Debugging Raspberry Pi Pico W using Windows Subsystem for Linux (WSL)

Date: 2024-06-21

Table of Contents

Windows Setup

1. Administrator PowerShell
2. Windows 11 Pro with WSL and Ubuntu distribution installed in WSL
   ```
   WSL version: 2.1.5.0
   Kernel version: 5.15.146.1-2
   WSLg version: 1.0.60
   MSRDC version: 1.2.5105
   Direct3D version: 1.611.1-81528511
   DXCore version: 10.0.25131.1002-220531-1700.rs-onecore-base2-hyp
   Windows version: 10.0.22635.3785
   ```
3. Install usbipd-win. I used version 4.2.0.
   https://github.com/dorssel/usbipd-win/releases/tag/v4.2.0
   The installer installed it to c:\program files\usbipd-win\
4. JetBrains CLion IDE version version 2023.3 (or later). Make sure CLion is setup to open the project as a remote WSL project.

Passing the USB device to WSL

The USB device isn't exactly attached in the usual sense. We use a USB over TCP/IP transport tool called usbipd-win to effectively attach the usb debugging probe to WSL.

You need to use the **administrator PowerShell** to run these commands. Identify the BUSID of the usb debug probe. In my case it is 2-4.

```
cd c:\Program Files\usbipd-win
usbipd.exe list
```

```
PS C:\Program Files\usbipd-win> usbipd.exe list
Connected:
BUSID  VID:PID    DEVICE                                                        STATE
1-6    8087:0029  Intel(R) Wireless Bluetooth(R)                                Not shared
2-4    2e8a:000c  CMSIS-DAP v2 Interface, USB Serial Device (COM8)              Attached
5-2    1532:00b7  USB Input Device, Razer DeathAdder V3 Pro                     Not shared
5-3    0c45:5004  USB Input Device                                              Not shared
7-3    2a39:3f82  RME Fireface UCX II                                           Not shared
8-3    0b05:18f3  AURA LED Controller, USB Input Device                         Not shared
```

After you identify the BSID, bind and then attach the device to WSL.

```
.\usbipd.exe bind --busid 2-4
.\usbipd.exe attach --wsl --busid 2-4
```

If you need to reattach the device, you can physically disconnect and reconnect the cable or call the detach command and then rerun the attach command.

```
.\usbipd.exe dettach --wsl --busid 2-4
```

WSL Ubuntu Setup

1. I'm using the Ubuntu distribution in WSL with kernel 5.15.146.1-microsoft-standard-WSL2
2. Run apt install to install the required build and debug dependencies.
   - **apt install build-essential pkg-config libusb-1.0-0* libhidapi-dev cmake gcc-arm-none-eabi libnewlib-arm-none-eabi libstdc++-arm-none-eabi-newlib minicom gdb-multiarch libtool**
3. Change directory to /opt/ and clone the Raspberry Pi version of **openocd**. Compile and install it. This was the exact branch I used **https://github.com/raspberrypi/openocd/tree/rp2040-v0.12.0**
   - **cd /opt/**
     git clone https://github.com/raspberrypi/openocd.git
     cd /opt/openocd
     ./bootstrap/
     OPENOCD_CONFIGURE_ARGS="--enable-ftdi --enable-sysfsgpio --enable-bcm2835gpio --enable-picoprobe –enable-cmsis-dap"
     ./configure $OPENOCD_CONFIGURE_ARGS
     make -j $(nproc)
     sudo make install
4. Now you need to edit the pico-debug.cfg board file that was just installed by the custom openocd repo.
   - **sudo vim /usr/local/share/openocd/scripts/board/pico-debug.cfg**
   - Make sure it has the following contents. In my case, the target was set to use an obsolete file; it should be target/rp2040.cfg.

     ```
     source [find interface/cmsis-dap.cfg]
     adapter speed 4000

     set CHIPNAME rp2040
     source [find target/rp2040.cfg]
     ```

5. By default, Linux doesn't allow users to write to USB devices. We need to setup some permissions.
   - **# Add your user to the dialout and plugdev groups**
     sudo usermod -a -G dialout $(whoami)
     sudo usermod -a -G plugdev $(whoami)
     # Add a udev rule file specifying permissions for the ttyACM0 serial device. This is what the Raspberry Pi Debug Probe registers as every time it is attached.
     sudo vim /etc/udev/rules.d/00-usb-permissions.rules
     # Add this line. Save the file, and close it.
     KERNEL=="ttyACM0", SUBSYSTEM=="tty", MODE="0666", GROUP="dialout"
   - **KERNEL=="1-1:1.1", SUBSYSTEM=="usb", MODE="0666", GROUP="plugdev"**
   - **KERNEL=="1-1", SUBSYSTEM=="usb", MODE="0666", GROUP="plugdev"**
   - **KERNEL=="usb1", SUBSYSTEM=="usb", MODE="0666", GROUP="plugdev"**
     # Reload the udev daemon
     sudo udevadm control --reload-rules
6. Change directory to /usr/local/lib and clone the pico-sdk repo
   - **cd /usr/local/lib/pico-sdk/**
     git clone https://github.com/raspberrypi/pico-sdk.git --branch master
7. Clone the pico-examples code in your home directory.

- **`cd ~ && mkdir raspberry_pi && cd raspberry_pi`**
  `git clone` https://github.com/raspberrypi/pico-examples.git

8. Edit your ~/.bashrc file and add the following lines.
   - **`export PICO_SDK_PATH="/usr/local/lib/pico-sdk"`**
     `export PICO_EXAMPLES_PATH="$HOME/raspberry_pi/pico-examples"`
9. At this point, it is a good idea to shutdown and reload WSL.

JetBrains CLion IDE Setup

I'm using CLion for remote debugging because it supports embedded development. I followed the official documentation https://www.jetbrains.com/help/clion/openocd-support.html#run-debug.

1. Start CLion and open the pico-examples as a project under the WSL remote development.

2. Edit the CMake settings under *Build*, *Execution*, *Deployment*, and add the following cmake, environment, and cache variables.
   Note: The **cache variables** section is not expanded by default.

CMake Options	-DPICO_BOARD=pico_w -DPICO_CYW43_SUPPORTED=1 -DCMAKE_BUILD_TYPE=Debug -DPICO_PLATFORM=rp2040
Environment	PICO_SDK_PATH=/usr/local/lib/pico-sdk
Cache Variables	PICO_SDK_PATH=/usr/local/lib/pico-sdk

3. In the Settings dialog, find the *Build*, *Execution*, *Deployment* > *Embedded Development* panel and set the OpenOCD Location to **/usr/local/bin/openocd**. This path should have been created after calling the "make install" command when building the custom openocd project from the raspberry pi github organization.

4. In the CLion code editor, open the hello_serial.c program and build it. This should be enough for you to be able to build the project. CLion will process the build symbols.

5. Open the Run/Debug Configurations panel and add an OpenOCD Download & Run launcher. Choose board/pico-debug.cfg for the board config file.

6. Run the OpenOCD Download & Run launcher.

Connect Debug Serial Console (minicom)

The Pico W supports UART connections on GPIO pins 1 (TX), 2 (RX), and 3 (GND). Connect these to the debug probe, and use the minicom program to monitor the connection.

You may run **minicom -s** to enter the setup menu and select the correct serial device as /dev/ttyACM0.

Alternatively, you can specify all options as arguments.

```
minicom --color on --baudrate 115200 --device /dev/ttyACM0
```

Troubleshooting

1. **Unable to connect OpenOCD.** Use the --debug switch to get additional information.

   - ```
     /usr/local/bin/openocd -c "tcl_port disabled" -c "gdb_port 3333" -c "telnet_port 4444" -s /usr/local/share/openocd/scripts -f board/pico-debug.cfg -c "program /home/alexander/raspberry_pi/pico-examples/build/hello_world/serial/hello_serial.elf" -c "init;reset init;exit;" --debug
     ```

     ```
     It is likely you'll see a permission problem
     ```

     ```
     cmsis_dap_usb_bulk.c:105 cmsis_dap_usb_open(): could not open device 0x1d6b:0x0003: Access denied (insufficient permissions)

     In this case, use the udevadm info command to figure out what permissions are needed for /dev/ttyACM0.
     ```

2. **Determine what permissions are required for /dev/ttyACM0.**
   I have had to run the attribute walk and then add additional kernels & subsystems to the udev rule to allow my non-root WSL user to invoke the openocd commands. Simply adding the tty serial device wasn't enough. I had to add multiple kernels found under the usb subsystem.

   - `udevadm info --attribute-walk –name=/dev/ttyACM0`

3. **CLion Debugger is not working.**
   First check to see that you can manually run the openocd command that CLion is trying to execute, but add the --debug command line argument. If this command works, then check the gdb version used by the launcher configuration.

4. **Still having trouble connecting.**

   - Sometimes the physical USB cable needs to be reconnected securely.
     ```

- Double check all the connections to the SWD pin headers are secure.
- Restart wsl via PowerShell **wsl --shutdown**.
- Disconnect and reconnect the USB device via usbipd.exe.
- Verify that the udev permissions are set correctly after the usb device is reconnected. Check **stat /dev/ttyACM0** and make sure the permissions match the udev rule (0666).