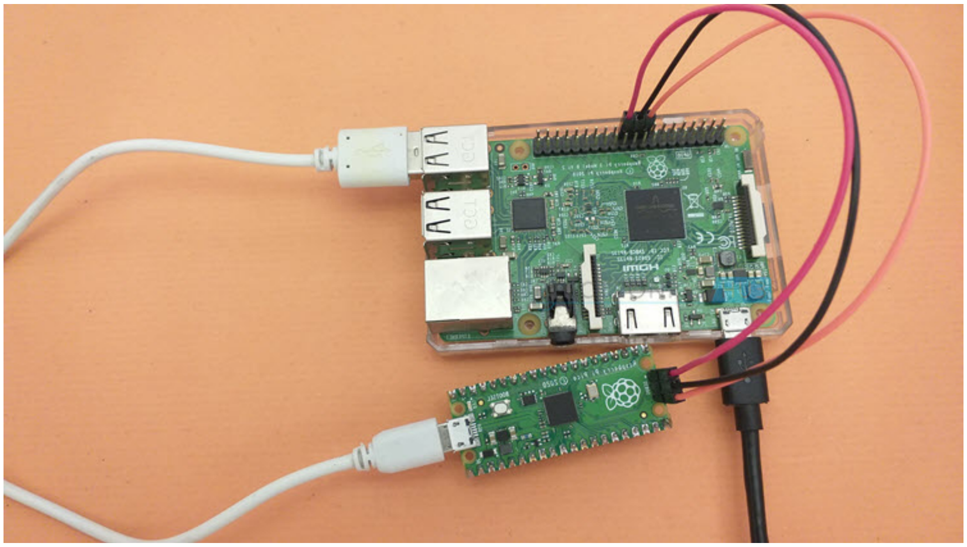
electronicshub.org

How to Program and Debug Raspberry Pi Pico with SWD?

By Ravi Teja (Embedded Engineer)

7-9 minutes

In this tutorial, we will learn how to upload a program to Raspberry Pi Pico using SWD and also debug with SWD. Till now, we have been using drag-and-drop method for uploading program to Pico. But if you want to use SWD Interface of Pico to program and debug, then this tutorial is for you. Learn the steps for Programming Raspberry Pi Pico with SWD and also debugging the code with SWD, OpenOCD and GDB.



A Brief Note on SWD

Have you ever worked on developing an Embedded System Application? If the answer is yes, then you might be familiar with the term ‘Debugging’. In embedded systems, which are mainly designed with a single task (or a very small number and specific set of tasks) in mind, the process of debugging and testing is very critical as we often work as low as CPU Register level.

There are several hardware and software solutions for debugging embedded systems. One such offering is called Serial Wire Debug or SWD in short. SWD is a Debug and Trace Port embedded into the silicon of most modern ARM based Microcontrollers and Microprocessors.

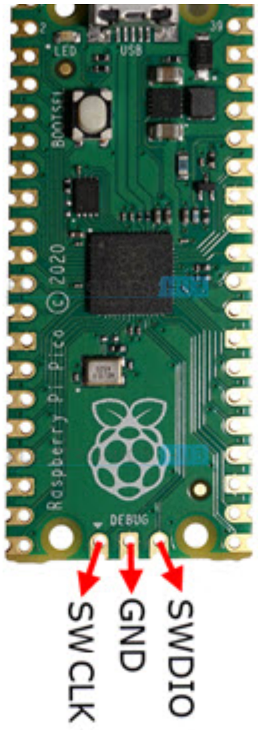
Using a SWD Probe (a small hardware often connected to SWD Port of the Microcontroller and maps them to USB) you can program the Flash of the Microcontroller, Debug the Firmware, Add Breakpoints, Stepping through the Code, etc. with just two wires.

The combination of SWD, GDB (GNU Debugger) and OpenOCD (an on-chip debugger which supports debugging, in-system programming and boundary scan for embedded systems) is a very powerful debugging setup, especially for ARM Cortex series of Processors.

Raspberry Pi Pico SWD Programming and Debug

Like all ARM Cortex processors, the Raspberry Pi Pico also has dedicated hardware for debugging via the SWD Interface. The two wires required for SWD Debugging are called SWDIO (bidirectional SWD Data) and SWCLK (SWD Clock).

On the Raspberry Pi Pico, the SWD Pins are separated from the rest of the GPIO Pins and are placed at the bottom of the Board.



The 2-wire SWD Interface of RP2040 on the Raspberry Pi Pico board allows you to do the following:

- Upload program into External Flash or Internal SRAM.
- Control the state of execution of the processor i.e., run, halt, step, set breakpoints, etc.
- Access processors memory and IO peripherals (which are memory mapped) through the system bus.

Installing Tools in Raspberry Pi

As mentioned earlier, GDB and OpenOCD are required for debugging any ARM Cortex Processor. So, we will now install these two in our host system, which in my case is a Raspberry Pi running the latest Raspberry Pi OS.

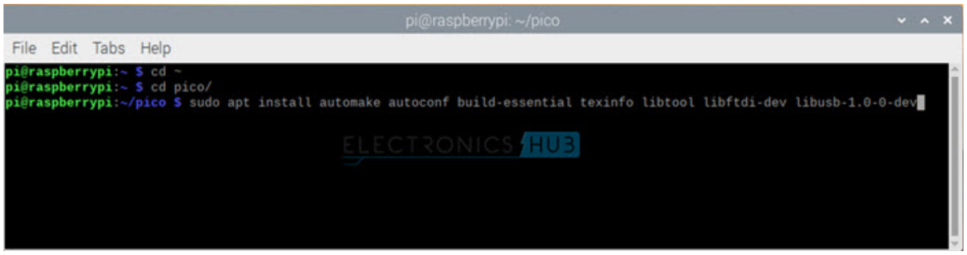
OpenOCD

To understand the SWD Protocol and control the ARM Cortex Processor (two in case of RP2040), you need a special translator called OpenOCD. Let us now see how to install OpenOCD in Raspberry Pi.

NOTE: The following steps will install OpenOCD in /home/pi

/pico/openocd.

sudo apt install automake autoconf build-essential texinfo libtool libftdi-dev libusb-1.0-0- dev



```
pi@raspberrypi:~ $ cd ~
pi@raspberrypi:~ $ cd pico/
pi@raspberrypi:~/pico $ sudo apt install automake autoconf build-essential texinfo libtool libftdi-dev libusb-1.0-0-dev
```

This will install all the tools require by OpenOCD. Next, we will clone the OpenOCD into our host and install OpenOCD. Enter the following commands one after the other.

git clone https://github.com/raspberrypi/openocd.git –recursive –branch rp2040 –depth=1

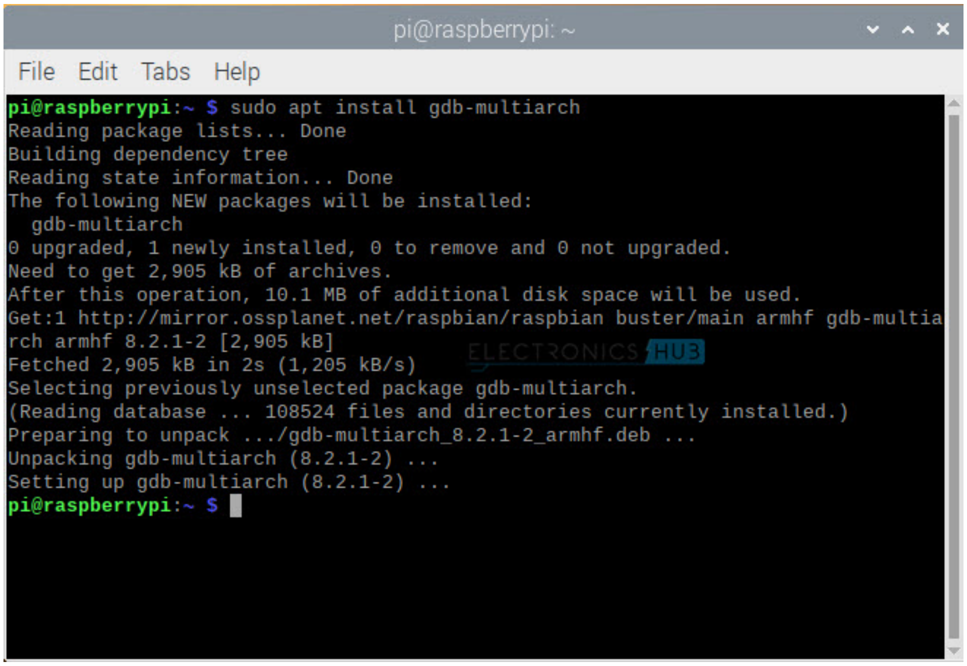
./configure –enable-ftdi –enable-sysfsgpio –enable-bcm2835gpio

I will take some time to build and install OpenOCD. Sit back and relax.

GDB

Next step is to install GDB. To install GDB Multi-Arch, use the following command:

sudo apt install gdb-multiarch



```
pi@raspberrypi:~ $ sudo apt install gdb-multiarch
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  gdb-multiarch
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 2,905 kB of archives.
After this operation, 10.1 MB of additional disk space will be used.
Get:1 http://mirror.ossplanet.net/raspbian/raspbian buster/main armhf gdb-multia
rch armhf 8.2.1-2 [2,905 kB]
Fetched 2,905 kB in 2s (1,205 kB/s)
Selecting previously unselected package gdb-multiarch.
(Reading database ... 108524 files and directories currently installed.)
Preparing to unpack .../gdb-multiarch_8.2.1-2_armhf.deb ...
Unpacking gdb-multiarch (8.2.1-2) ...
Setting up gdb-multiarch (8.2.1-2) ...
pi@raspberrypi:~ $
```

We will see how to debug using OpenOCD and GDB in the next section.

Wiring Raspberry Pi Pico and Raspberry Pi

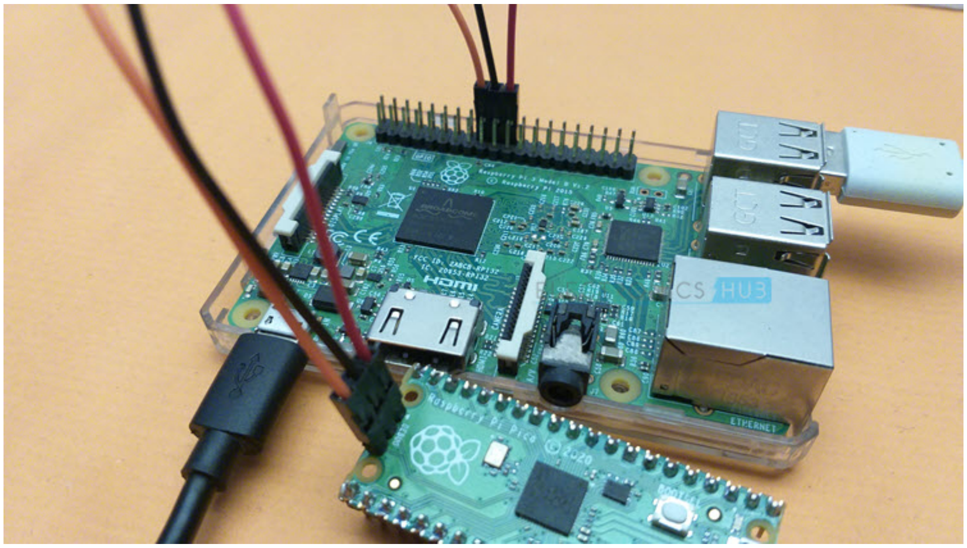
Before see how to program Raspberry Pi Pico using SWD, you have to first properly wire the Raspberry Pi Pico.

IMPORTANT NOTE: I do not know the exact reason but before making the connections, I had to shutdown Raspberry Pi to successfully program Raspberry Pi Pico using SWD. I read in Raspberry Pi Forum that both the Raspberry Pi and the target i.e., Raspberry Pi Pico in this case must be powered down before connecting SWD pins.

Since Raspberry Pi Pico is connected to Raspberry Pi through USB (to power it up), all I had to do was shutdown Raspberry Pi, make the SWD Connections and then power on Raspberry Pi.

The following table shows all the necessary connections between Raspberry Pi and Raspberry Pi Pico that you need to make.

Raspberry Pi Pico	Raspberry Pi
SWDIO	GPIO 24 (PIN 18)
SWD GND	GND (PIN 20)
SWCLK	GPIO 25 (PIN 22)

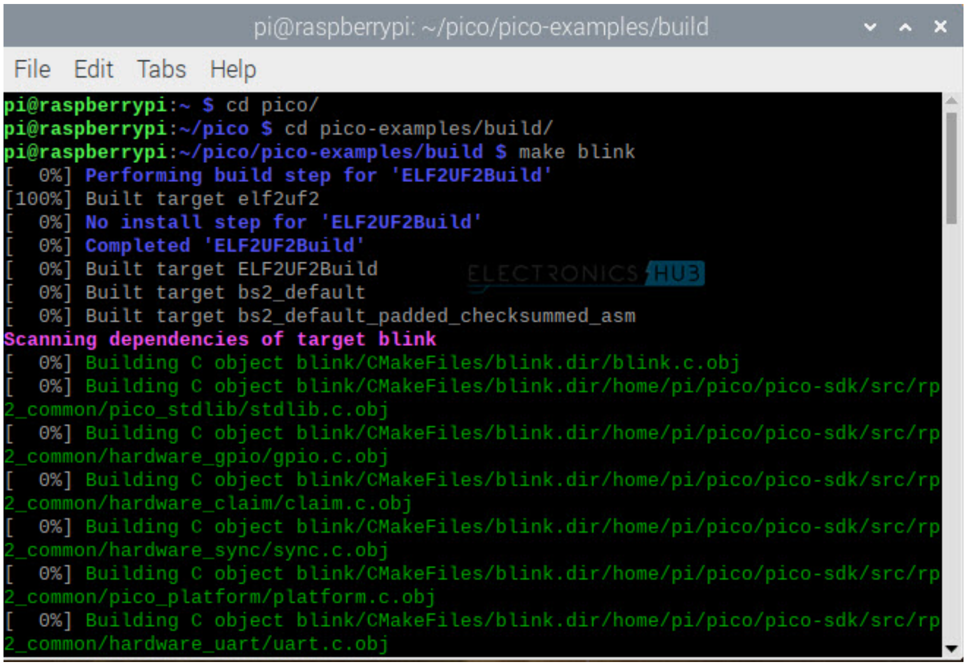


Programming Raspberry Pi Pico with SWD

Let us use the ‘Blink’ program as an example to understand how Raspberry Pi Pico SWD Programming works. If you remember in the ‘Programming Raspberry Pi Pico with C’ tutorial, we already built the Blink program, which resulted in a few target files.

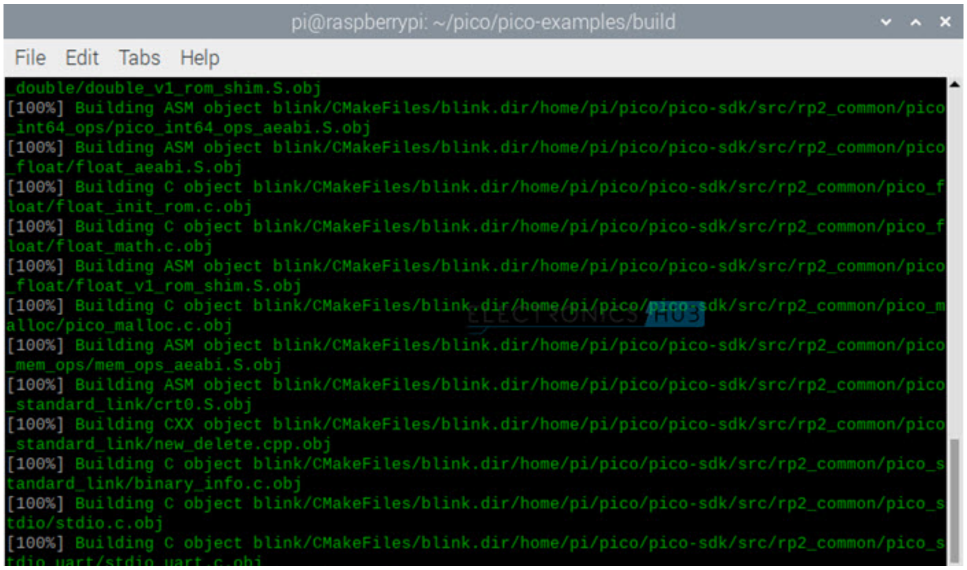
For drag-and-drop programming via USB, we used the .uf2 file. But OpenOCD uses .elf file to upload the program.

Use the following commands to Program Raspberry Pi Pico with SWD.

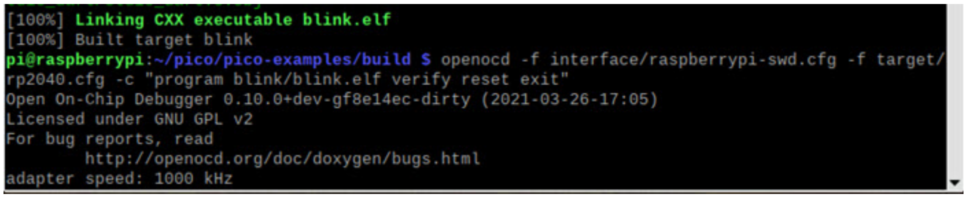


```
pi@raspberrypi:~ $ cd pico/
pi@raspberrypi:~/pico $ cd pico-examples/build/
pi@raspberrypi:~/pico/pico-examples/build $ make blink
[  0%] Performing build step for 'ELF2UF2Build'
[100%] Built target elf2uf2
[  0%] No install step for 'ELF2UF2Build'
[  0%] Completed 'ELF2UF2Build'
[  0%] Built target ELF2UF2Build
[  0%] Built target bs2_default
[  0%] Built target bs2_default_padded_checksummed_asm
Scanning dependencies of target blink
[  0%] Building C object blink/CMakeFiles/blink.dir/blink.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp
2_common/pico_stdlib/stdlib.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp
2_common/hardware_gpio/gpio.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp
2_common/hardware_claim/claim.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp
2_common/hardware_sync/sync.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp
2_common/pico_platform/platform.c.obj
[  0%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp
2_common/hardware_uart/uart.c.obj
```

openocd -f interface/raspberrypi-swd.cfg -f target/rp2040.cfg -c “program blink/blink.elf verify reset exit”

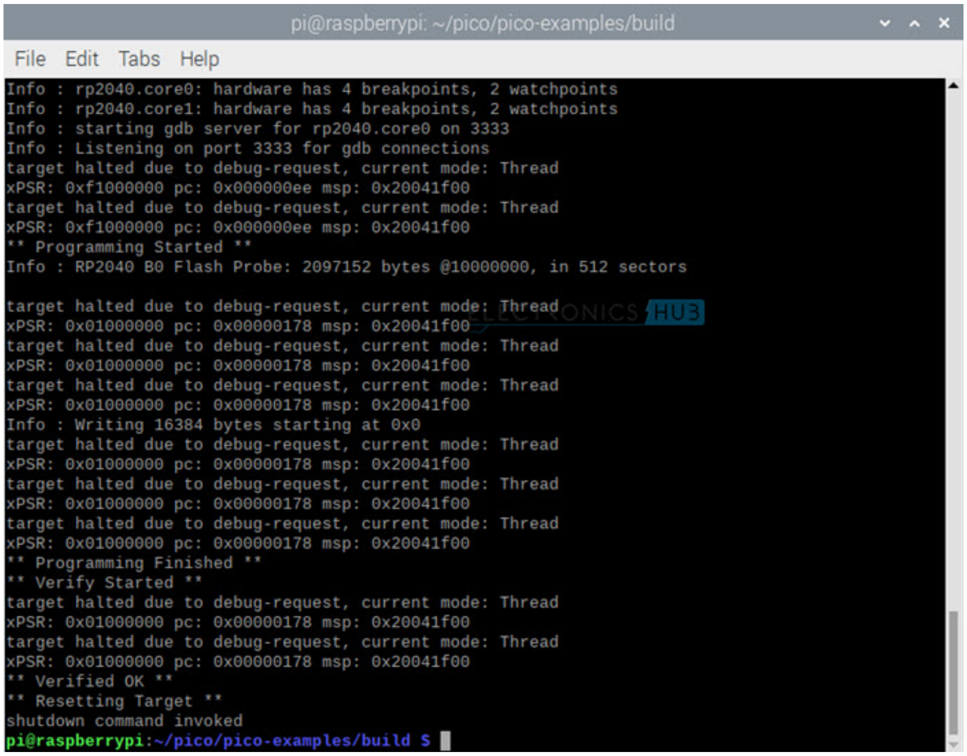


```
_double/double_v1_rom_shim.S.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico
_int64_ops/pico_int64_ops_aeabi.S.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico
_float/float_aeabi.S.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_f
loat/float_init_rom.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_f
loat/float_math.c.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico
_float/float_v1_rom_shim.S.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_m
alloc/pico_malloc.c.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico
_mem_ops/mem_ops_aeabi.S.obj
[100%] Building ASM object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico
_standard_link/crt0.S.obj
[100%] Building CXX object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico
_standard_link/new_delete.cpp.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_s
tandard_link/binary_info.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_s
tdio/stdio.c.obj
[100%] Building C object blink/CMakeFiles/blink.dir/home/pi/pico/pico-sdk/src/rp2_common/pico_s
tdio_uart/stdio_uart.c.obj
```



```
[100%] Linking CXX executable blink.elf
[100%] Built target blink
pi@raspberrypi:~/pico/pico-examples/build $ openocd -f interface/raspberrypi-swd.cfg -f target/
rp2040.cfg -c "program blink/blink.elf verify reset exit"
Open On-Chip Debugger 0.10.0+dev-gf8e14ec-dirty (2021-03-26-17:05)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 1000 kHz
```

The above command will invoke OpenOCD to program the blink.elf file to Raspberry Pi Pico, reset the board and exit the OpenOCD. If everything goes well, your terminal should display something like this and the LED on Raspberry Pi Pico should start to blink.



```
Info : rp2040.core0: hardware has 4 breakpoints, 2 watchpoints
Info : rp2040.core1: hardware has 4 breakpoints, 2 watchpoints
Info : starting gdb server for rp2040.core0 on 3333
Info : Listening on port 3333 for gdb connections
target halted due to debug-request, current mode: Thread
xPSR: 0xf1000000 pc: 0x000000ee msp: 0x20041f00
target halted due to debug-request, current mode: Thread
xPSR: 0xf1000000 pc: 0x000000ee msp: 0x20041f00
** Programming Started **
Info : RP2040 B0 Flash Probe: 2097152 bytes @10000000, in 512 sectors

target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
Info : Writing 16384 bytes starting at 0x0
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
** Programming Finished **
** Verify Started **
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000178 msp: 0x20041f00
** Verified OK **
** Resetting Target **
shutdown command invoked
pi@raspberrypi:~/pico/pico-examples/build $ 
```

Debugging Raspberry Pi Pico with SWD

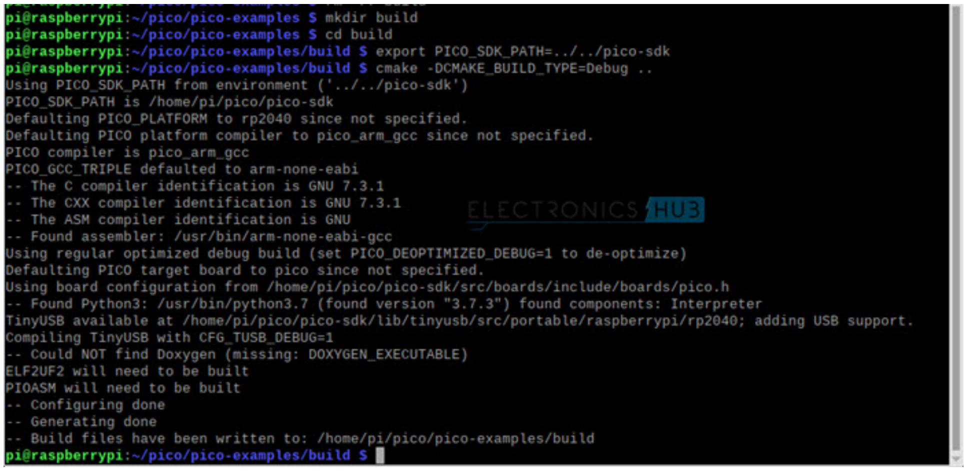
Let us now see how to debug the code suing SWD, OpenOCD and GDB. We already installed GDB in the previous step. Now, to include the debug related settings in the build files, you have use the CMake directive ‘ -DCMAKE_BUILD_TYPE=Debug ’.

But before that, you need to remove the ‘build’ directory from ‘pico-examples’ directory and create a new ‘build’ directory. Use the following commands to build examples with Debug information.

export PICO_SDK_PATH=../../pico-sdk

cmake -DCMAKE_BUILD_TYPE=Debug ..

```
pi@raspberrypi:~/pico/pico-examples $ rm -rf build
```



```
pi@raspberrypi:~/pico/pico-examples $ mkdir build
pi@raspberrypi:~/pico/pico-examples $ cd build
pi@raspberrypi:~/pico/pico-examples/build $ export PICO_SDK_PATH=../../pico-sdk
pi@raspberrypi:~/pico/pico-examples/build $ cmake -DCMAKE_BUILD_TYPE=Debug ..
Using PICO_SDK_PATH from environment ('../../pico-sdk')
PICO_SDK_PATH is /home/pi/pico/pico-sdk
Defaulting PICO_PLATFORM to rp2040 since not specified.
Defaulting PICO platform compiler to pico_arm_gcc since not specified.
PICO compiler is pico_arm_gcc
PICO_GCC_TRIPLE defaulted to arm-none-eabi
-- The C compiler identification is GNU 7.3.1
-- The CXX compiler identification is GNU 7.3.1
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/arm-none-eabi-gcc
Using regular optimized debug build (set PICO_DEOPTIMIZED_DEBUG=1 to de-optimize)
Defaulting PICO target board to pico since not specified.
Using board configuration from /home/pi/pico/pico-sdk/src/boards/include/boards/pico.h
-- Found Python3: /usr/bin/python3.7 (found version "3.7.3") found components: Interpreter
TinyUSB available at /home/pi/pico/pico-sdk/lib/tinyusb/src/portable/raspberrypi/rp2040; adding USB support.
Compiling TinyUSB with CFG_TUSB_DEBUG=1
-- Could NOT find Doxygen (missing: DOXYGEN_EXECUTABLE)
ELF2UF2 will need to be built
PIOASM will need to be built
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/pico/pico-examples/build
pi@raspberrypi:~/pico/pico-examples/build $
```

Let us use the 'hello_world' example and build for the serial variant. You cannot use USB based serial connection for SWD Debugging as the USB Device will be disconnected when stopping the processor core in debugging.

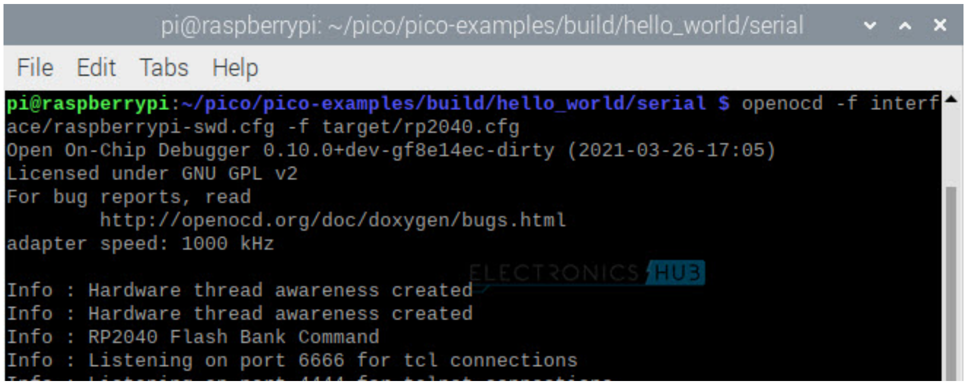
Also, the connections between Raspberry Pi Pico and Raspberry Pi to view the output of Raspberry Pi Pico's UART Serial Output on Raspberry Pi is as follows:

Raspberry Pi Pico	**Raspberry Pi**
GPIO 0 (UART0_TX)	GPIO 15 (UART_RX0) PIN 10
GPIO 1 (UART0_RX)	GPIO 14 (UART_TX0) PIN 8
GND	GND (Pin 14)

After making the connections, open the hello_world UART directory and build it.

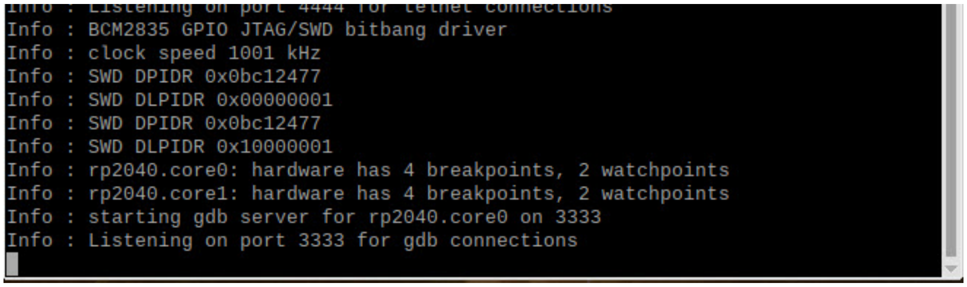
Use OpenOCD to open the GDB Server.

openocd -f interface/raspberrypi-swd.cfg -f target/rp2040.cfg



pi@raspberrypi: ~/pico/pico-examples/build/hello_world/serial
File Edit Tabs Help

```
pi@raspberrypi:~/pico/pico-examples/build/hello_world/serial $ openocd -f interf
ace/raspberrypi-swd.cfg -f target/rp2040.cfg
Open On-Chip Debugger 0.10.0+dev-gf8e14ec-dirty (2021-03-26-17:05)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 1000 kHz

Info : Hardware thread awareness created
Info : Hardware thread awareness created
Info : RP2040 Flash Bank Command
Info : Listening on port 6666 for tcl connections
```



```
Info : Listening on port 4444 for telnet connections
Info : BCM2835 GPIO JTAG/SWD bitbang driver
Info : clock speed 1001 kHz
Info : SWD DPIDR 0x0bc12477
Info : SWD DLPIDR 0x00000001
Info : SWD DPIDR 0x0bc12477
Info : SWD DLPIDR 0x10000001
Info : rp2040.core0: hardware has 4 breakpoints, 2 watchpoints
Info : rp2040.core1: hardware has 4 breakpoints, 2 watchpoints
Info : starting gdb server for rp2040.core0 on 3333
Info : Listening on port 3333 for gdb connections
```

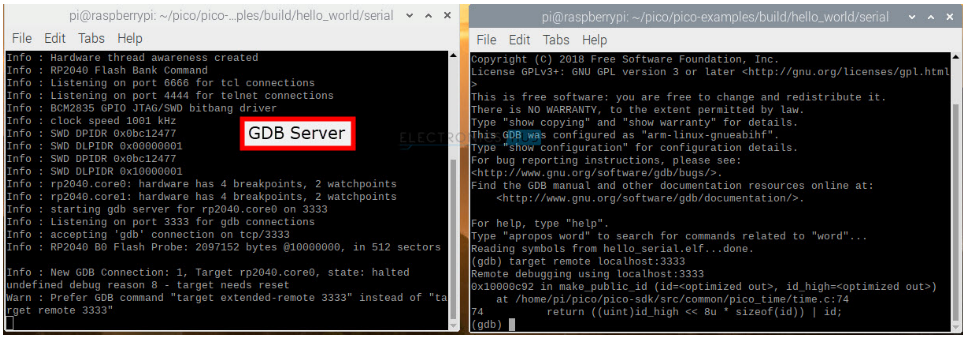
Keep this terminal as it is and open another terminal window and browse to UART Serial directory in the build directory.

cd pico/pico-examples/build/hello_world/serial/

Open GDB and connect to OpenOCD Server.

gdb-multiarch hello_serial.elf

(gdb) target remote localhost:3333



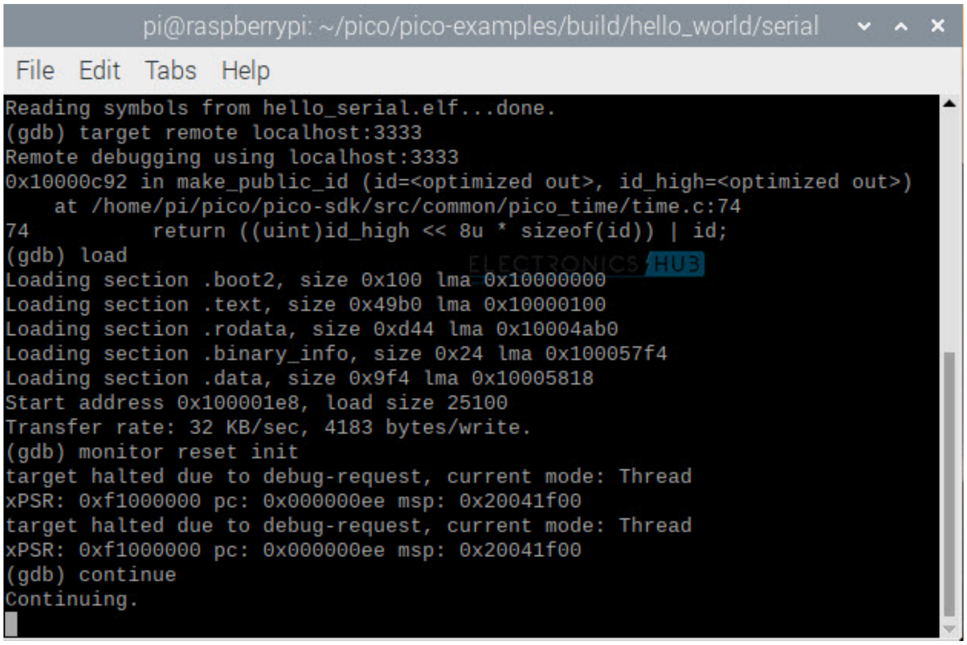
pi@raspberrypi: ~/pico/pico-...ples/build/hello_world/serial
File Edit Tabs Help
GDB Server
pi@raspberrypi: ~/pico/pico-examples/build/hello_world/serial
File Edit Tabs Help

```
Info : Hardware thread awareness created
Info : RP2040 Flash Bank Command
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : BCM2835 GPIO JTAG/SWD bitbang driver
Info : clock speed 1001 kHz
Info : SWD DPIDR 0x0bc12477
Info : SWD DLPIDR 0x00000001
Info : SWD DPIDR 0x0bc12477
Info : SWD DLPIDR 0x10000001
Info : rp2040.core0: hardware has 4 breakpoints, 2 watchpoints
Info : rp2040.core1: hardware has 4 breakpoints, 2 watchpoints
Info : starting gdb server for rp2040.core0 on 3333
Info : Listening on port 3333 for gdb connections
Info : accepting 'gdb' connection on tcp/3333
Info : RP2040 B0 Flash Probe: 2097152 bytes @10000000, in 512 sectors

Info : New GDB Connection: 1, Target rp2040.core0, state: halted
undefined debug reason 8 - target needs reset
Warn : Prefer GDB command "target extended-remote 3333" instead of "ta
rget remote 3333"
```

```
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html
>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_serial.elf...done.
(gdb) target remote localhost:3333
Remote debugging using localhost:3333
0x10000c92 in make_public_id (id=<optimized out>, id_high=<optimized out>)
    at /home/pi/pico/pico-sdk/src/common/pico_time/time.c:74
74          return ((uint)id_high << 8u * sizeof(id)) | id;
(gdb)
```

To load the program into the flash memory of Raspberry Pi Pico, use load command.

Start running the code.



pi@raspberrypi: ~/pico/pico-examples/build/hello_world/serial
File Edit Tabs Help

```
Reading symbols from hello_serial.elf...done.
(gdb) target remote localhost:3333
Remote debugging using localhost:3333
0x10000c92 in make_public_id (id=<optimized out>, id_high=<optimized out>)
    at /home/pi/pico/pico-sdk/src/common/pico_time/time.c:74
74          return ((uint)id_high << 8u * sizeof(id)) | id;
(gdb) load
Loading section .boot2, size 0x100 lma 0x10000000
Loading section .text, size 0x49b0 lma 0x10000100
Loading section .rodata, size 0xd44 lma 0x10004ab0
Loading section .binary_info, size 0x24 lma 0x100057f4
Loading section .data, size 0x9f4 lma 0x10005818
Start address 0x100001e8, load size 25100
Transfer rate: 32 KB/sec, 4183 bytes/write.
(gdb) monitor reset init
target halted due to debug-request, current mode: Thread
xPSR: 0xf1000000 pc: 0x000000ee msp: 0x20041f00
target halted due to debug-request, current mode: Thread
xPSR: 0xf1000000 pc: 0x000000ee msp: 0x20041f00
(gdb) continue
Continuing.
```

If you are familiar with GDB commands, you can explore them.

Conclusion

A complete tutorial on Programming and Debugging Raspberry Pi Pico with SWD. Learn how Raspberry Pi Pico SWD Interface works, the necessary connections between Raspberry Pi Pico and Raspberry Pi for SWD, program Raspberry Pi Pico with SWD, use GDB to debug Raspberry Pi Pico using SWD.

Related Posts:

- How to Program Raspberry Pi Pico with Visual Studio Code?
- Learn How to Create Your Own New Project for…
- How to Setup Raspberry Pi Pico Serial Programming?…
- Programming Raspberry Pi Pico with MicroPython | A…
- Programming Raspberry Pi Pico using C | Getting…
- Getting Started With Raspberry Pi Pico | An Introduction