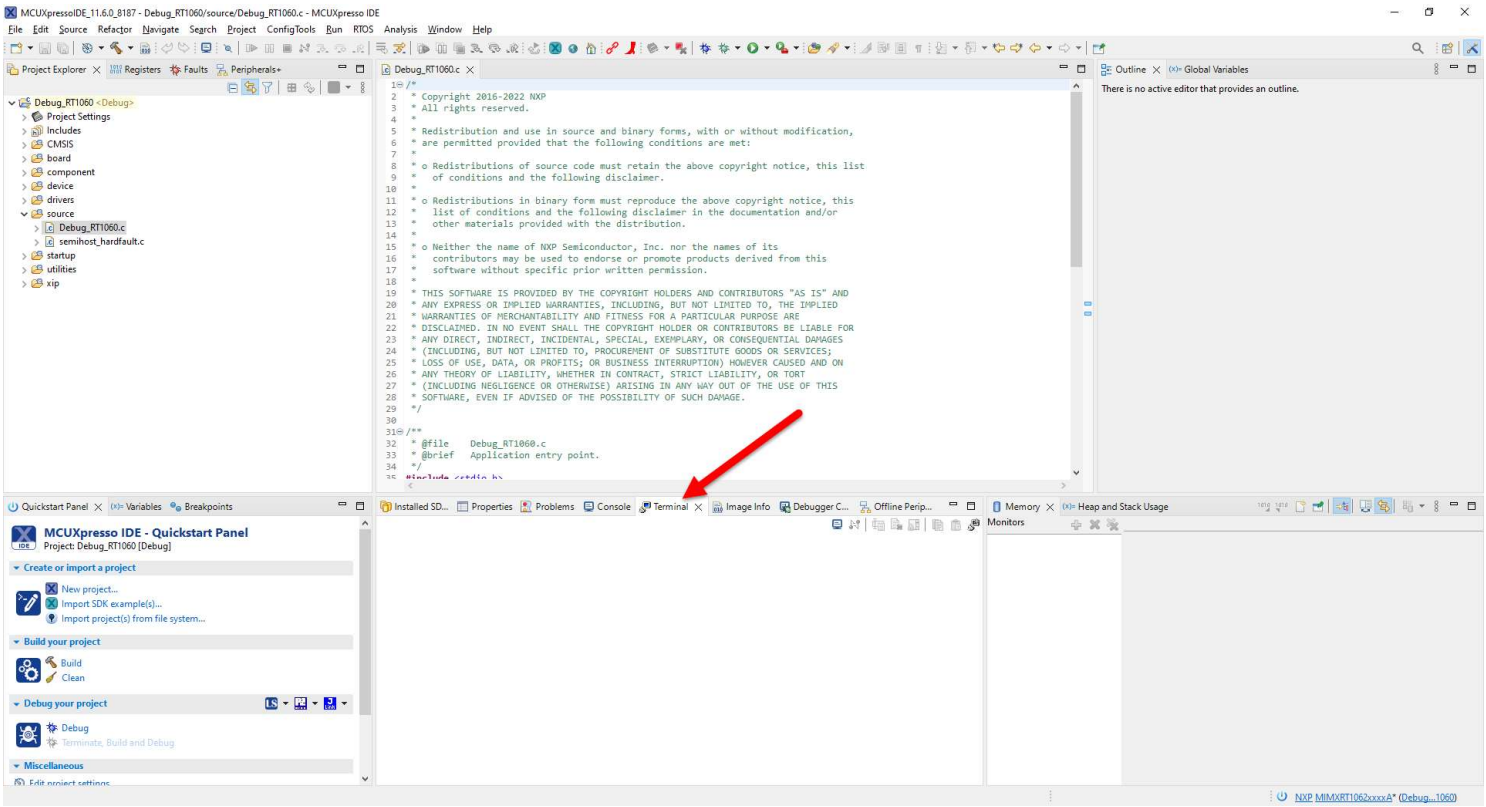


19. Ensure the board is powered, and the debug probe is connected. This lab uses the on-board debugger of MIMXRT1060-EVKB with USB connector J1 using Jlink firmware, plus powered through USB J48. For more details on preparing the hardware, see Zephyr_hello_world_Sample_Lab.pdf.

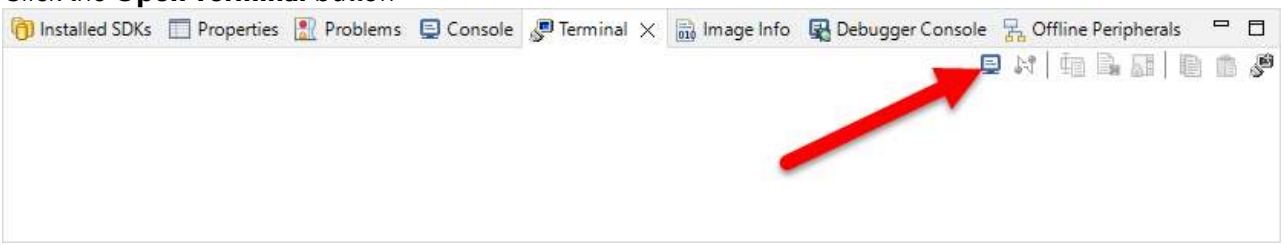
    To re-use PuTTY, connect the terminal to the board's COM port, for help see Zephyr_hello_world_Sample_Lab.pdf



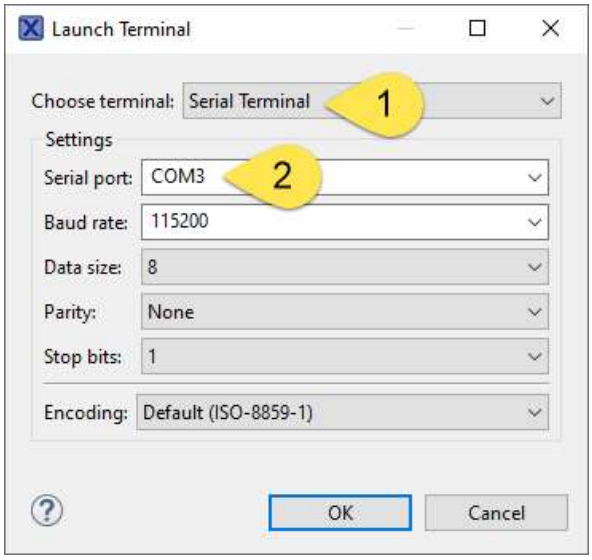
20. To use the Terminal in MCUXpresso IDE, close any other terminal programs like PuTTY, then click the **Terminal View tab**:



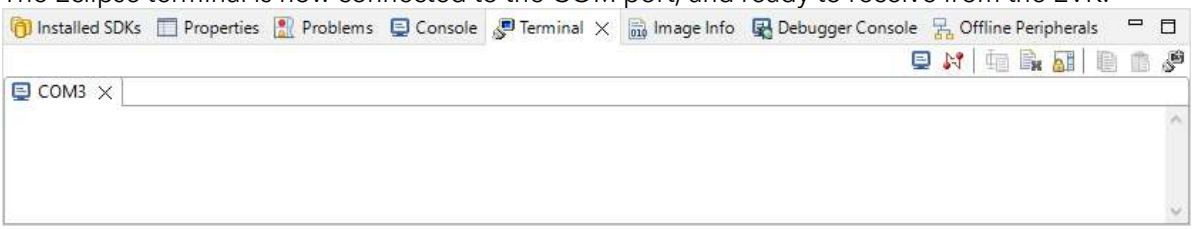
21. Click the **Open Terminal** button



22. Configure the Terminal:
    a. Change the terminal type to **Serial Terminal**
    b. The serial port should default to the correct COM port. But if multiple COM ports are available, you may need to **select the correct COM port** for the EVK.
    c. Click OK



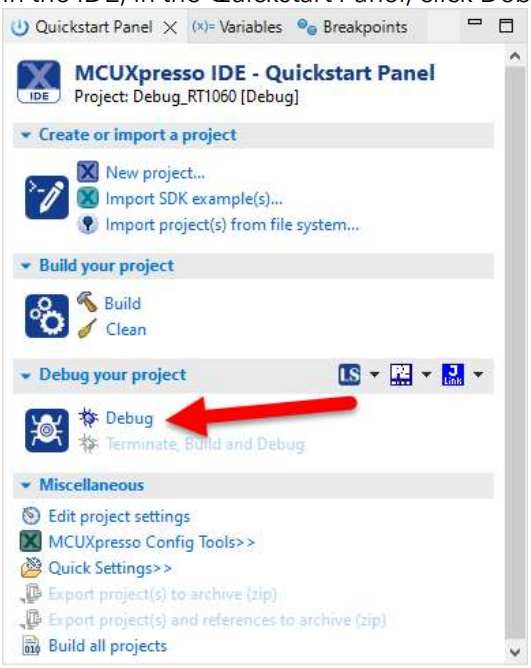
The Eclipse terminal is now connected to the COM port, and ready to receive from the EVK:



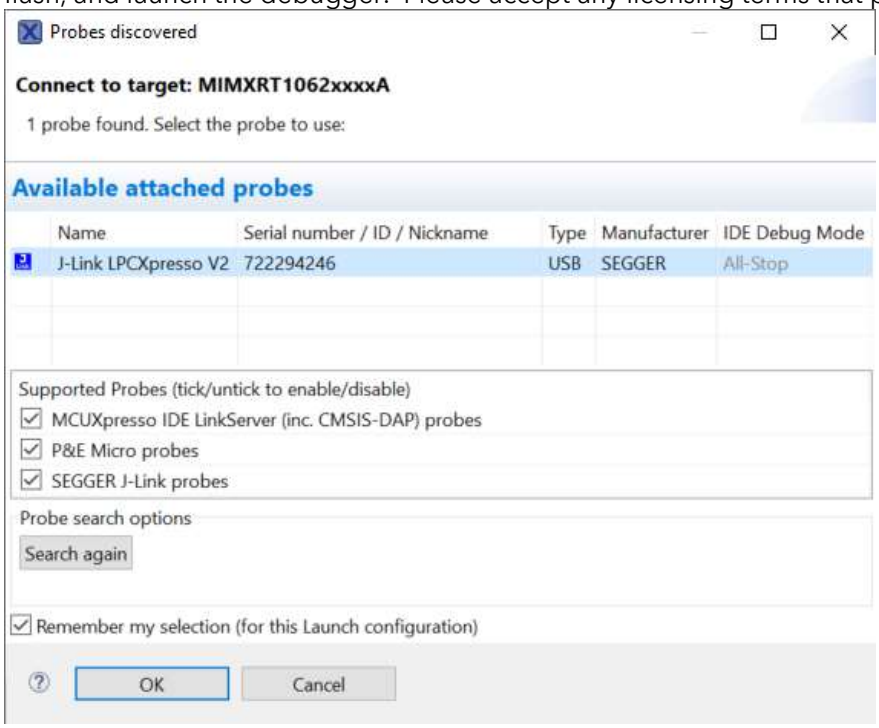
Create a Debug Configuration

In Eclipse, a Debug Configuration is stored with a project, and includes the settings needed to flash and debug the application on the hardware. For Zephyr, we will use the IDE to generate this configuration, and then modify it for the Zephyr application. These lab steps are specific to using Segger Jlink debug probe. For other probes supported in MCUXpresso IDE, refer to the document installed with the IDE at \MCUXpressoIDE_11.6.0_8187\MCUXpresso_IDE_ZephyrRTOS_Debug_Guide.pdf.

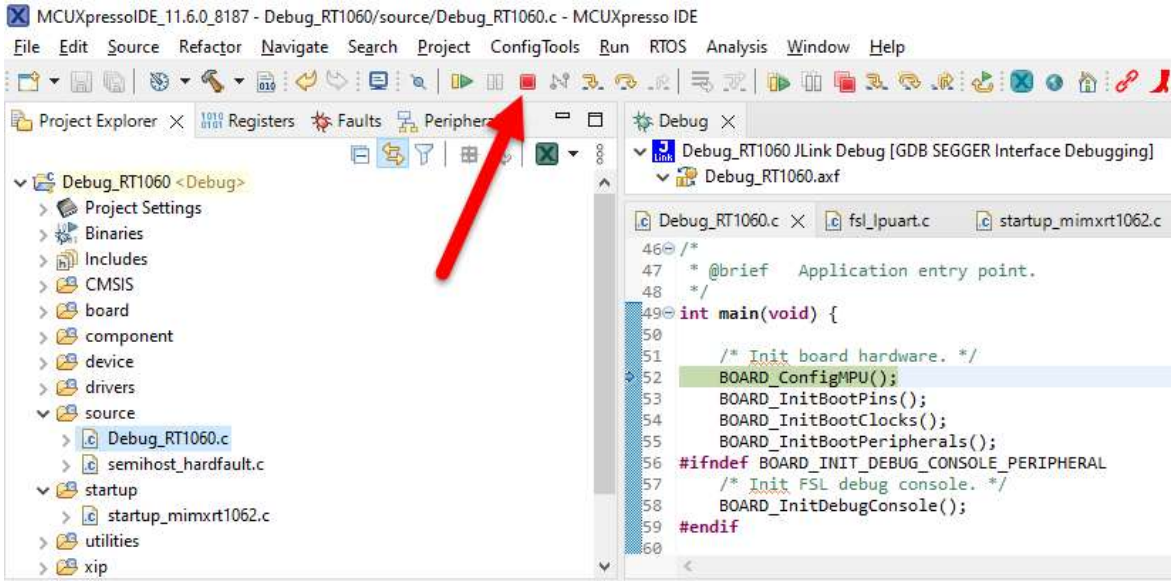
23. In the IDE, in the Quickstart Panel, click Debug



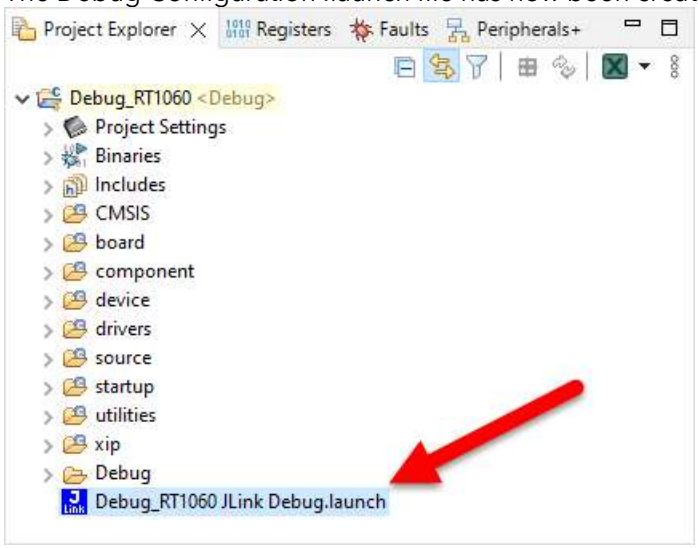
24. Select the debug probe detected by the IDE. Click OK. This will build the Debug project, program the flash, and launch the debugger. Please accept any licensing terms that pop-up.



25. The debugger will run the Debug app and halt at main(). After it halts, **Terminate** the debug connection.



26. The Debug Configuration .launch file has now been created and added to the Debug project.

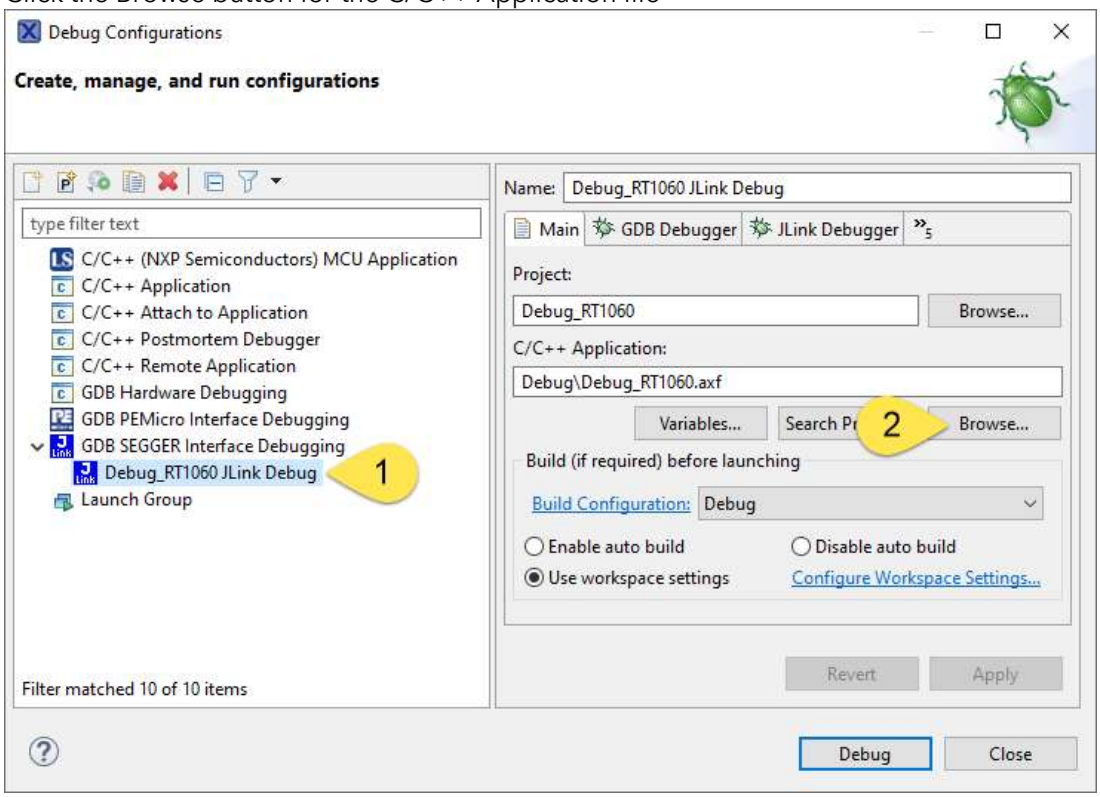


Modifying Debug Configuration for Zephyr application

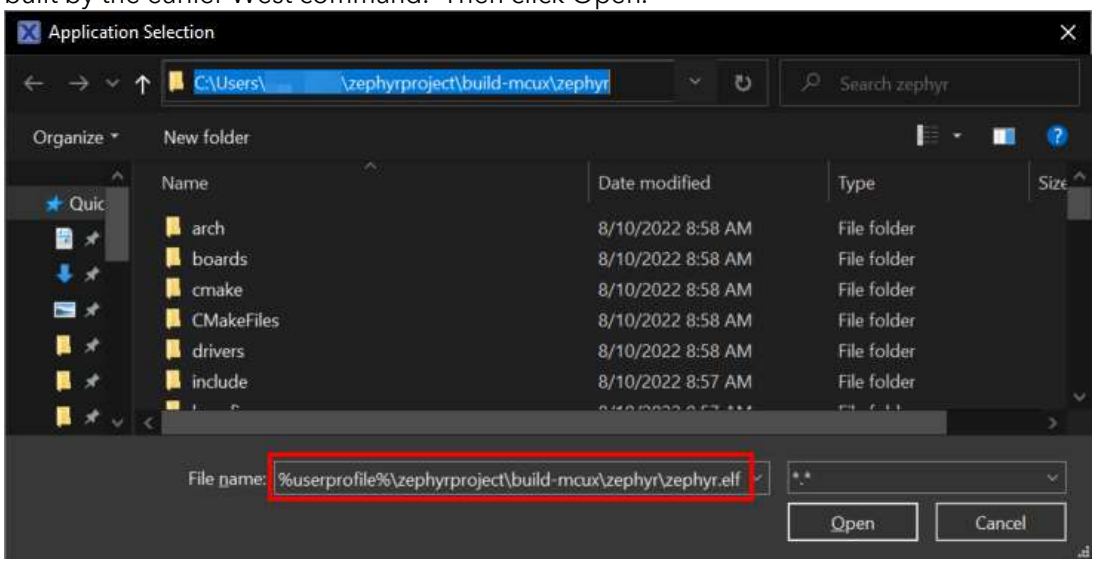
Now that the IDE created the Debug Configuration for us, we will modify it to point to the Zephyr application previously built by West, instead of the app this Debug project builds.

27. In the IDE, use the menu **Run->Debug Configurations**…

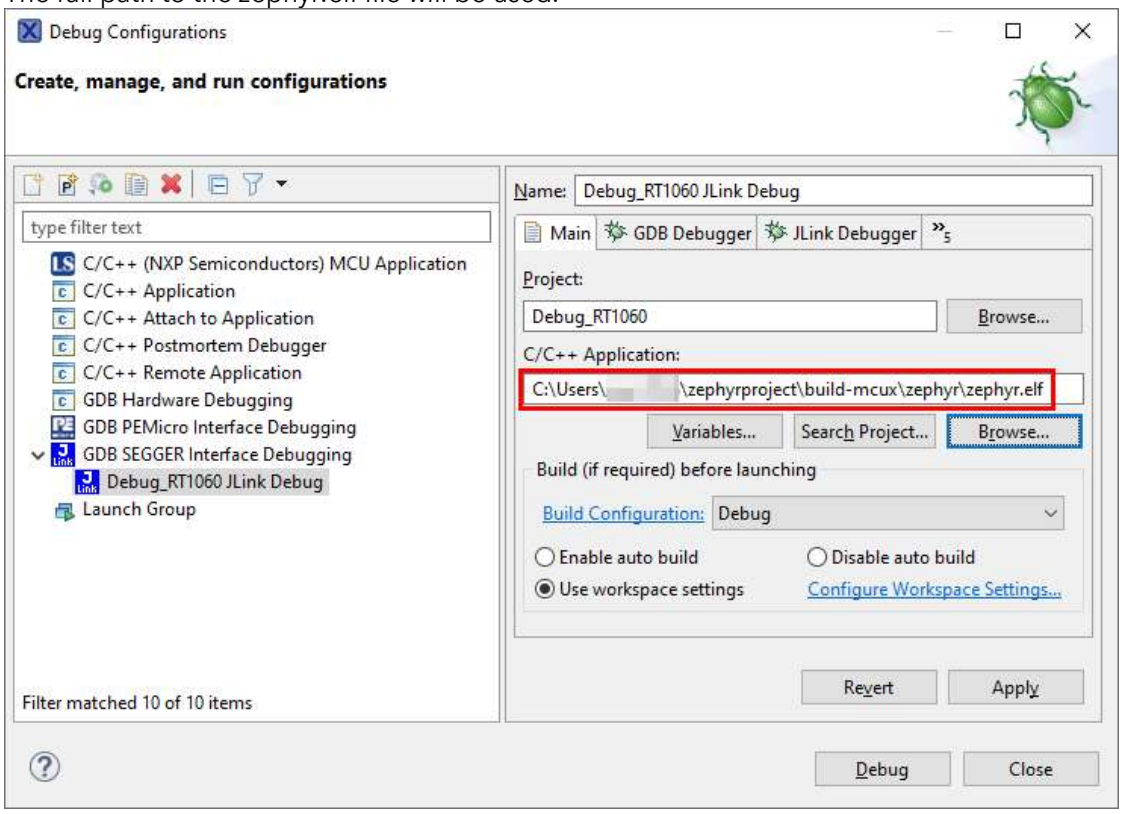
28. Modify the Debug Configuration just created:
    a. In the left window, select the Debug Configuration
    b. Click the Browse button for the C/C++ Application file



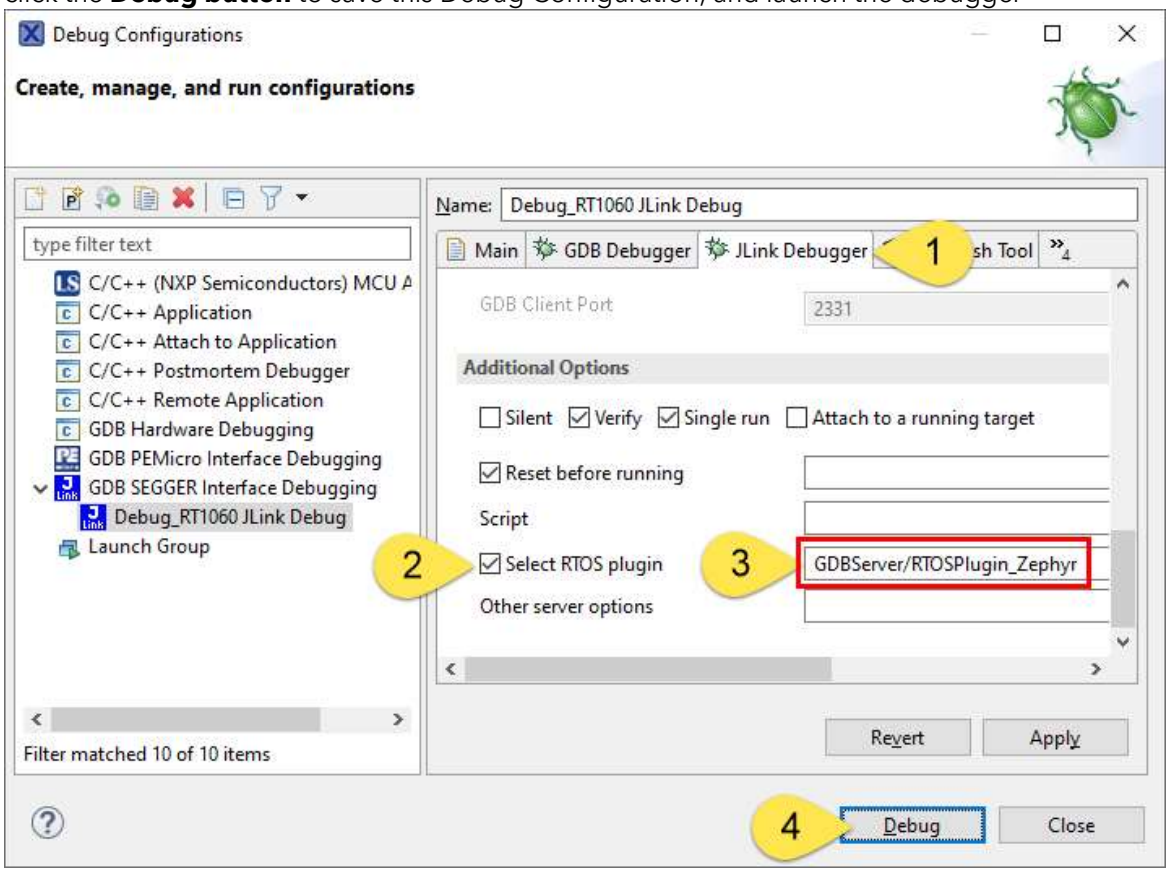
29. Paste **%userprofile%\zephyrproject\build-mcux\zephyr\zephyr.elf** , or browse to the zephyr.elf file built by the earlier West command. Then click Open.



The full path to the zephyr.elf file will be used.



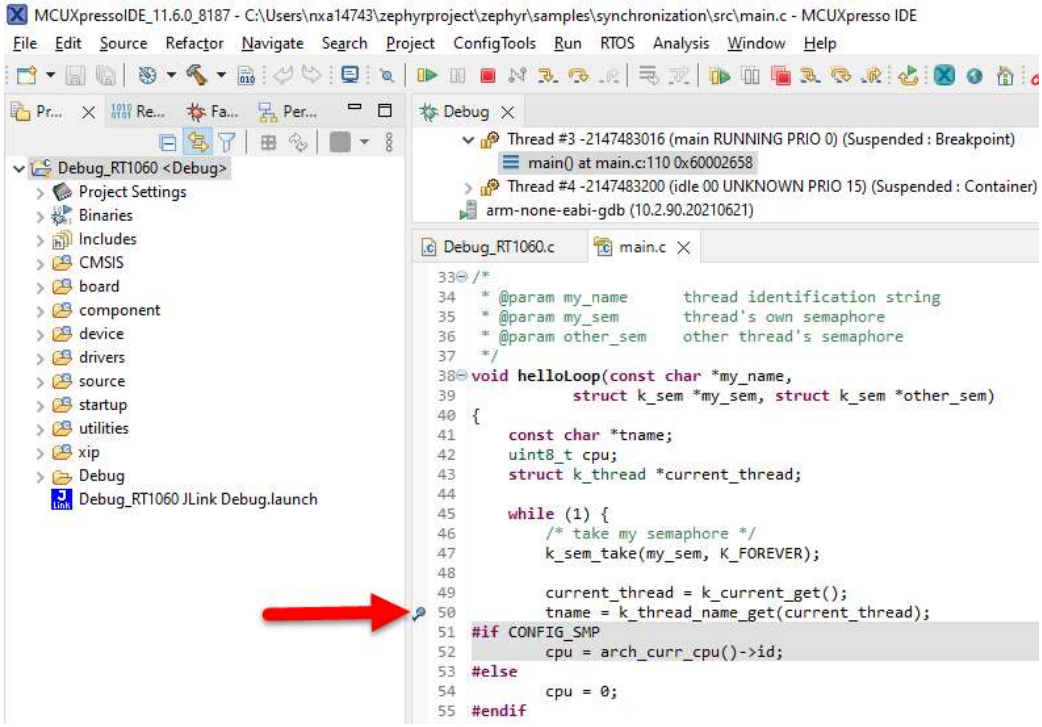
30. Configure the Zephyr RTOS plugin for Jlink:
    c. Click the tab **JLink Debugger**
    d. Check the box for **Select RTOS plugin**, this box is under Additional Options, you may need to scroll down to the bottom to find it.
    e. Paste in **GDBServer/RTOSPlugin_Zephyr**
    f. click the **Debug button** to save this Debug Configuration, and launch the debugger



Debugging in MCUXpresso IDE with Thread Awareness

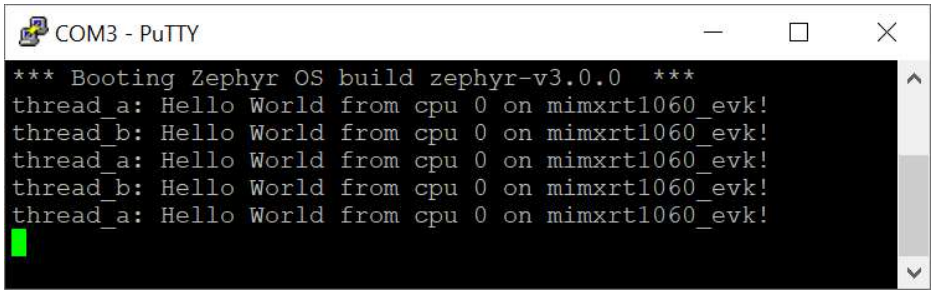
31. The next step will add a breakpoint in main.c. With the debugger halting in main(), typically main.c should already be open in the editor for this lab. The debugger will open source files as needed using the zephyr.elf symbolic file. But since this Debug project is not otherwise linked to the Zephyr sample project, the Project Explorer View will not list the source files for the application. To browse the source files in the Zephyr application, see the other lab guide Zephyr_MCUXpresso_IDE_Editor_Lab.pdf. Or use the menu File->Open, and browse to the desired file in the Zephyr tree.

32. **Add a breakpoint** in main.c at line 50. Do this by double-clicking the empty column to the left of the #50 line number. This will halt the application in the loop of helloLoop().

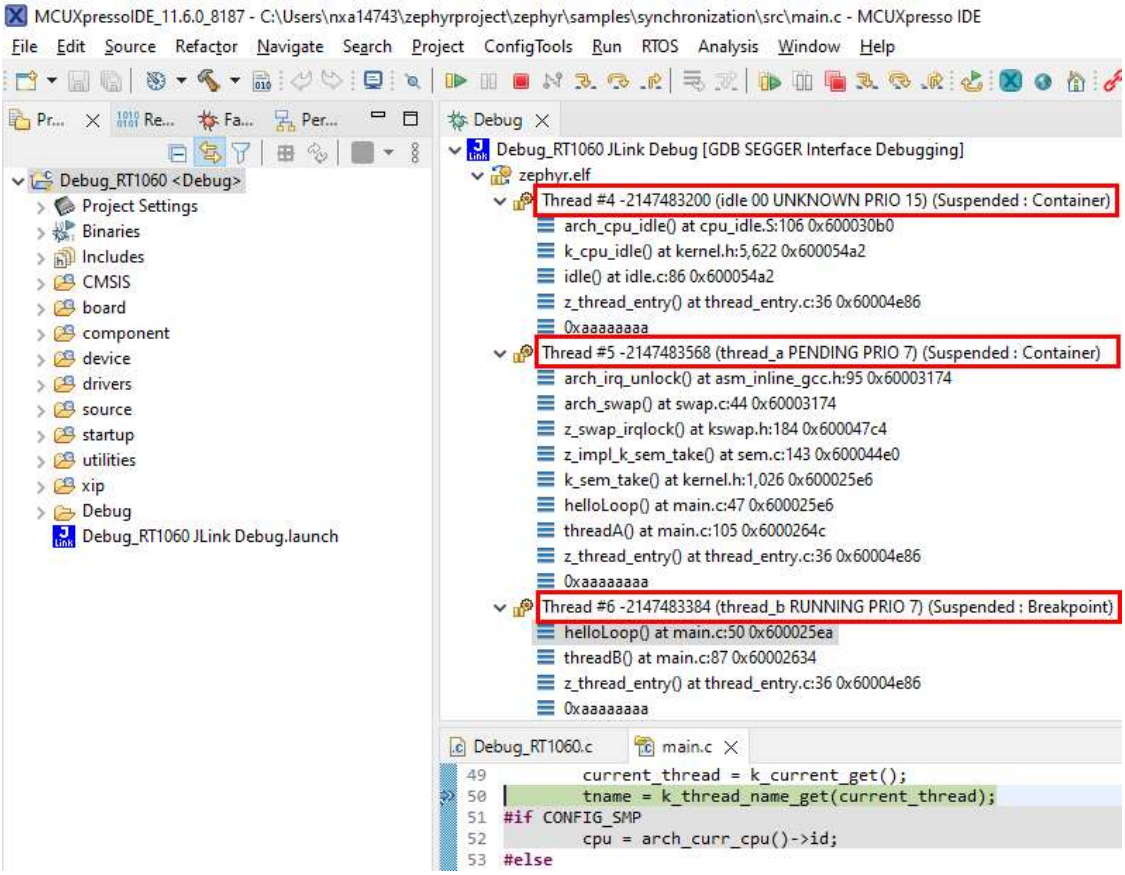


33. Click the **Resume button** (F8) to continue the program. It will halt at the breakpoint

34. Click the **Resume button a few times** to see the synchronization sample application print to the terminal



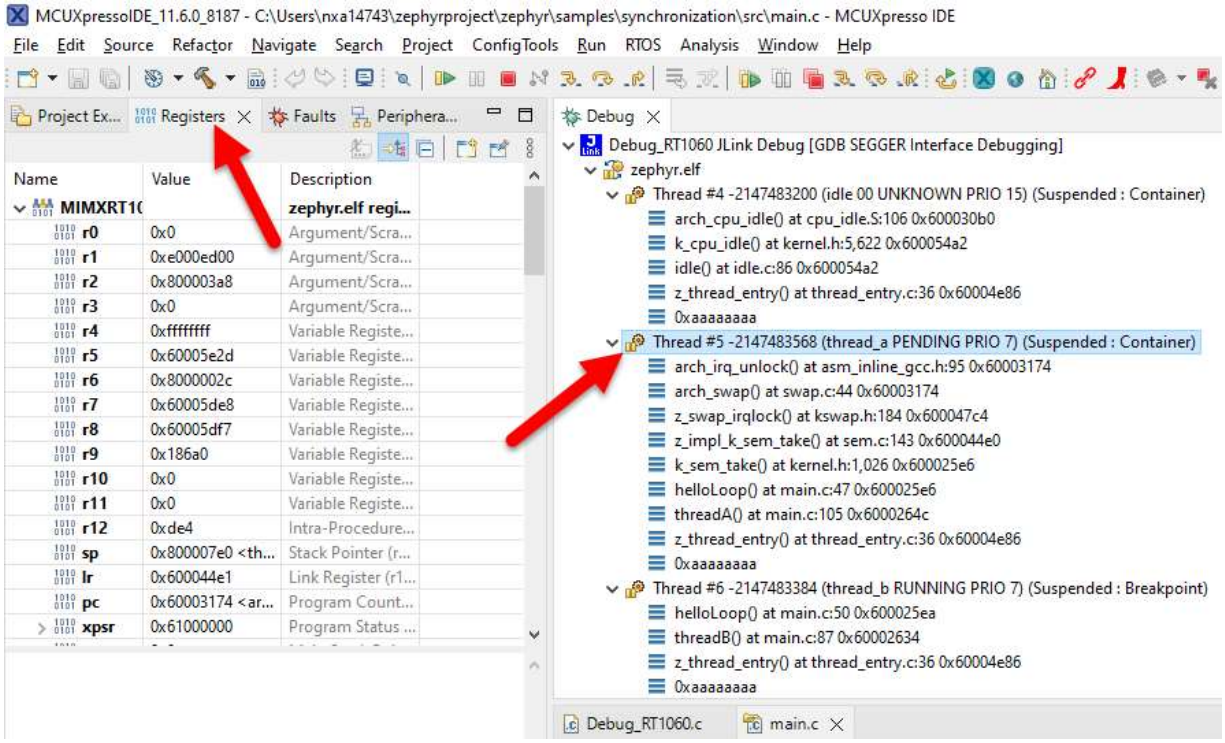
35. After the debugger halts at the breakpoint, notice the application threads and their details are displayed in the Debug View:



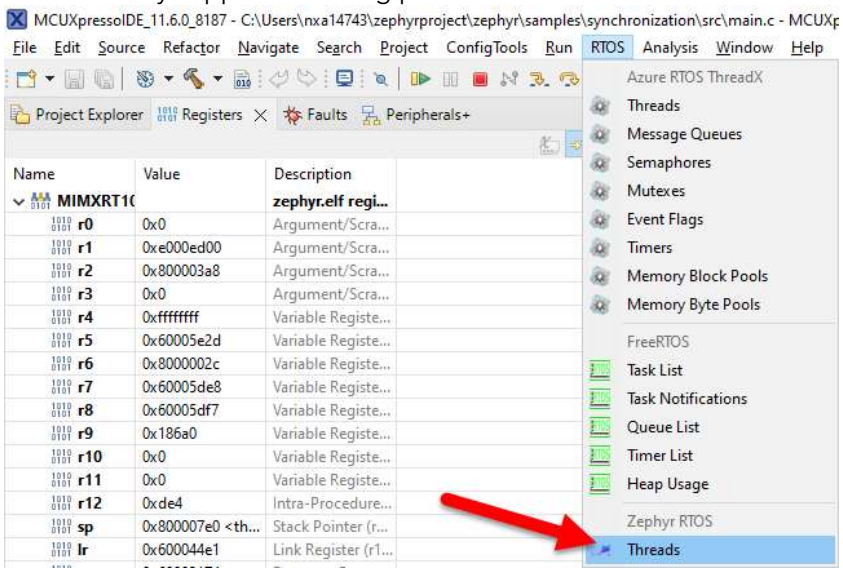
Thread Awareness details shown in the Debug View:

- Name of thread, i.e. thread_b
- Current state, PENDING, RUNNING, etc.
- Task Priority
- Call stack for each thread

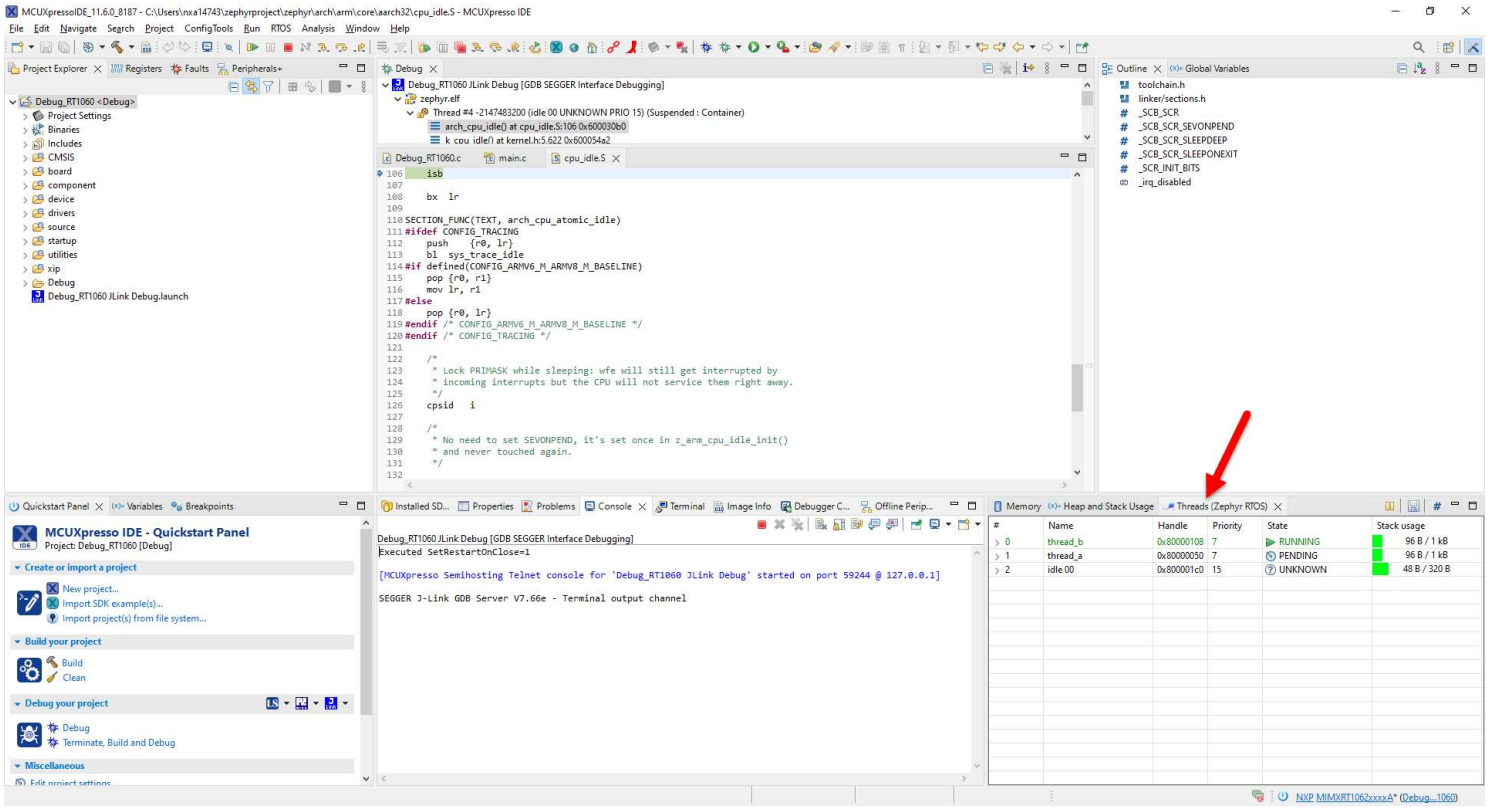
36. By selecting a thread and then selecting the Registers view, one can see the CPU registers, stack pointer, and program counter of each thread context. It also labels each thread as running or suspended.



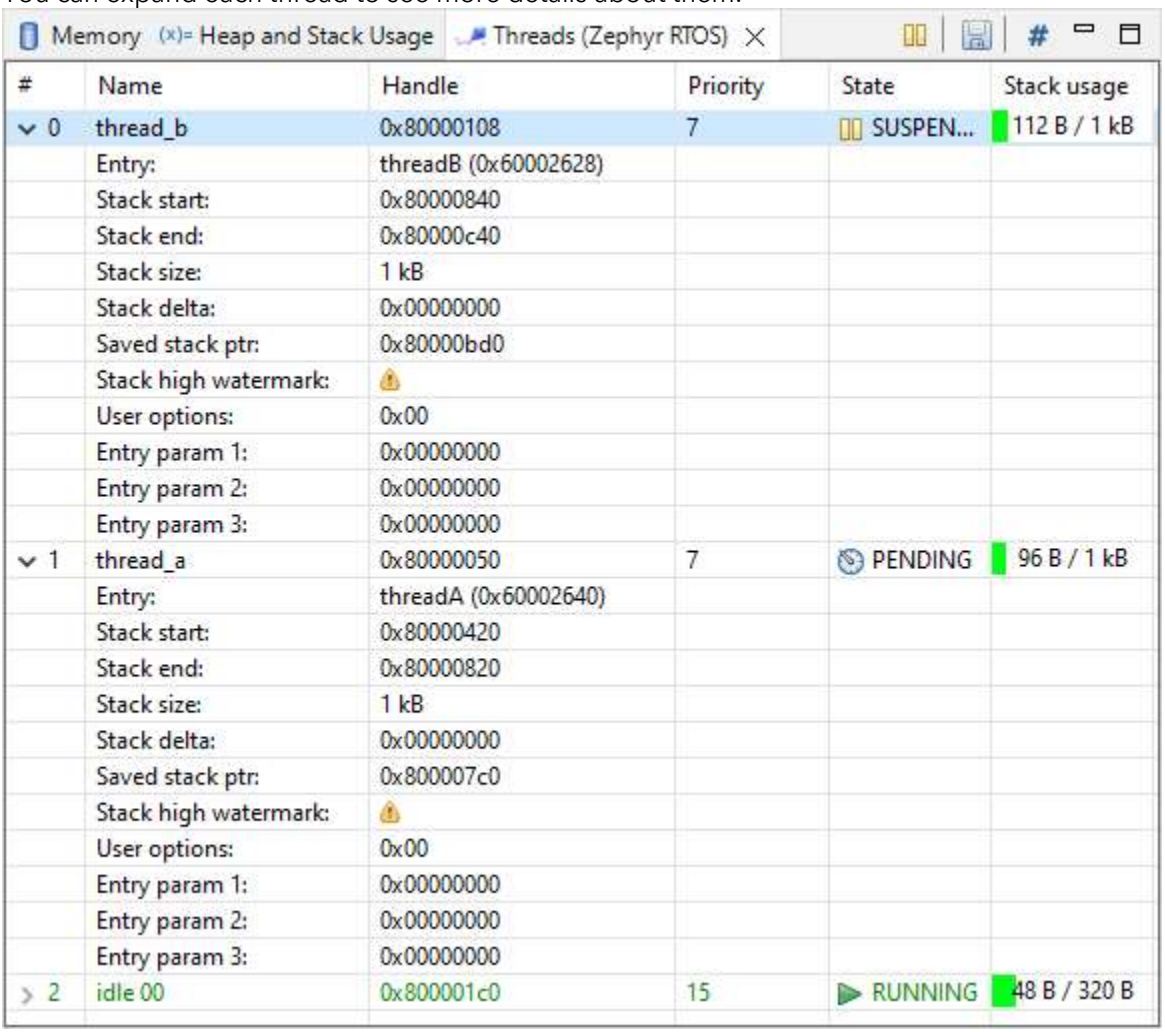
37. Open the Thread Aware Debug (TAD) View for Zephyr.
Use the menu **RTOS->Zephyr RTOS->Threads**. The TAD View is completely decoupled from the GDB Thread awareness support (built in the GDB server). This View specific to MCUXpresso IDE can be used with any supported debug probe in the IDE.



38. This adds the Threads (Zephyr RTOS) View to the Perspective:

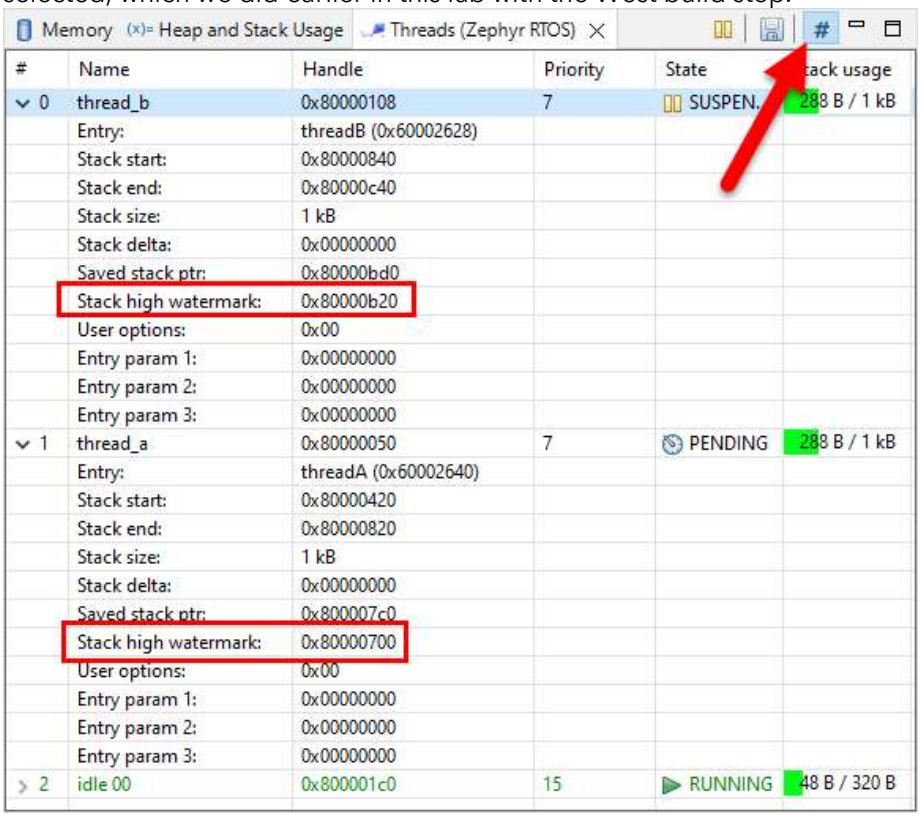


39. You can expand each thread to see more details about them:



#	Name	Handle	Priority	State	Stack usage
0	thread_b	0x80000108	7	SUSPEN...	112 B / 1 kB
	Entry:	threadB (0x60002628)			
	Stack start:	0x80000840			
	Stack end:	0x80000c40			
	Stack size:	1 kB			
	Stack delta:	0x00000000			
	Saved stack ptr:	0x80000bd0			
	Stack high watermark:				
	User options:	0x00			
	Entry param 1:	0x00000000			
	Entry param 2:	0x00000000			
	Entry param 3:	0x00000000			
1	thread_a	0x80000050	7	PENDING	96 B / 1 kB
	Entry:	threadA (0x60002640)			
	Stack start:	0x80000420			
	Stack end:	0x80000820			
	Stack size:	1 kB			
	Stack delta:	0x00000000			
	Saved stack ptr:	0x800007c0			
	Stack high watermark:				
	User options:	0x00			
	Entry param 1:	0x00000000			
	Entry param 2:	0x00000000			
	Entry param 3:	0x00000000			
2	idle 00	0x800001c0	15	RUNNING	48 B / 320 B

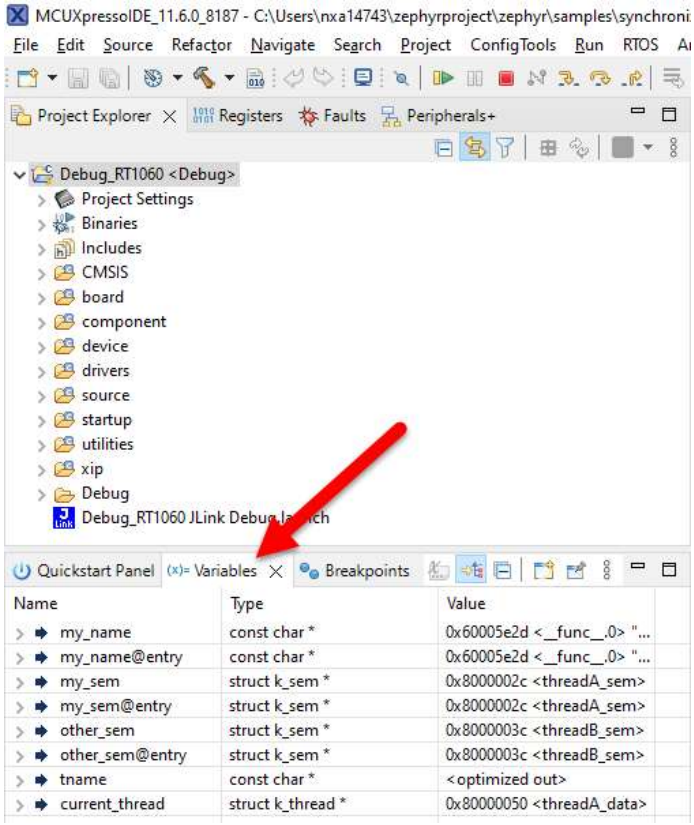
40. Notice the stack high watermark is not shown. The stack usage shown in the previous step is based on the current stack pointer for each task. Click the **# button** to enable the stack high watermark. Now the high watermark is detected, and the stack usage numbers increase because they are based on high watermark. This high watermark feature requires the Kconfig symbol CONFIG_INIT_STACKS is selected, which we did earlier in this lab with the West build step.



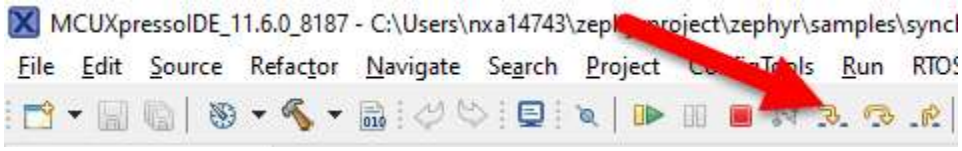
The Threads (Zephyr RTOS) View of the IDE provides the following thread details:

- **#** - Thread Control Block index
- **Name** - Name of the thread
- **Handle** - Address of the thread handle
- **Priority** - Priority of the thread
- **State** - Current task state (e.g. running, suspended, queued, etc.)
- **Stack Usage** - Graphical view of current stack usage, with current allocation and stack size available to the thread
- **Entry** - Entry function for current thread
- **Stack start** - Stack start address
- **Stack end** - Stack end address
- **Stack size** - Size of stack
- **Saved stack ptr** - Thread's saved stack pointer during the last context switch
- **Stack high watermark** - Highest address used by stack
- **User options** - User facing 'thread options'. Values defined in kernel.h
- **Entry param 1/2/3** - The entry point parameters

41. The remaining debugging steps are to help get familiar with MCUXpresso IDE, and to compare with debugging command line with GDB. Click the **Variables tab** to see the local variables. The debugger needs to be Suspended to view these variables. If this view is empty, click the Resume button again, and the IDE will update this view when it halts at the breakpoint.

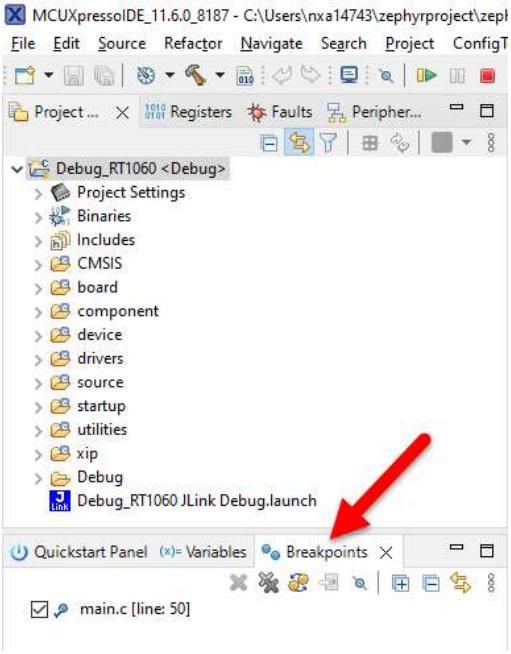


42. **Step Into** the next line

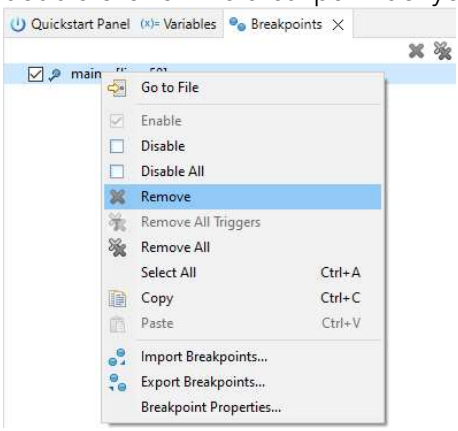


43. **Step Over** the next line

44. Click the **Breakpoints View**.



45. Remove the breakpoint at line 50. Right-Click the breakpoint, and select **Remove**. Or you can double-click on the breakpoint dot you set earlier in main.c.



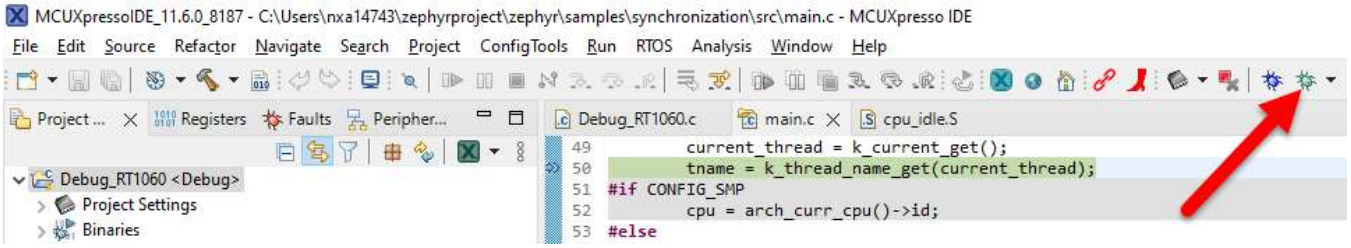
46. Click the **Resume button** to let the application run without breakpoints. The app will continue to print to the terminal.

47. **Terminate** the debug connection

48. Disconnect the Terminal

This Debug project can be easily used to debug other Zephyr applications on the same board. Just modify the Debug Configuration to load the zephyr.elf file of another app. Or you can create multiple Debug Configurations in this project, that each debug different apps.

Note: Now that the Debug Configuration has been created, you can quickly relaunch that Configuration using the Green Bug icon on the toolbar.



This completes this lab.

Revision History

Rev	Date	Details
1.0	5/29/2022	Initial Version
2.0	8/15/2022	Updated to have IDE create dummy project and Debug Configuration, new Thread Awareness Debug View for Zephyr
2.1	8/18/2022	Renamed this IDE project as Debug_RT1060, or Debug project, Added reference to new guide Zephyr_MCUXpresso_IDE_Editor_Lab, Add steps to use Terminal in Eclipse