Objectives

In this lab, you will learn:

- How to use the MCUXpresso Device Tree viewer for VS Code
- How to verify devicetree settings for a Zephyr application
- How to modify devicetree source files
- How to create a devicetree board overlay file
- How to change the devicetree pin properties for GPIO
- How to change the baud rate property for a UART in the devicetree

Devicetree Lab

This lab uses the Zephyr *button* sample application, which reacts when a button on the board is pressed. The following sections will make changes to the devicetree, and use the *button* sample to confirm these changes. These devicetree changes include the polarity of the GPIO signal for the button, and the baud rate of the Console UART.

1. Devicetree Overview

A devicetree is a hierarchical data structure that is used to describe and configure hardware. Like Linux, Zephyr uses devicetree to describe the hardware platform. Unlike Linux, Zephyr's devicetree uses very little memory, is static, and set at build-time.

The devicetree is a hierarchy of nodes and sub-nodes. A node typically describes a hardware component. Nodes have properties that describe the characteristics of the node, and can be used to configure the hardware. Below is an example of a GPIO node from the FRDM-MCXN947 devicetree board file. This is the GPIO signal connected to the SW2 button on the board. The node name is `button_0`, and has a node label `user_button_2`. The properties of this node tie it to GPIO port0 pin23, and configure it as an active-low signal with a pull-up resistor enabled. In this lab, we will change the property of this pin to active-high, and confirm how this changes the behavior of the *button* sample application.

```
user_button_2: button_0 {
    label = "User SW2";
    gpios = <&gpio0 23 (GPIO_ACTIVE_LOW | GPIO_PULL_UP)>;
```

The *button* application code interacts with this button hardware using a devicetree alias named `sw0`. An alias creates another devicetree name to reference a specific node. For example, the FRDM-MCXN947 devicetree board file defines the `sw0` alias below, which points to the node label `user_button_2`. Aliases are convenient for hardware abstraction, and Zephyr sample applications use aliases frequently. For example, the *button* sample

application runs on dozens of boards supported by Zephyr. The requirements of this button sample to run on a board are:

1. the board must support the GPIO driver, and
2. the board's devicetree must have an alias `sw0` that points to the GPIO pin node of the button.

The board's devicetree is then free to map the `sw0` alias to any GPIO pin.

```
aliases{
    led0 = &red_led;
    led1 = &green_led;
    led2 = &blue_led;
    sw0 = &user_button_2;
    sw1 = &user_button_3;
};
```

Devicetrees can also include chosen nodes. Chosen nodes are similar to aliases, because they point software to specific devicetree nodes for hardware abstraction. For example, most Zephyr applications use Zephyr's Console subsystem, including this *button* sample application. This Console uses the chosen node `zephyr,console` to know which hardware interface should be used. Below are some chosen node examples from the FRDM-MCXN947 board devicetree file. This board uses the `flexcomm4_lpuart4` peripheral for the Console interface. In this lab, we will change the baud rate in the devicetree for this `flexcomm4_lpuart4` node, and confirm how the change affects the Console interface.

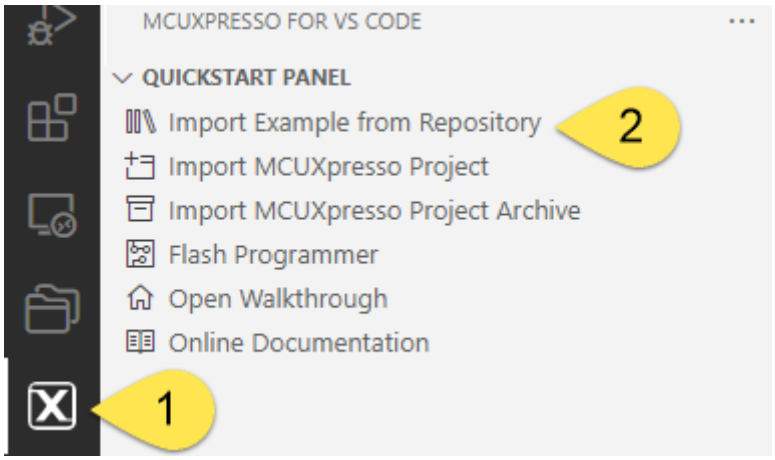
```
chosen {
    zephyr,console = &flexcomm4_lpuart4;
    zephyr,shell-uart = &flexcomm4_lpuart4;
};
```

2. Explore with the MCUXpresso Device Tree viewer

First we will import the Zephyr *button* sample application into VS Code, build it, and explore the devicetree.

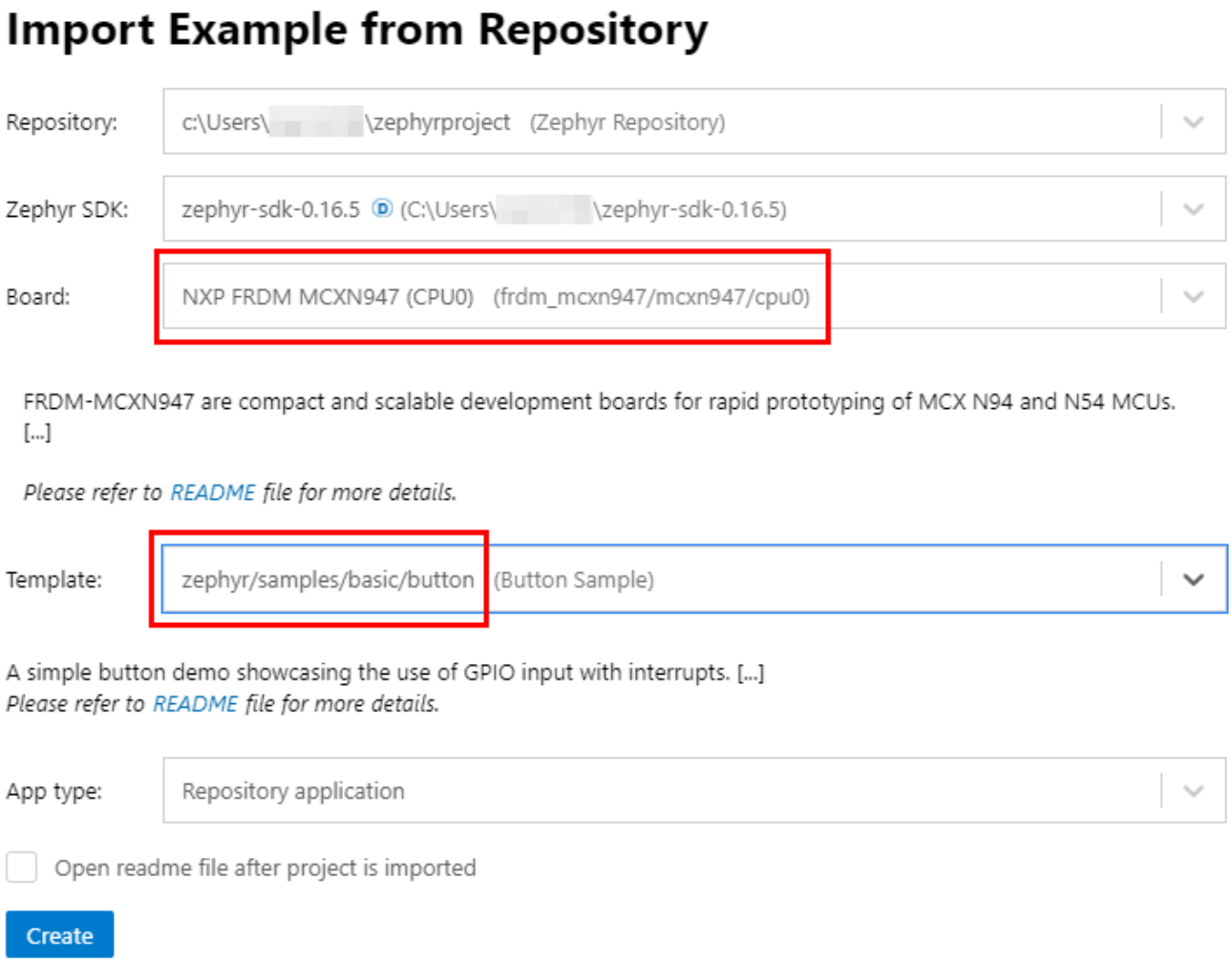
This lab focuses on using the Devicetree Viewer included in the MCUXpresso extension. The Viewer enables a user to confirm the settings in the devicetree, and help find the source that specifies those settings. There is another option to confirm the devicetree without the Viewer. When an application is built, the tools generate a file `zephyr.dts` which lists the final merged devicetree used during that build. The build generates a build folder, and this file is located at `<build path>/zephyr/zephyr.dts`.

1. Open VS Code. In the Quickstart Panel, click **Import Example from Repository**. If needed, the steps to import an example, build, and debug it are included in [[Zephyr Lab MCXN947 Hello World]].

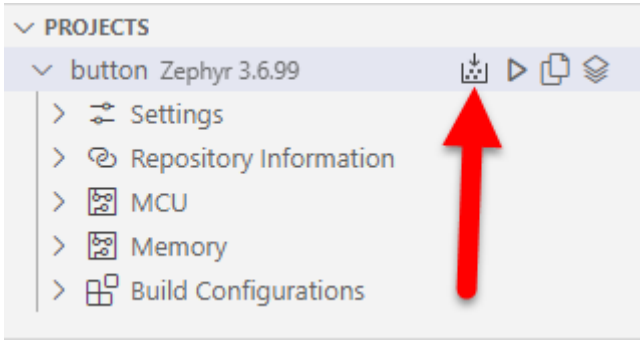


2. Import the *button* sample with the following settings:

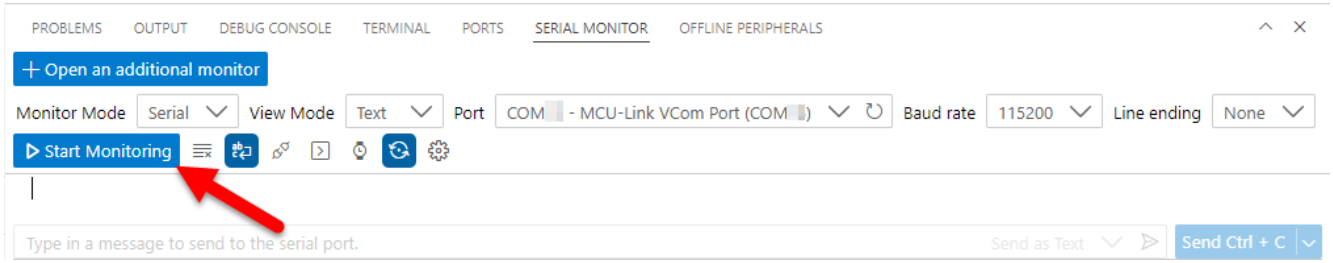
   - For the board, you can type **n947** to filter the board target **frdm_mcxn947/mcxn947/cpu0**. Be sure this board target ends with `cpu0`.
   - For the example template, type **button** to filter the app **zephyr/samples/basic/button**
   - For Application type, select **Repository Application**
 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



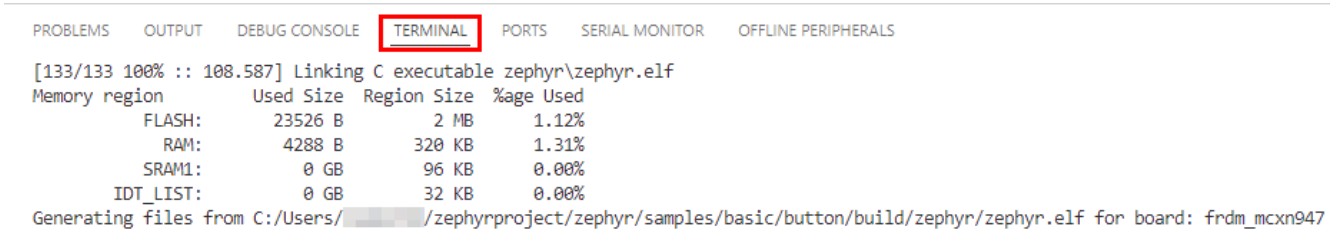
3. Build the *button* project



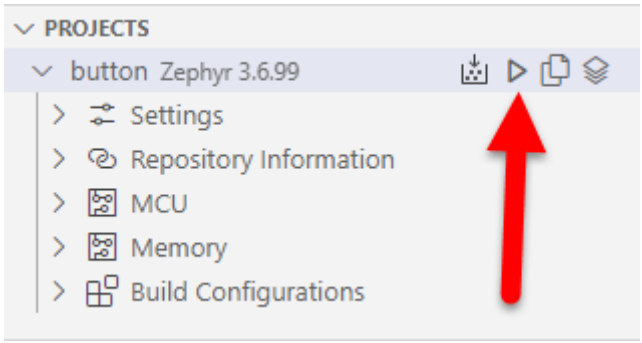
4. If not already connected to the board, connect the USB Type-C cable to J17, to power and debug the board. In VS Code, connect the Serial Monitor to the board's COM port, and click **Start Monitoring**.



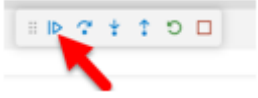
5. When the build completes, the Terminal view shows the image executable *zephyr.elf* was built, and gives a summary of the memory usage.



6. Debug the *button* project.



7. The execution will pause. To continue execution click **Continue** on the debug options.



8. In the Serial Monitor view, you will see the app prints the following:

Button sample output in Serial Monitor:

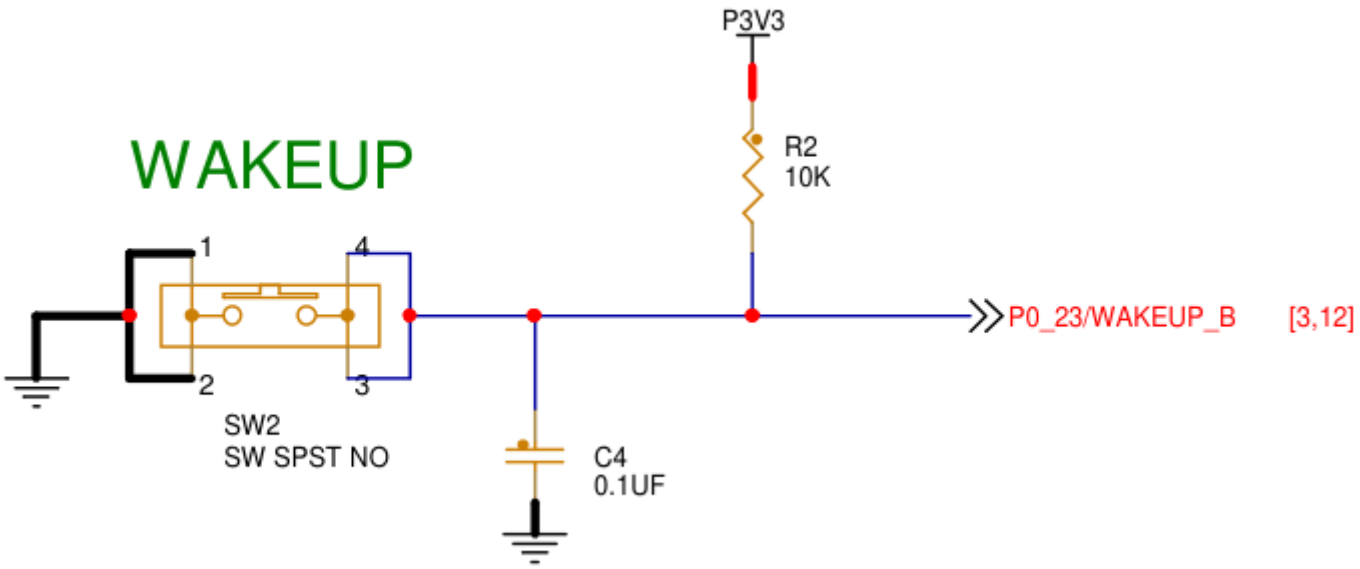
```
*** Booting Zephyr OS build v3.6.0-3471-gaa72e8178cea ***
Set up button at gpio@96000 pin 23
Set up LED at gpio@96000 pin 10
Press the button
```

9. Notice the RGB LED D2 is not lit (near the top-center of the board). **Press the button SW2** in the upper-right corner of the board. The red LED turns on when the button is pressed, and turns back off when the button is released. The Console also prints every time the button is pressed.

   Serial Monitor output after *button* is pressed twice:

```
*** Booting Zephyr OS build v3.6.0-3471-gaa72e8178cea ***
Set up button at gpio@96000 pin 23
Set up LED at gpio@96000 pin 10
Press the button
Button pressed at 181695378
Button pressed at 425603040
```

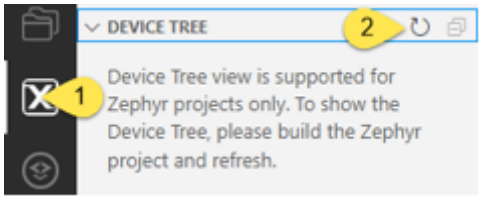
Also notice the Console prints the "Button pressed" message when the button is pressed, not when the button is released. This is important for a devicetree change we will make in the next lab section. The board schematic snippet below shows pressing the button *SW2* shorts the GPIO signal to GND. So the `Button pressed` message occurs on the falling edge of this GPIO signal.



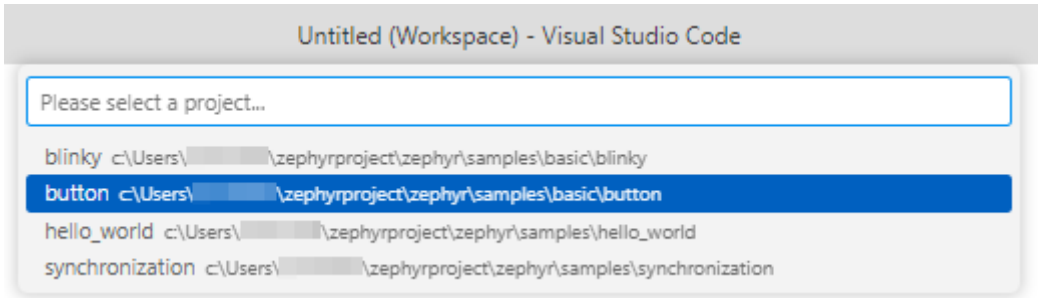
10. Stop the debugger

Now that we have the default *button* sample working, we will explore the devicetree, the `sw0` alias, and the `button_0` GPIO node.

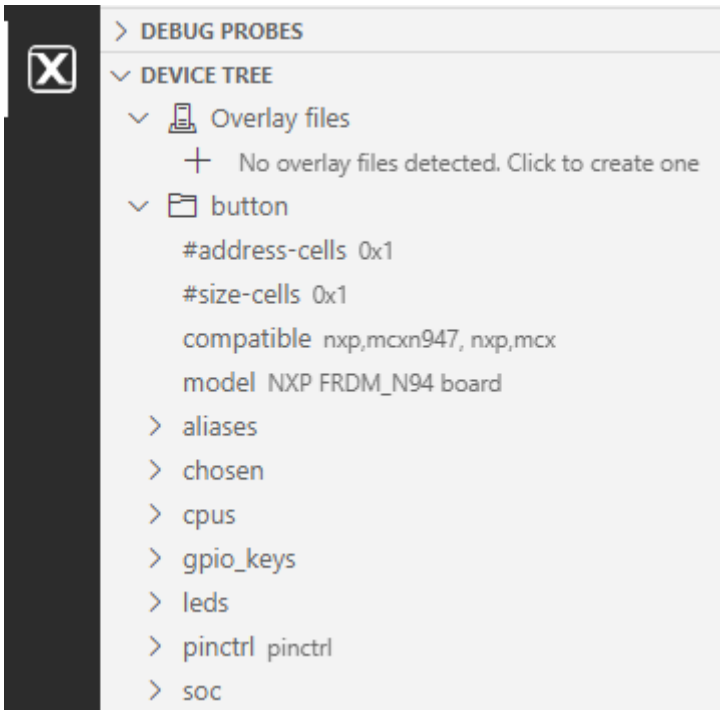
11. Open the MCUXpresso Device Tree view in VS Code. Click the MCUXpresso extension on the left toolbar, and find the Device Tree view. Click the **Refresh Device Tree button**. Notice the refresh should be done after the Zephyr application is built. That build generates the devicetree files used by this viewer.



12. If multiple projects are open, VS Code will prompt to select the project of the devicetree to view. **Select the *button* project**.

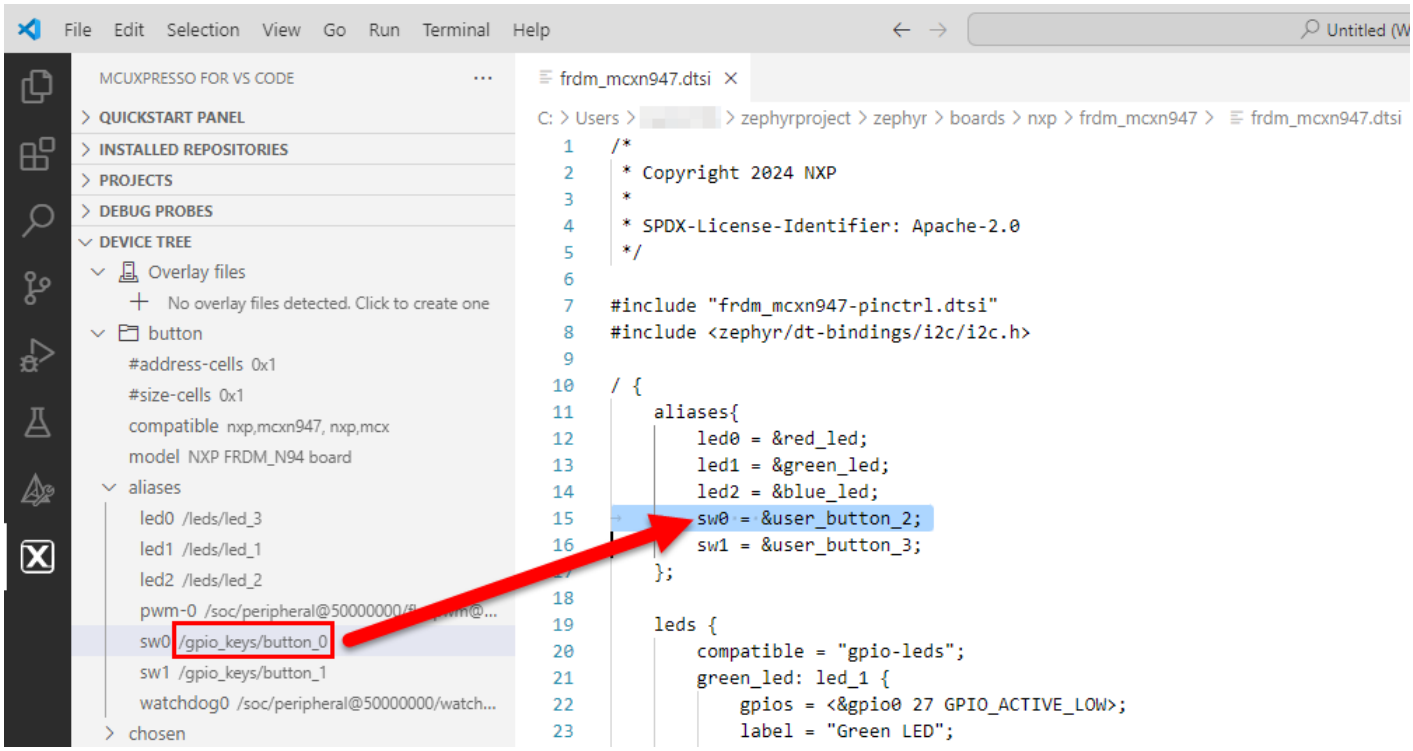


The viewer takes some time parsing the devicetree before displaying it:

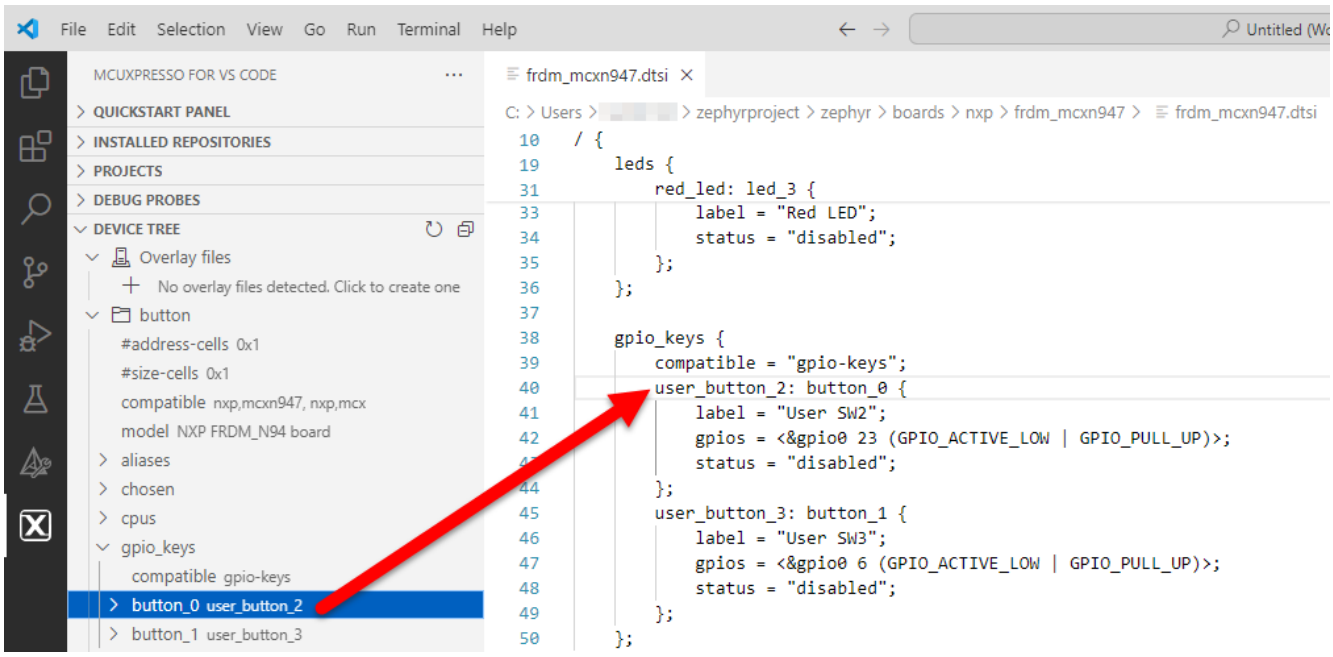


13. In the Device Tree view, expand the `aliases` group, and **click on the `sw0` alias**. When clicking a node or property, the MCUXpresso Device Tree viewer will open the source location where that node is defined or a property is set. In this case, the viewer opens the board devicetree include file frdm_mcxn947.dtsi. And the viewer finds the line of code where the alias `sw0` is set to node label `user_button_2`.

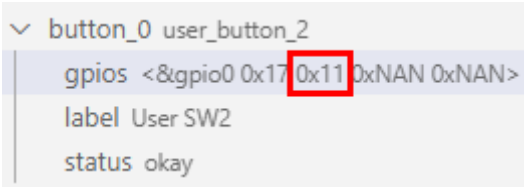
Because `sw0` is an alias, notice the Device Tree viewer also shows us the full devicetree path to the aliased node. This shows the `sw0` alias points to the sub-node `button_0`, which is part of the node `gpio_keys`.



14. In the Device Tree viewer, expand the node `gpio_keys`, and **click on the `button_0` node**. The viewer finds this node in the same board devicetree file.



15. Also notice the Device Tree viewer shows the property values for nodes. Here, the property flags `GPIO_PULL_UP` and `GPIO_ACTIVE_LOW` combine to the value of 0x11. This will be important in the next section when we modify this property.



3. Modify a board devicetree file

Now that we found the `button_0` GPIO node in the board's devicetree source file, we will modify its properties, and see how it changes the behavior of the button sample application. We will change this pin to be active-high, and confirm that the button event in the app now occurs on the rising-edge of the button release.

16. The board devicetree include file *frdm_mcxn947.dtsi* is already open in the editor. Modify the `gpios` property for the `button_0` node highlighted below, and change it to `GPIO_ACTIVE_HIGH`. Save the file (Ctrl-S).

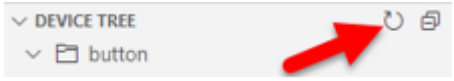
```
gpio_keys {
    compatible = "gpio-keys";
    user_button_2: button_0 {
        label = "User SW2";
```

```
        // change the property below to GPIO_ACTIVE_HIGH
        gpios = <&gpio0 23 (GPIO_ACTIVE_HIGH | GPIO_PULL_UP)>;
```

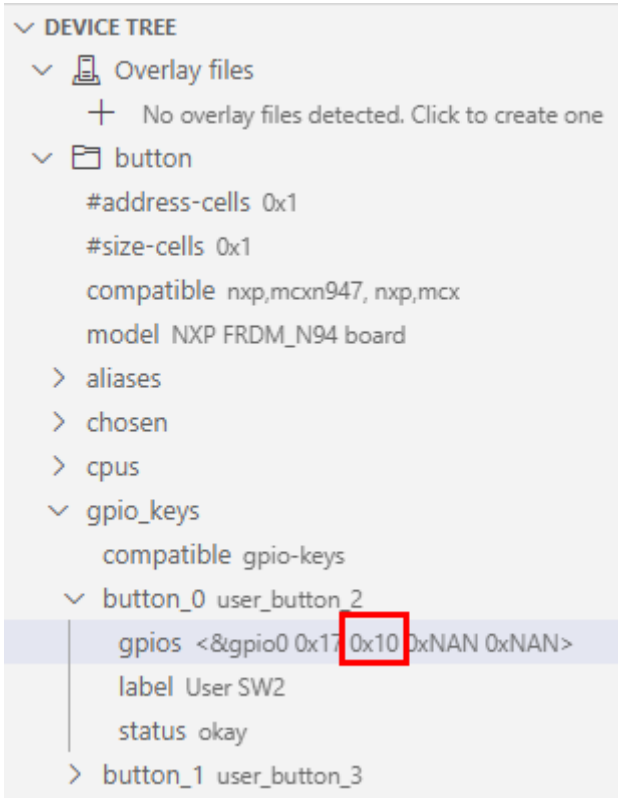
```
        status = "disabled";
    };
};
```

17. Build the *button* project.

18. Wait for the build to complete, and then **Refresh the Device Tree viewer**.

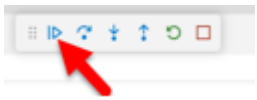


The viewer will take some time to parse the devicetree, and then will show the property update. Notice the property value for the flags has now changed from 0x11 to **0x10**.



19. Debug the *button* project.

20. Continue executing the application.



21. **Press the button *SW2*** on the board. Notice the red LED is now on until the button is pressed. And using the Serial Monitor, notice the Console prints now when the button is released, on the rising-edge of the GPIO signal.

    Button sample Console output

```
*** Booting Zephyr OS build v3.6.0-3471-gaa72e8178cea ***
Set up button at gpio@96000 pin 23
Set up LED at gpio@96000 pin 10
Press the button
Button pressed at 36022917
```

22. Stop the debugger

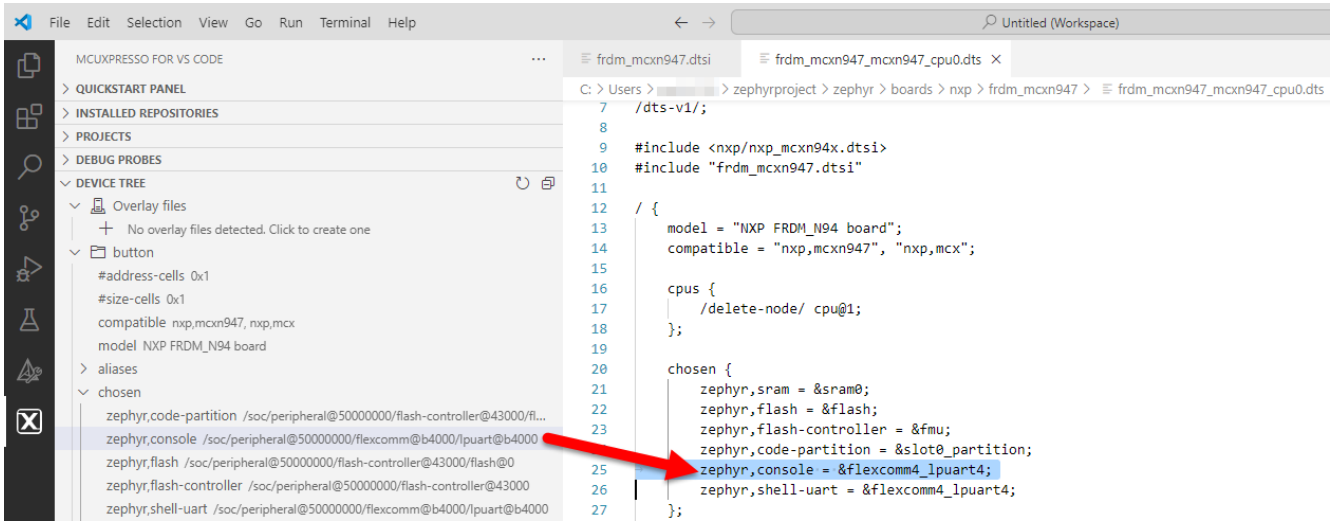
4. Create a devicetree overlay file for UART baud rate

In the last section, we changed the hardware configuration by modifying the board's devicetree file. That source file is used by every application built for that board. That is a common method when working with a custom board, which would likely be managed in your personal repository, not in the public Zephyr repository. For a custom board, it is best to clone an existing board using a similar SOC, like NXP's evaluation boards. And then modify that new board's files to match your hardware and applications. For the specific steps to clone a board, see Zephyr's Board Porting Guide.

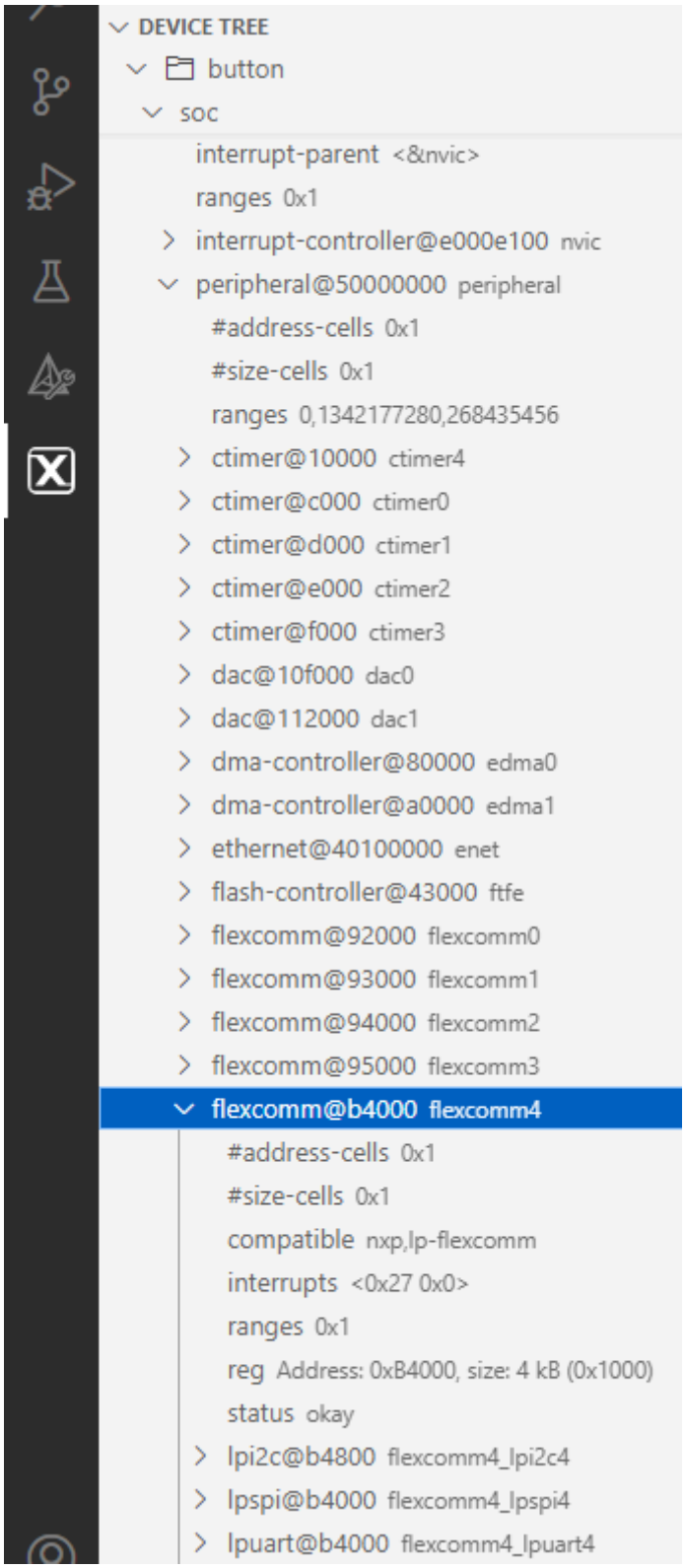
In this section, we will modify the devicetree using a different method: called an overlay file. Overlays work with the existing devicetree files, rather than modifying the original files. The devicetree properties in the overlay files will override the same settings in the board and SOC devicetree files. Basically, the board files create a default hardware configuration used for most applications. But if one application wants a change to the hardware configure, it can add an overlay file. Since Zephyr has hundreds of sample and test applications that run on the same boards, some Zephyr applications use overlay files when hardware changes are required. For example, the Zephyr ADC driver sample at samples/drivers/adc/adc_dt has many board overlay files for different boards, which specify the channel properties for the ADC example. If you test your own application on an eval board, overlay files are a good option to configure the hardware for your needs.

In this lab section, we will create a simple overlay that changes the baud rate of the `flexcomm4_lpuart4` used for the Console. The first steps will use the MCUXpresso Device Tree viewer to confirm the default baud rate setting.

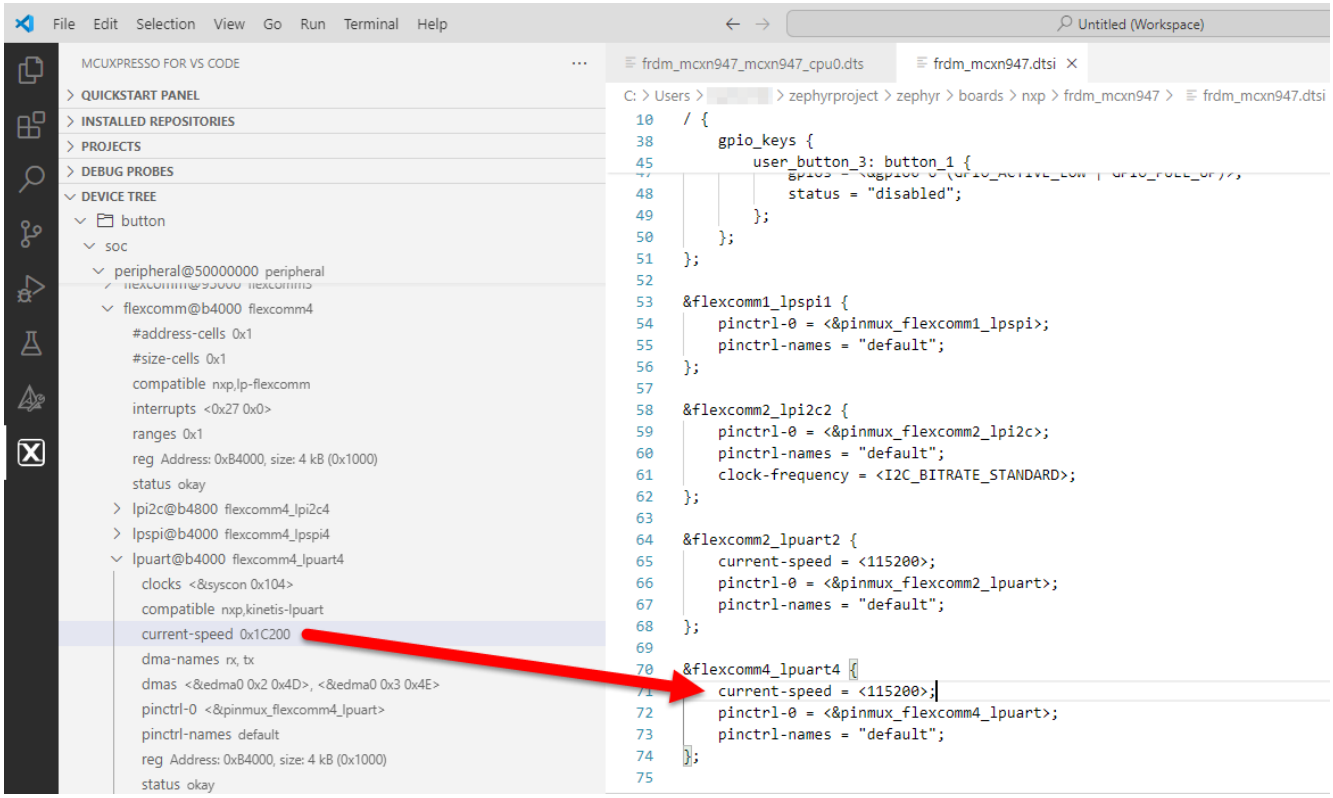
23. In VS Code, return to the Device Tree viewer. Expand the chosen group, and **click on the `zephyr,console` node**. The viewer will find the line in the board devicetree file frdm_mcxn947_mcxn947_cpu0.dts, which points this chosen node to the `flexcomm4_lpuart4` node. Notice the viewer gives us the full devicetree path to this node on the left: the heirarchy of nodes from the top level is `soc->peripheral->flexcomm->lpuart`.



24. In the Device Tree viewer, **expand the following nodes**: first `soc`, then `peripheral`, then `flexcomm4`.



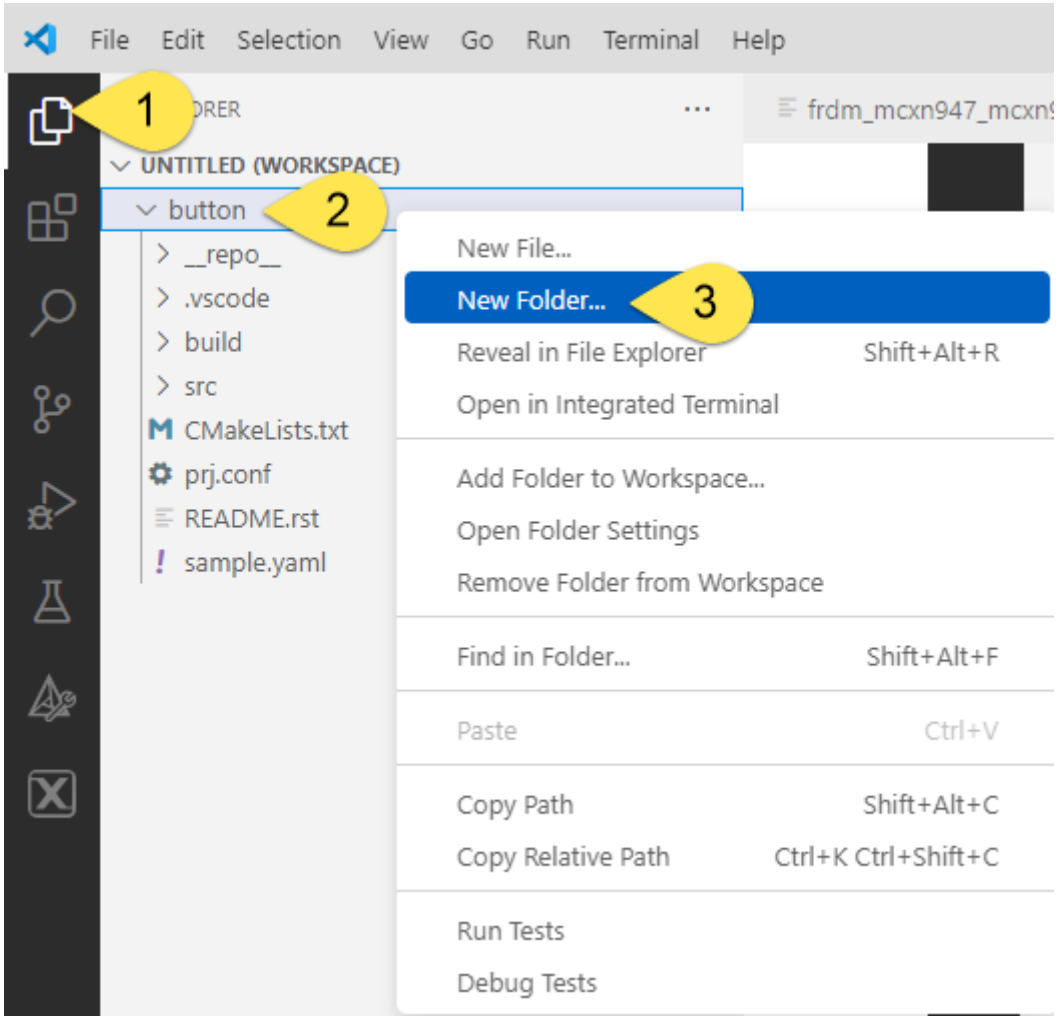
25. **Expand the node** `flexcomm4_lpuart4`. The properties for that node are listed. For the baud rate, **click on the property** `current-speed`. The viewer finds this property setting in the board file, and we confirmed the default baud rate is 115,200. Notice the viewer shows the `current-speed` value is 0x1C200; this is the hex equivalent of 115,200.



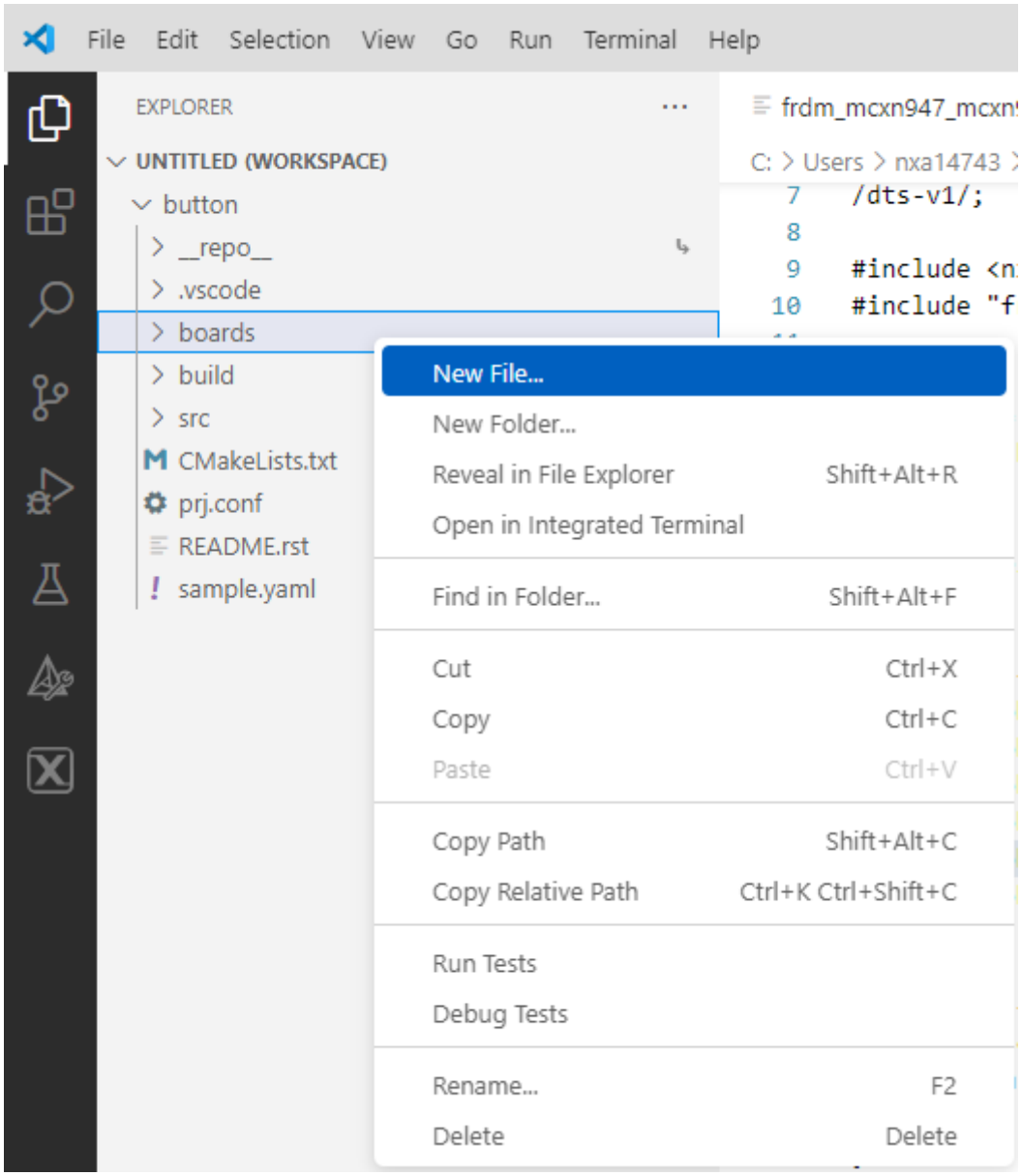
Now that we learned how to confirm the default baud rate, and what that rate is, we will create an overlay file to change that baud rate.

The Devicetree Viewer can create overlay files. But due to a bug in v1.7.50 of the MCUXpresso extension for this FRDM-MCXN947 board, we will create the overlay file manually:

26. In the Explorer view, **right-click the button folder**, and select **New Folder...**. Name the new folder `boards`.

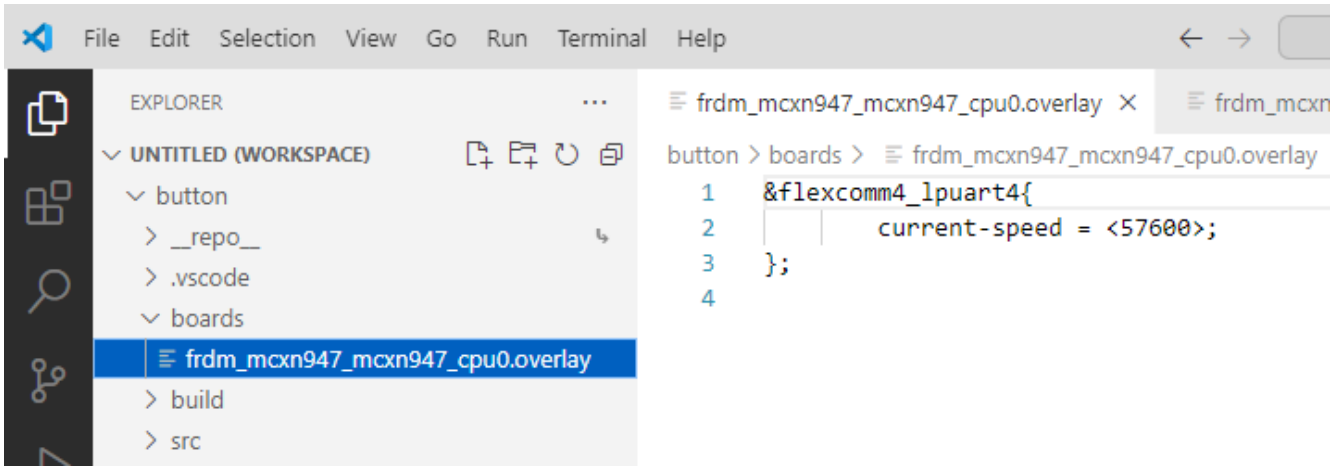


27. **Right-click the new `boards` folder**, and select **New File...** The filename must match the board target. Name the file `frdm_mcxn947_mcxn947_cpu0.overlay`.

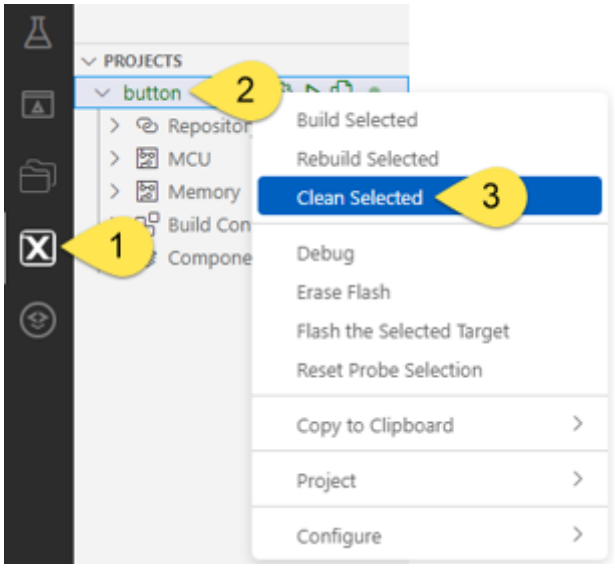


28. Modify the file *frdm_mcxn947_mcxn947_cpu0.overlay*, and **add the lines below** to override the `flexcomm4_lpuart4` property. Save this file (Ctrl-S).

```
/* Add these lines */
&flexcomm4_lpuart4{
        current-speed = <57600>;
};
```

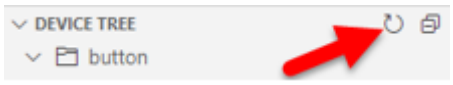


29. Clean the *button* project. Note that asking VS Code to build the project without cleaning it will not work. The build system will not detect that a new overlay file was added to the application, so it will think nothing has changed since the last build. To force the build system to include the new file, go to the MCUXpresso view, right-click the *button* project, and **select Clean Selected**.

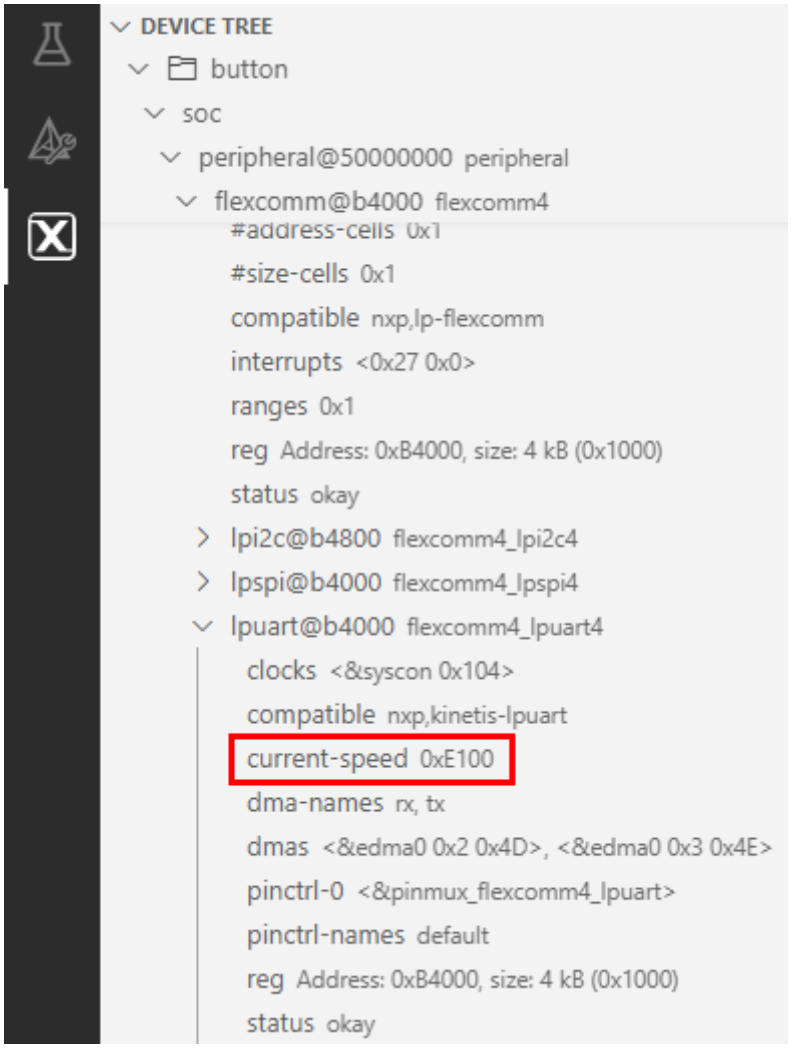


30. Build the *button* project.

31. Refresh the Device Tree viewer.

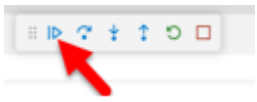


The viewer will take some time to parse the devicetree, and then will show the property update. Notice the property value for the `current-speed` has now changed from 0x1C200 to **0xE100**.



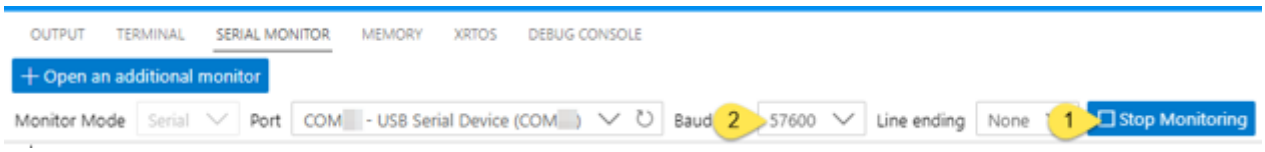
32. Debug the *button* project.

33. Continue executing the application.



34. The Serial Monitor shows a bunch of garbage characters were received. Or you may find the Serial Monitor is not receiving any characters. Oops, we forgot to update the baud rate to match our devicetree change.

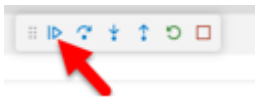
35. Click the **Stop Monitoring button**. Then, change the **baud rate to 57,600**.



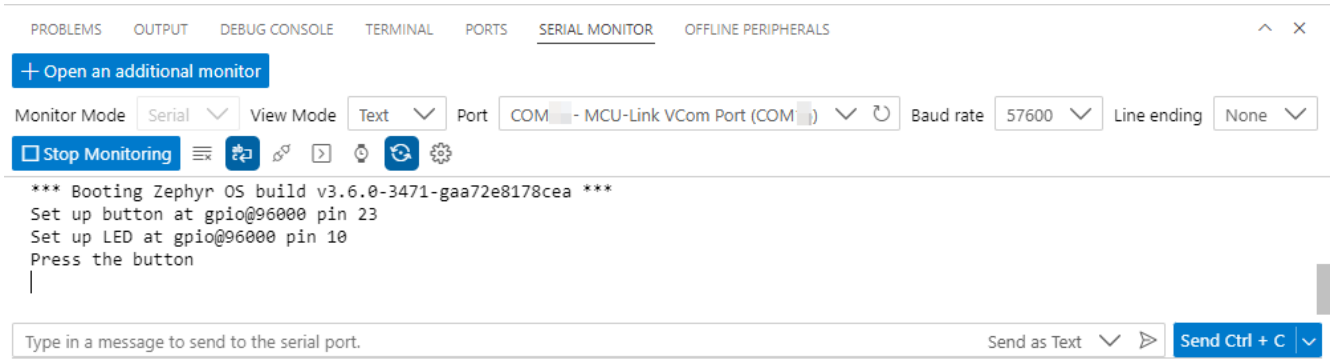
36. Click the **Start Monitoring button** with the new baud rate setting.

37. In the debugger pop-up, **click the Restart button**.

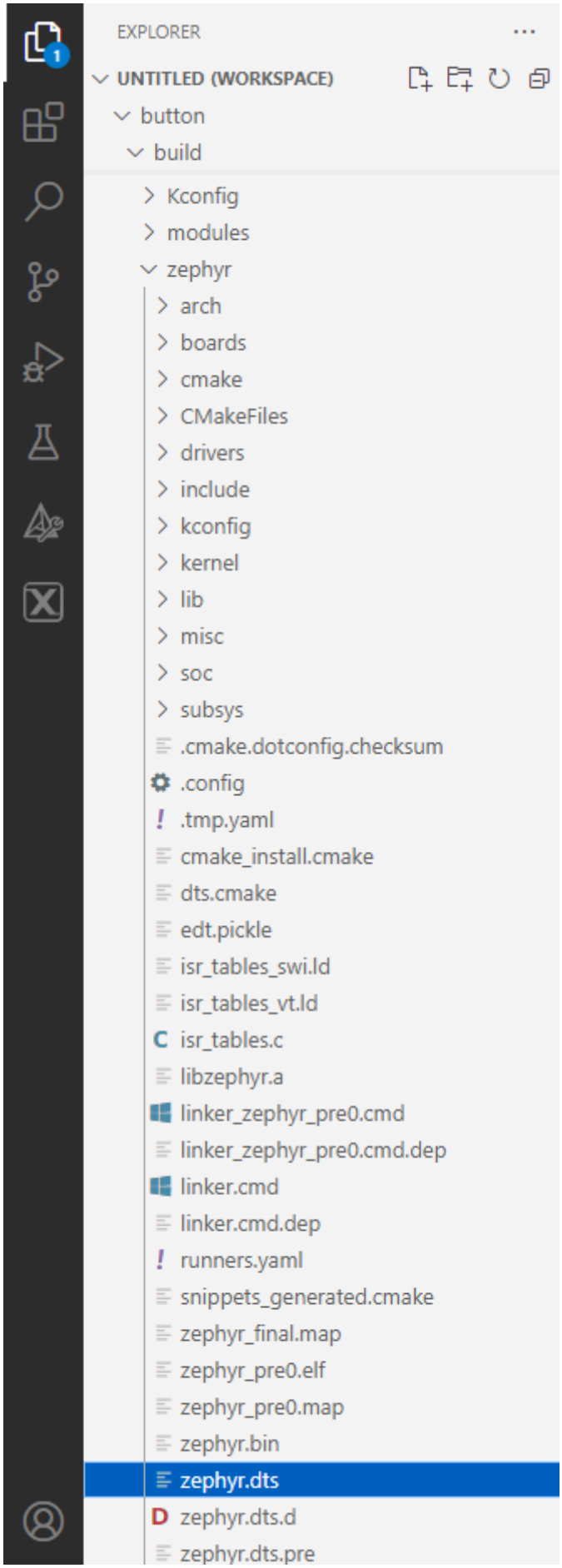
38. click the Continue button.



39. Much better. This confirms the devicetree overlay file we added changed the `flexcomm4_lpuart4` baud rate to 57,600.



40. Optional: one alternative to using the Devicetree Viewer to confirm the devicetree settings is the generated `zephyr.dts` file. This file is generated during the build, and is the final merged devicetree settings used during the build. In the Explorer view, under the *button* project, expand the folders **build** then **zephyr**, and then open the file `zephyr.dts`. Search this file for `flexcomm4_lpuart4` to confirm the devicetree settings:

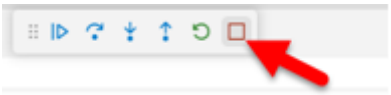

EXPLORER
UNTITLED (WORKSPACE)
button
build
Kconfig
modules
zephyr
arch
boards
cmake
CMakeFiles
drivers
include
kconfig
kernel
lib
misc
soc
subsys
.cmake.dotconfig.checksum
.config
.tmp.yaml
cmake_install.cmake
dts.cmake
edt.pickle
isr_tables_swi.ld
isr_tables_vt.ld
isr_tables.c
libzephyr.a
linker_zephyr_pre0.cmd
linker_zephyr_pre0.cmd.dep
linker.cmd
linker.cmd.dep
runners.yaml
snippets_generated.cmake
zephyr_final.map
zephyr_pre0.elf
zephyr_pre0.map
zephyr.bin
zephyr.dts
zephyr.dts.d
zephyr.dts.pre

```
flexcomm4_lpuart4: lpuart@b4000 {
    compatible = "nxp,kinetis-lpuart";
    reg = < 0xb4000 0x1000 >;
    clocks = < &syscon 0x104 >;
    dmas = < &edma0 0x2 0x4d >, < &edma0 0x3 0x4e >;
    dma-names = "rx", "tx";
    status = "okay";
    current-speed = < 0xe100 >;
    pinctrl-0 = < &pinmux_flexcomm4_lpuart >;
    pinctrl-names = "default";
};
```

5. Clean up after lab

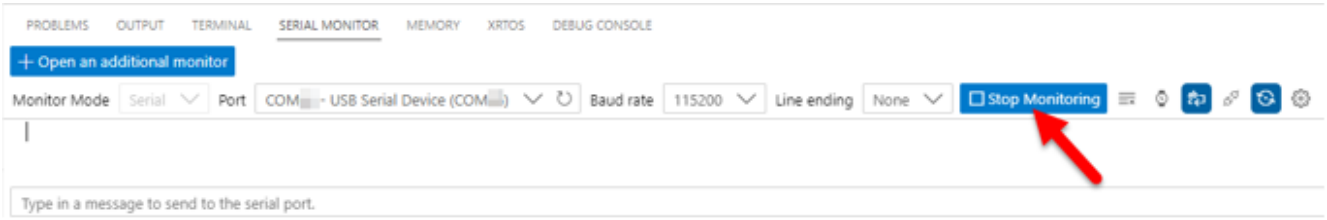
This Devicetree lab has completed. But the following steps will clean up the VS Code workspace:

41. Stop the Debugger



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

43. 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

- Zephyr Devicetree documentation
- Zephyr Devicetree HOWTOs
- Devicetree specifications
- Deep Dive in Zephyr Device model
- Zephyr's Board Porting Guide
- Webinar with more Devicetree training: Application Portability Made Easy With Zephyr OS and NXP

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