Objectives

In this lab, you will learn:

- How to build and debug a Zephyr application in VS Code
- How to use various debugging tools and controls
- How to enable Zephyr thread awareness using Kconfig symbols
- How to use view thread details in the debugger
- How to enable and use the Peripheral view

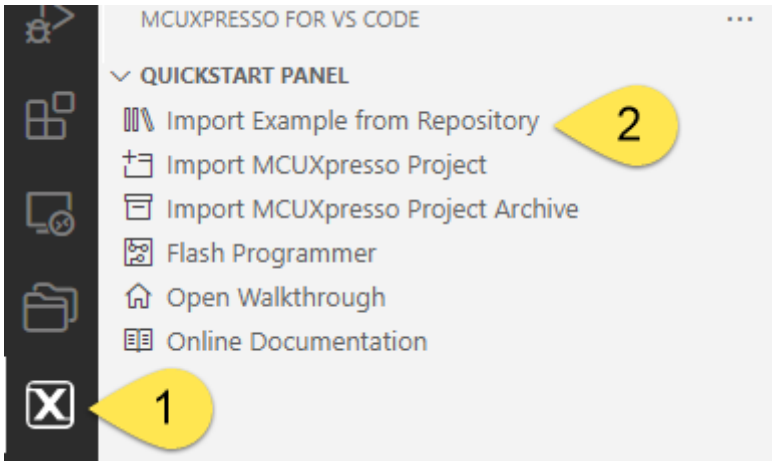
Debug Lab

In this lab, we will go through detailed steps to run the Zephyr *Synchronization* sample application. The Debug lab includes steps to build/debug the application and details on debug tools/controls in VS Code. It also includes steps to enable thread awareness by setting up Kconfig symbols for the application example. We will take a closer look at key features like Thread status, Stack usage, Call Stack, Thread structure, and more.

1. Import *Synchronization* application example

Follow the steps to import the example, similar to the steps in the [[Zephyr Lab MCXN947 Hello World]].

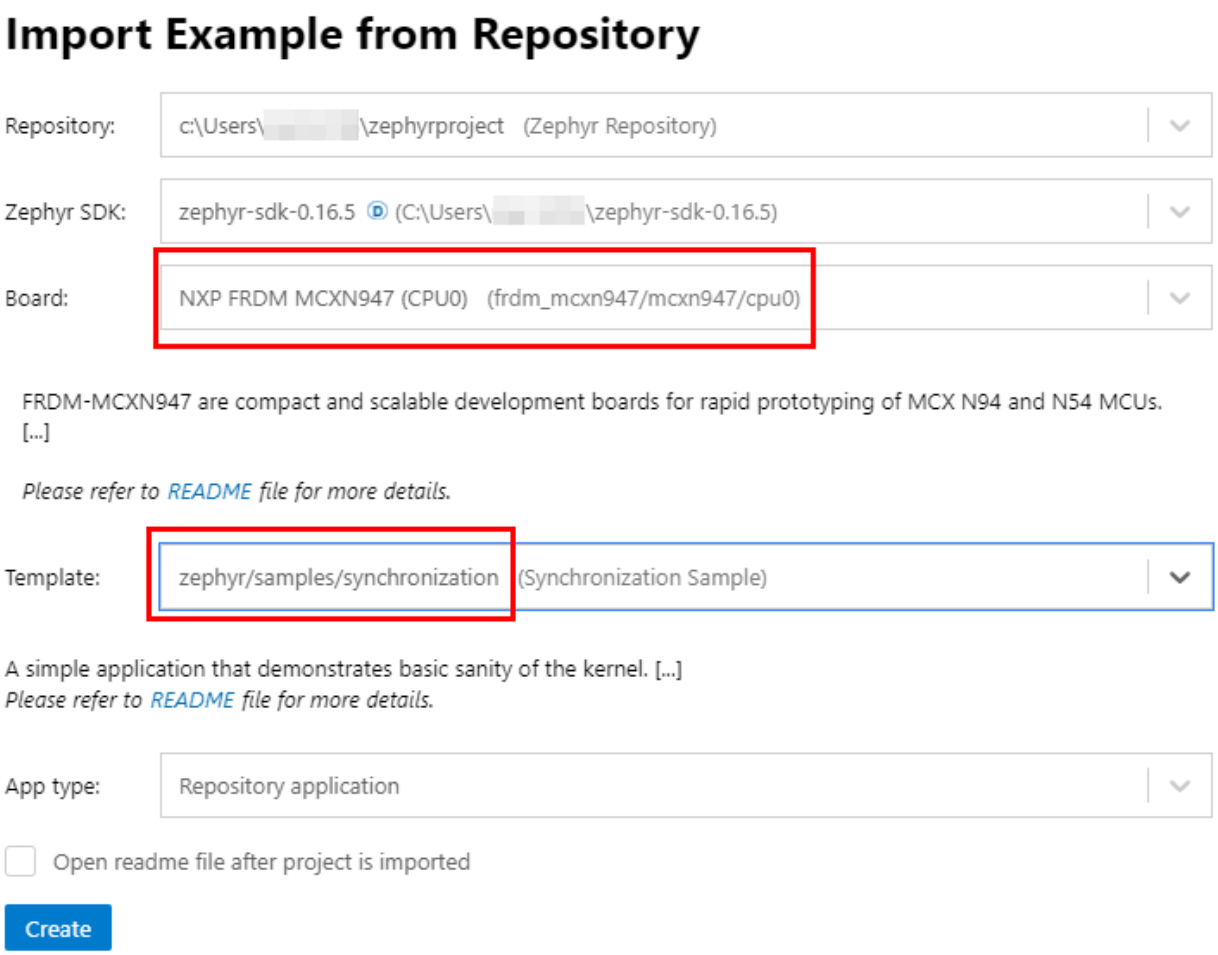
1. To import the *synchronization* application from the Zephyr Repository, click **Import example from Repository** in the Quickstart Panel.



2. Select the following board settings to import the example:

    - For the board, you can type **n947** to filter the board target **frdm_mcxn947/mcxn947/cpu0**. Be sure this board target ends with `cpu0`.
    - For the example template, type **sync** to filter the app **zephyr/samples/synchronization**
    - For Application type, select **Repository Application**

- Click **Create**

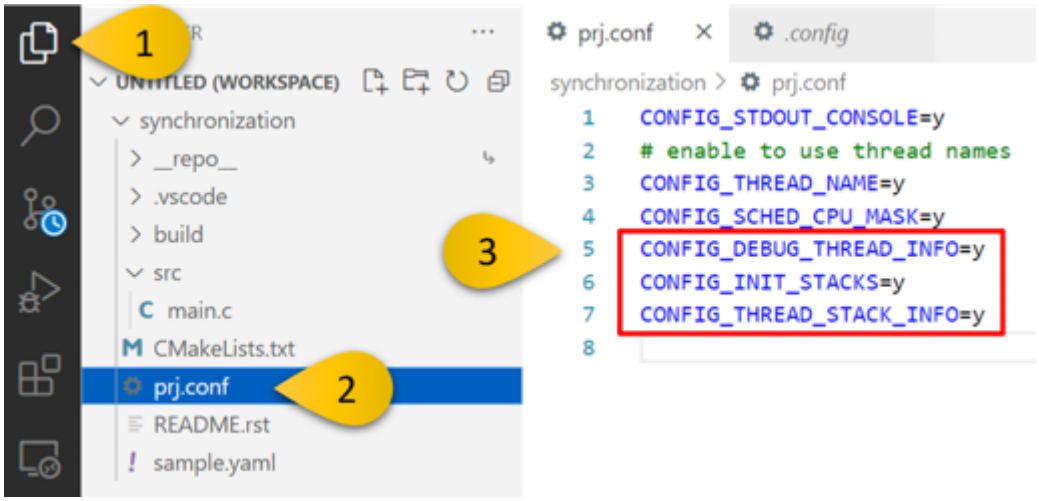


2. Enable thread awareness and build the example

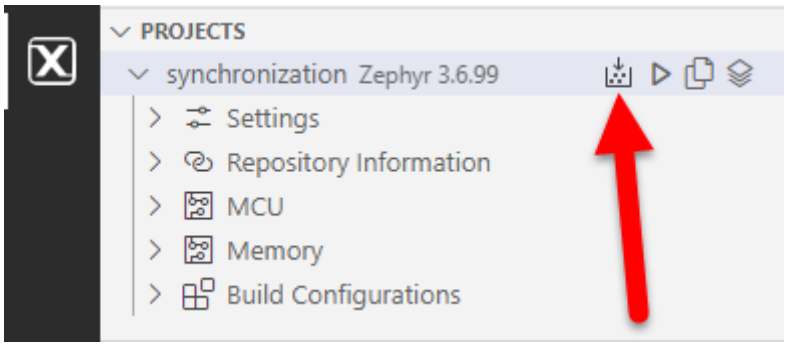
To enable thread awareness for debugging, we need to add specific Kconfig symbols to the application example. These steps will modify the *prj.conf* file and verify the generated *.config* file, similar to the [[Zephyr Lab MCXN947 Kconfig]] guide.

3. Add the following Kconfig symbols to the **prj.conf** file and save the file (Ctrl+S):

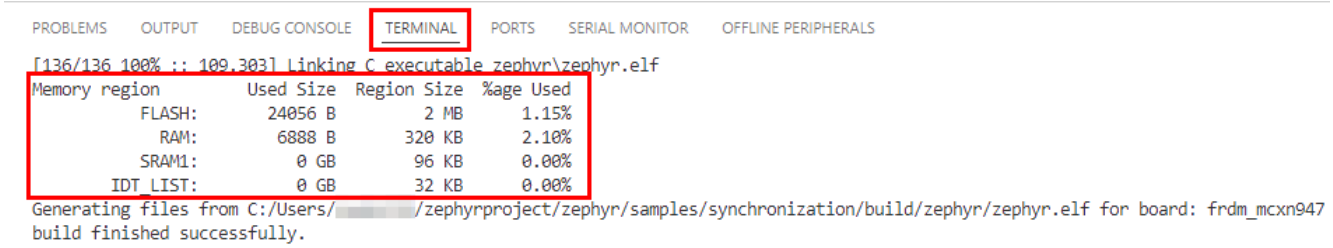
```
CONFIG_DEBUG_THREAD_INFO=y
CONFIG_INIT_STACKS=y
CONFIG_THREAD_STACK_INFO=y
```



4. Build the project by clicking the **Build Selected** icon.



After the build, the Terminal window displays the memory usage (or compiler errors if any).



To enable the Peripheral view in VS Code for debugging, we need to edit the debugger *launch.json* file, and add the `svdPath` pointing to the SVD file. The SVD files are located in a repository included with the MCUXpresso SDK. Please refer to the [[Zephyr Lab Installation and Preparation]] guide and follow the steps to import the MCUXpresso SDK repository.

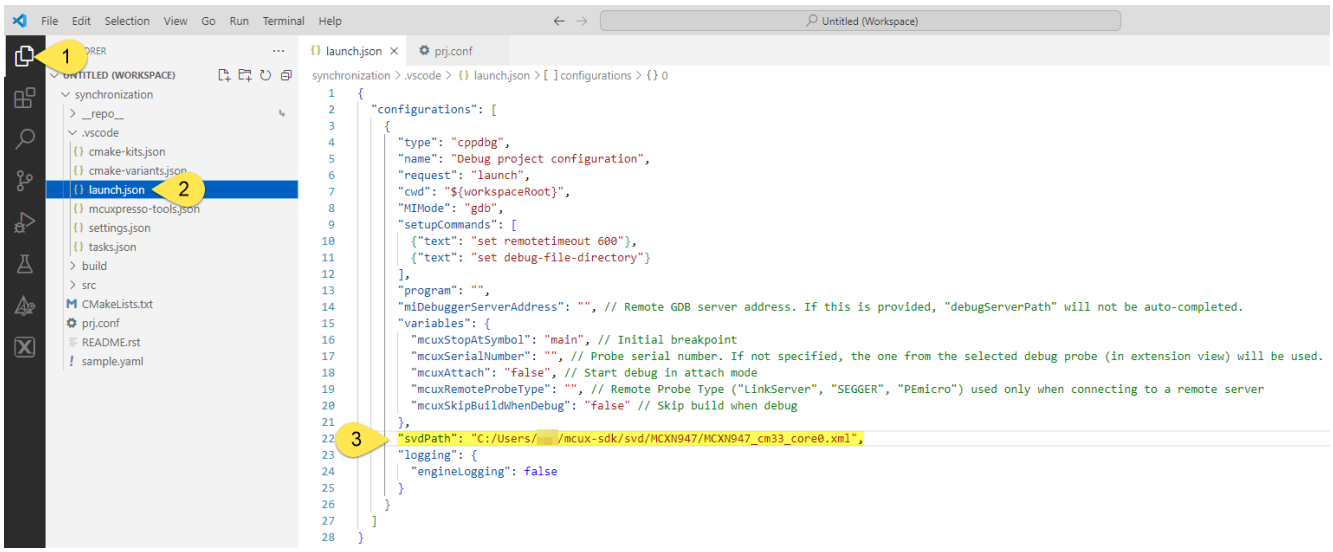
5. Switch to the **Explorer view** (Ctrl+Shift+E) and open the file

   synchronization > .vscode > launch.json

   Add this line to the **launch.json** file. Note, be sure to change the path to your SVD file, and save the file (Ctrl+S):

   > Check if this SVD/XML file exists in the path below. If not, see the lab addendum 3_Zephyr-Lab-MCXN947-Debug-SVD-file.txt.

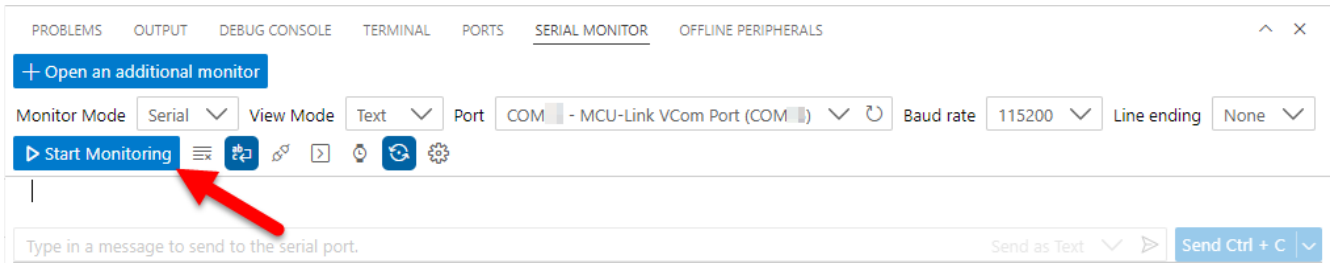
```
"svdPath": "C:/Users/NXP/mcux-sdk/svd/MCXN947/MCXN947_cm33_core0.xml",
```



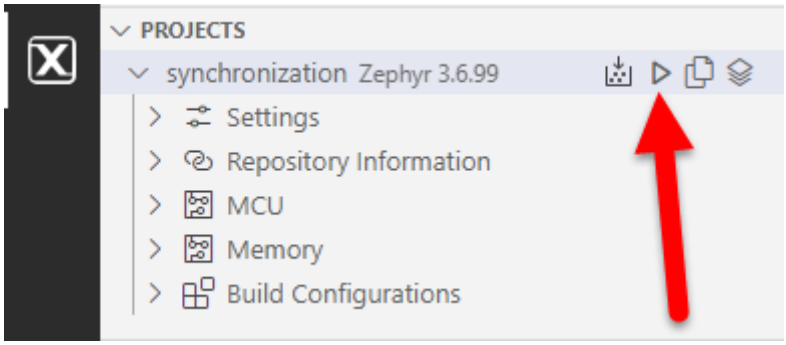
3. Exploring Debug Tools and Controls

After building the application and enabling thread awareness, let's explore the debug tools and controls in the Run and Debug view.

6. If not already connected to the board, connect the USB Type-C cable to J17, to power and debug the board. Connect the Serial monitor to the board's COM port and click **Start Monitoring**.



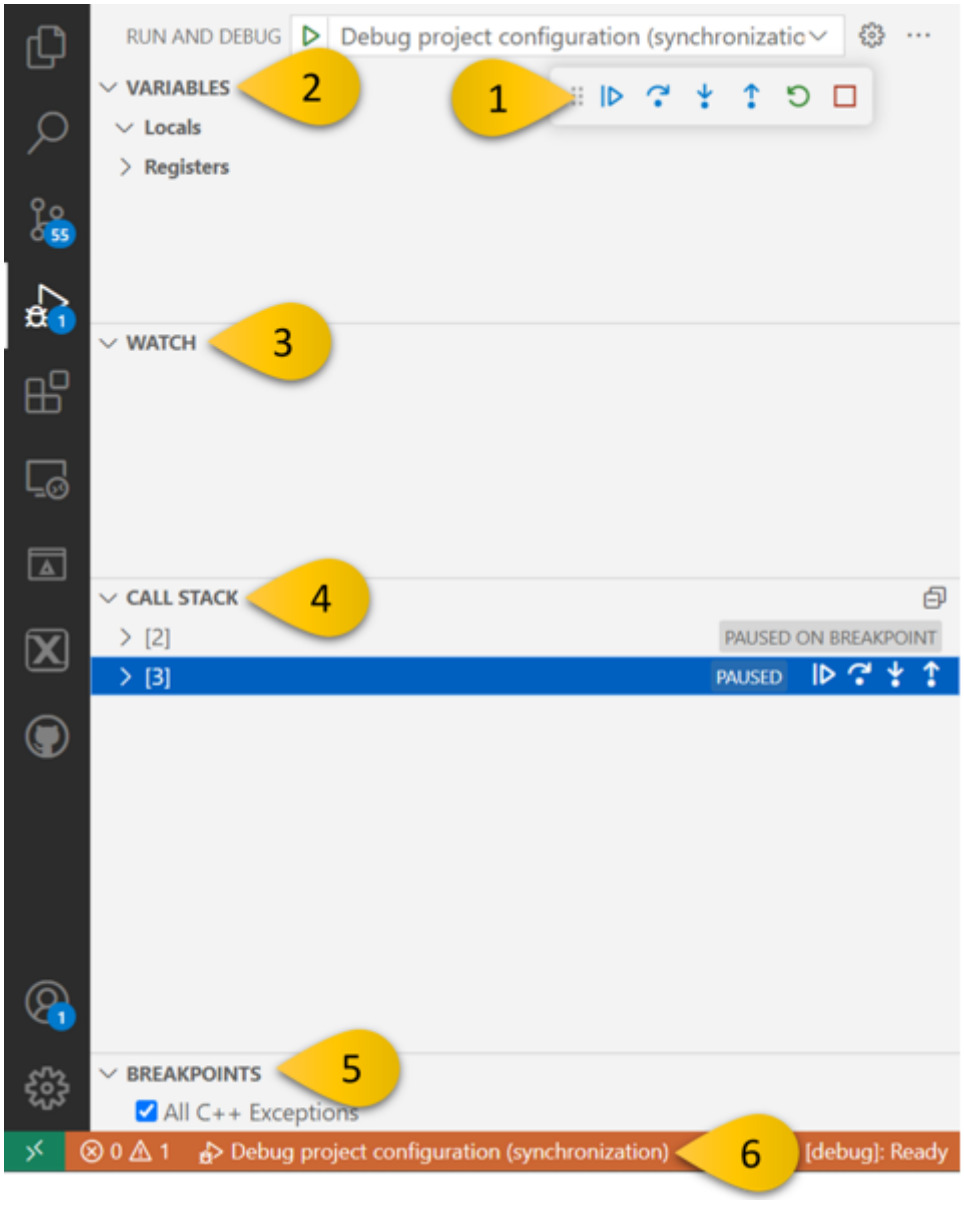
7. Click the play icon to **Debug** the Synchronization application.



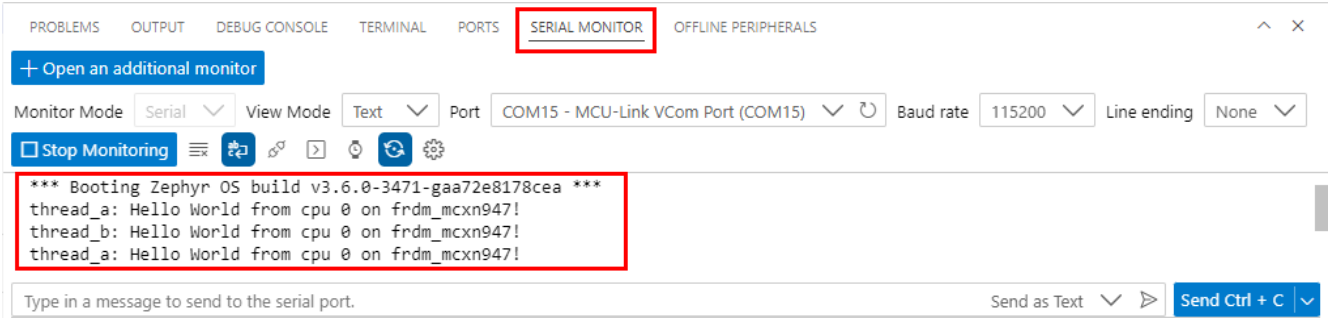
8. The following debug tools/controls are enabled in the Run and Debug view:

    1. **Debug controls**: Resume/Pause, Step over, Step into, Restart, Stop

2. **Variables**: Locals and Registers
3. **Watch**: Add expressions to monitor
4. **Call Stack**: List of active subroutines in the program
5. **Breakpoints**: Add, toggle, and view breakpoints in the code
6. **Status ribbon**: Ribbon color changes to orange



9. To continue with the debug execution click **Resume**. The application prints hello world (both threads) in the Serial Monitor view.

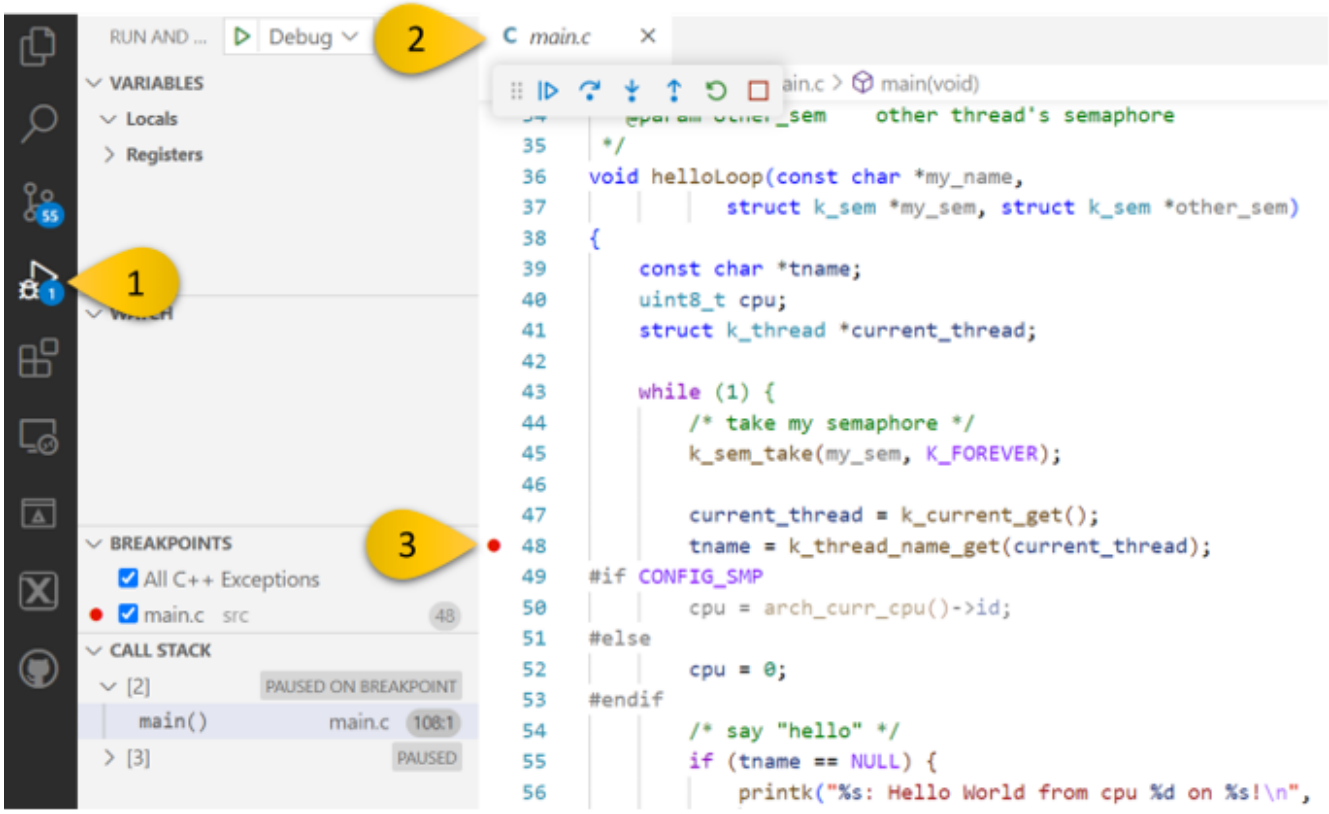


4. Debugging with Thread awareness – RTOS Viewer

The synchronization application has two threads (threadA and threadB) taking turns printing a message to the console. The example uses semaphores to synchronize the two tasks.

10. While the debug session is active, go to the **Run and Debug** view, and open the source file with the main() subroutine in the editor.

    Add a breakpoint by double-clicking on the left side of line 48. A red dot next to the line and an entry in the Breakpoints window confirms the addition of the breakpoint.

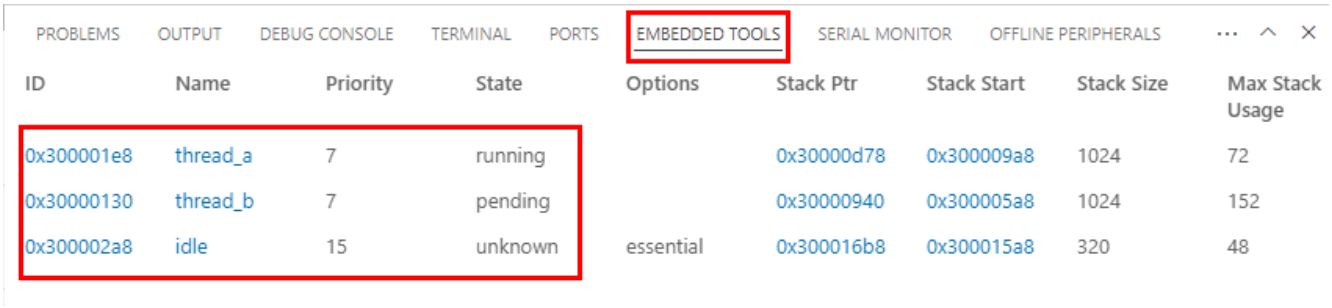


11. The debug session will halt at the helloLoop(), and the debug controls can be used to resume or step through the code. The synchronization application will print to the terminal after clicking **Resume** a couple of times.

    When the debugger halts at the breakpoint, the application threads and details are displayed in the Embedded Tools window. Click the **Embedded Tools** tab to see the thread details. These details include the

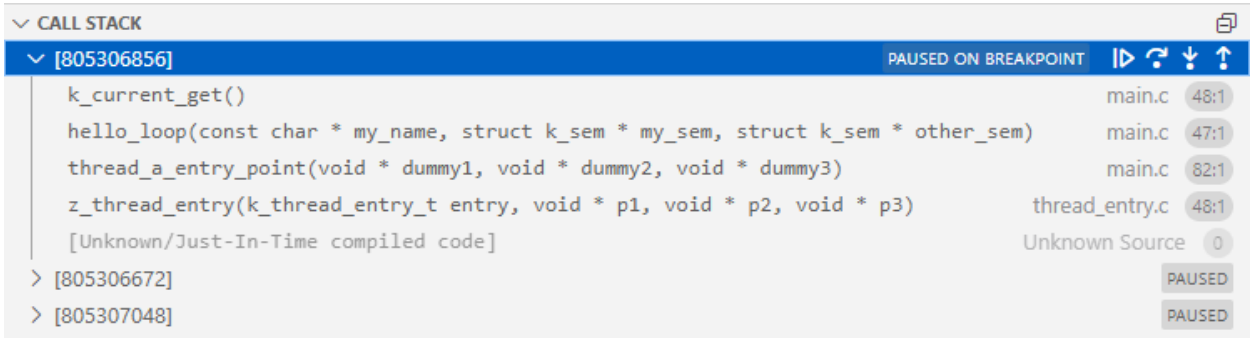
status of each thread along with its priorities, stack pointer, and stack usage.

Pause the debug session to notice the changes in the status of each thread.

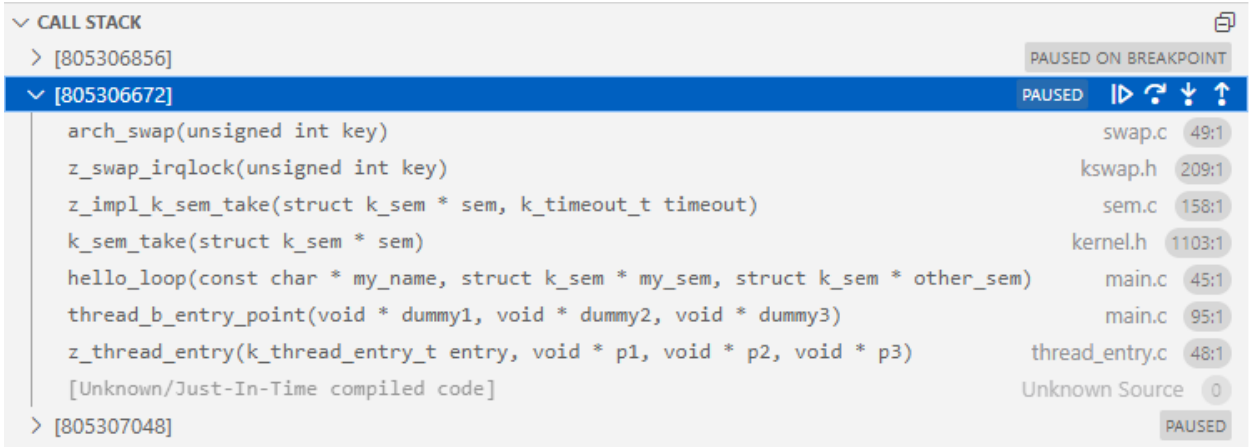


12. Along with the Embedded Tools view, the **Call Stack** window provides more information about the threads in the application:

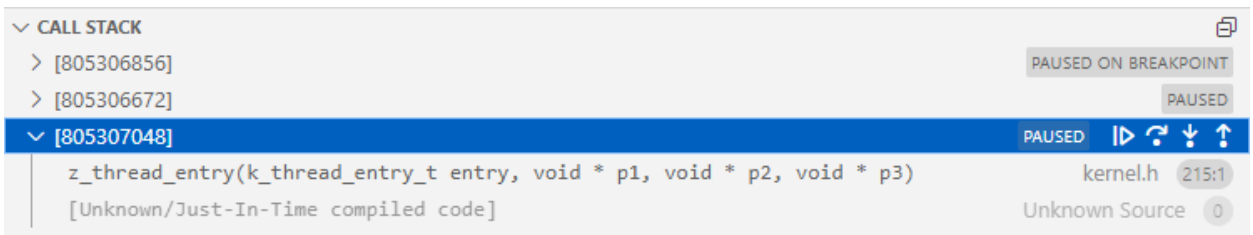
    - Running thread:



    - Pending thread:

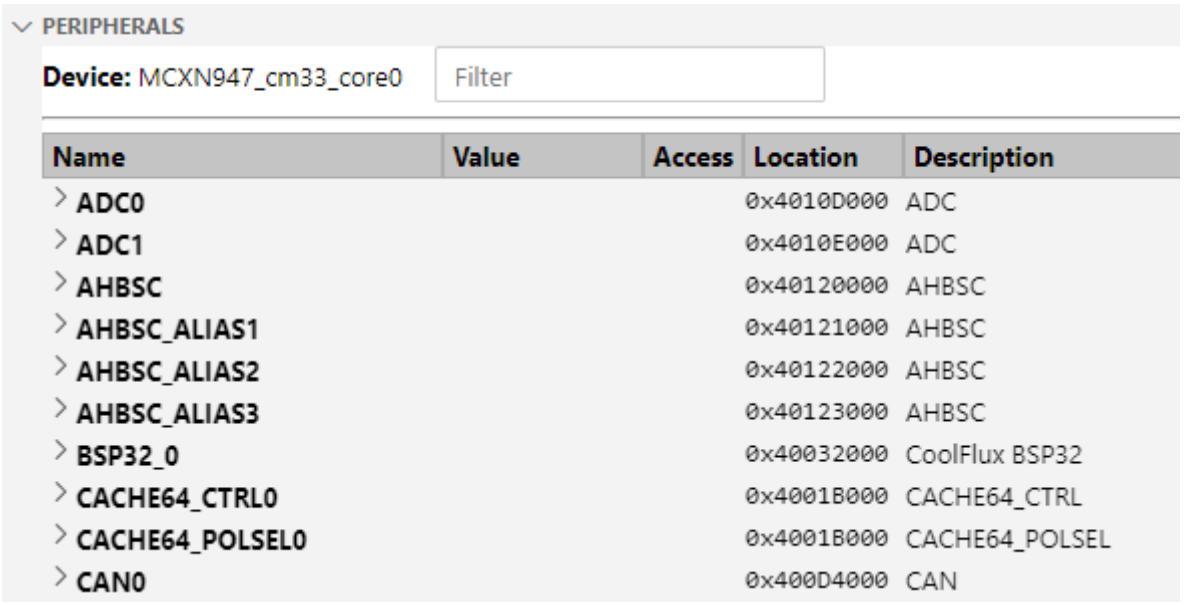


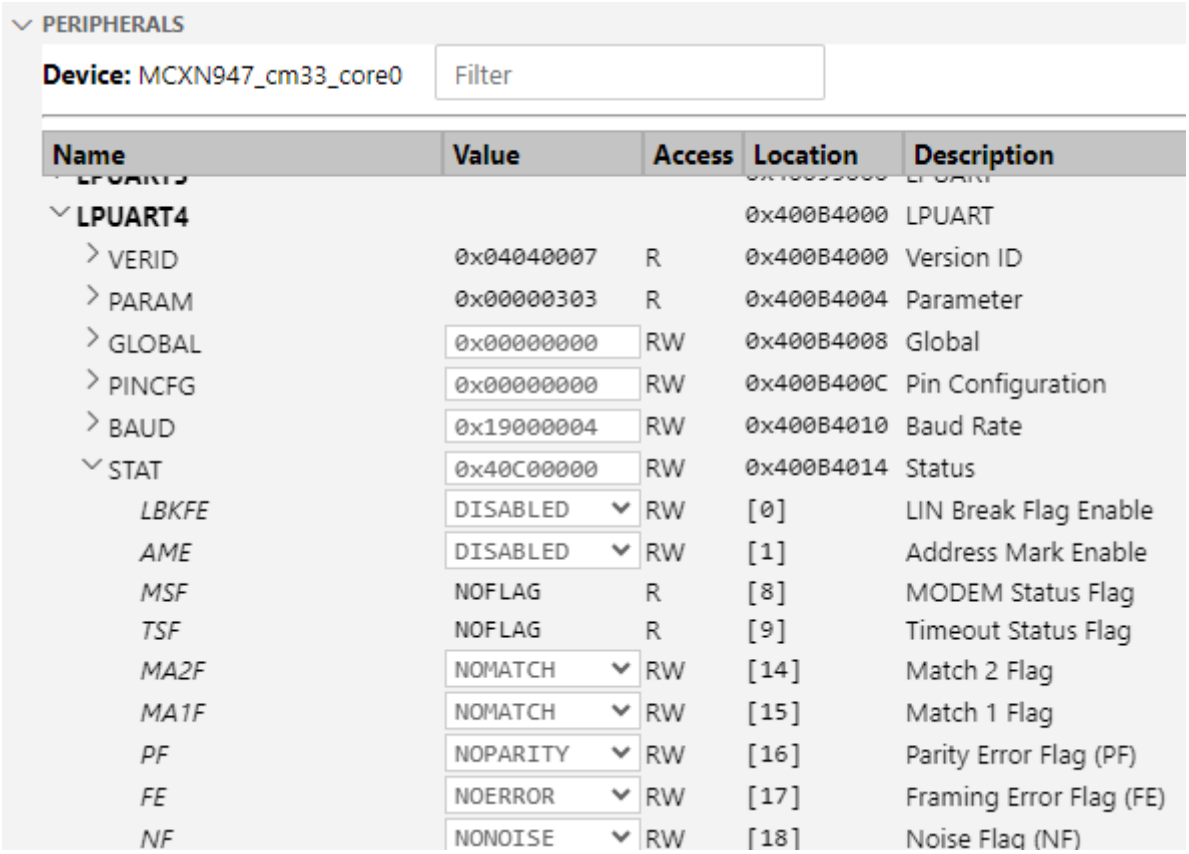
    - Idle Thread:



13. The **Watch** window and the **Variables** window can be used to add local variables and CPU registers that need to be monitored.

14. The Peripheral View will be added to the Debug view when the debugger is connected and the `svdPath` is added to the *launch.json* file. Registers and bits can be viewed for the different peripherals (eg. LPUART4).





5. Clean up after lab

This lab is completed. But the following steps will clean up the VS Code workspace:

16. Click **Stop** to end the debug session.

17. Close all Editor tabs. Right-Click on a tab, and select **Close All**.

18. If this is the last lab and you are done using the board, you should disconnect the Serial Monitor. Find the Serial Monitor view, and click **Stop Monitoring**.

Additional Resources

- VS Code Debugging

- Embedded Tooling RTOS View

> Lab completed. Return to the MCXN947 Zephyr Labs Overview [[Training Zephyr Getting Started MCXN947]]