Raspberry Pi Debug Probe

About the Debug Probe

Edit this on GitHub

The Raspberry Pi Debug Probe is a USB device that provides both a UART serial port and a standard Arm Serial Wire Debug (SWD) interface. The probe is designed for easy, solderless, plug-and-play debugging. It has the following features:

- USB to ARM Serial Wire Debug (SWD) port
- USB to UART bridge
- Compatible with the CMSIS-DAP standard
- Works with OpenOCD and other tools supporting CMSIS-DAP
- Open source, easily upgradeable firmware

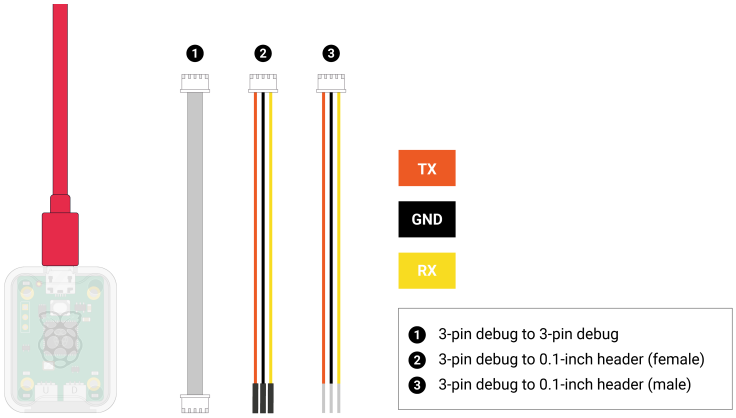
NOTE

For more information on the Raspberry Pi three-pin debug connector see the specification.

This makes it easy to use a Raspberry Pi Pico on non-Raspberry Pi platforms such as Windows, Mac, and “normal” Linux computers, where you don’t have a GPIO header to connect directly to the Pico’s serial UART or SWD port.

The Debug Probe

The probe operates at 3.3V nominal I/O voltage.



Included with the Debug Probe is a USB power cable and three debug cables:

- 3-pin JST connector to 3-pin JST connector cable
- 3-pin JST connector to 0.1-inch header (female)
- 3-pin JST connector to 0.1-inch header (male)

The two 0.1-inch header cables — intended for breadboard (male) or direct connection to a board with header pins (female) — are coloured as below:

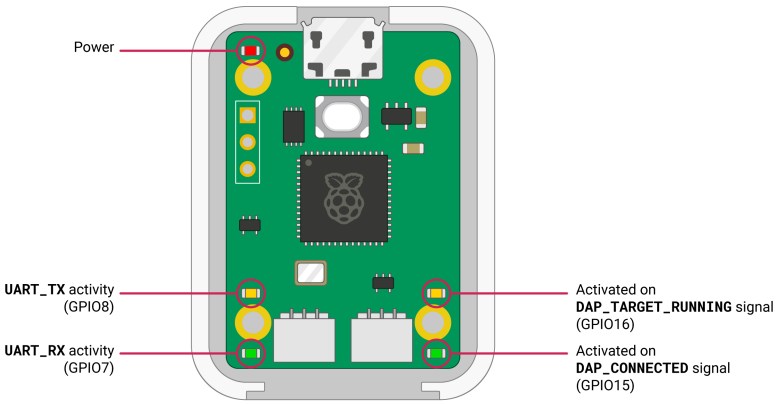
Orange
: TX/SC (Output from Probe)

Black
: GND

Yellow
: RX/SD (Input to Probe or I/O)

While the cable with 3-pin JST connectors is intended to be used with the standard 3-pin connector which newer Raspberry Pi boards use for the SWD debug port and UART connectors.

The Debug Probe has five LEDs, a red LED to indicate power, and four more activity indicator LEDs

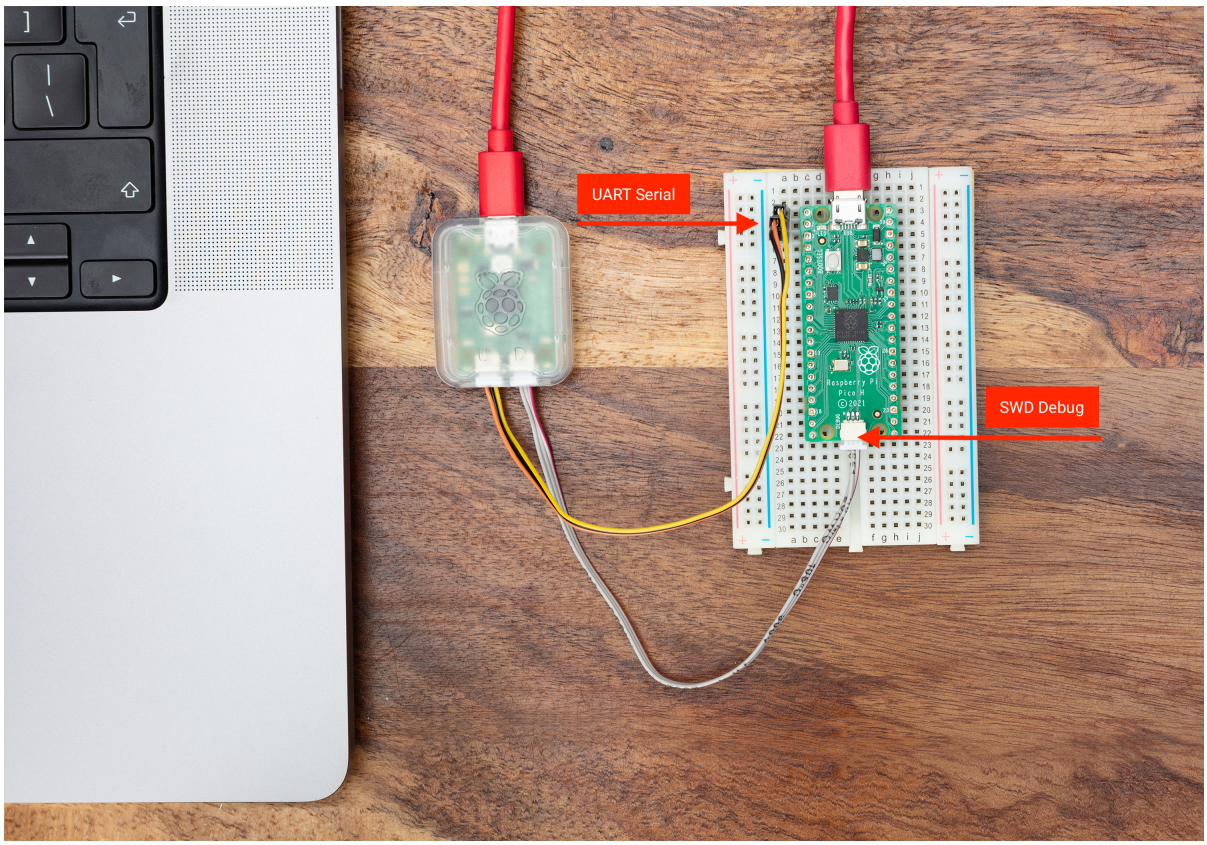


NOTE

OpenOCD just switches both DAP LEDs on when the target is connected, and turns them off when it calls `DAP_DISCONNECT`.

Getting started

Edit this on GitHub



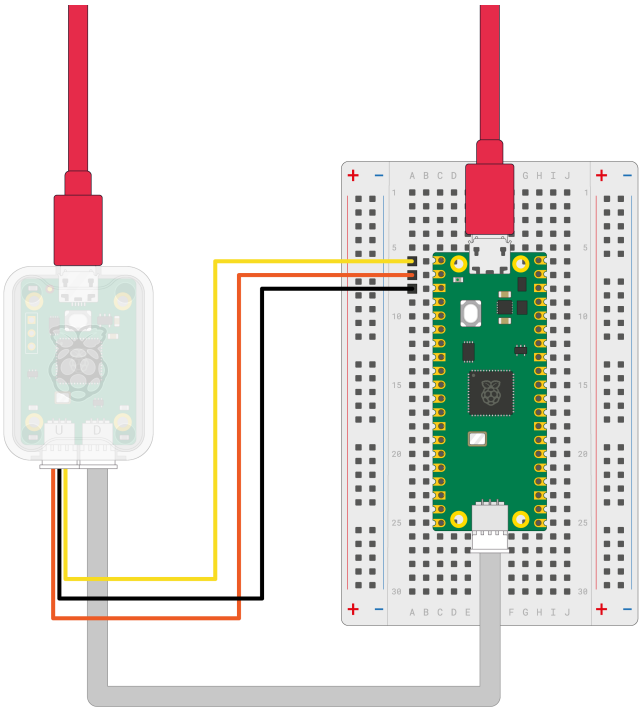
Depending on your setup, there are several ways to wire the Debug Probe to a Raspberry Pi Pico. Below, we connect the Debug Probe to a Raspberry Pi Pico H which has the newer three-pin JST connector for SWD.

Here we have connected:

- The Debug Probe "D" connector to Pico H SWD JST connector
- The Debug Probe "U" connector has the three-pin JST connector to 0.1-inch header (male)
    - Debug Probe RX connected to Pico H TX pin
    - Debug Probe TX connected to Pico H RX pin
    - Debug Probe GND connected to Pico H GND pin

NOTE

If you have an "original" Raspberry Pi Pico, or Pico W, without a JST connector, you can still connect it to a Debug Probe. To do so, solder a male connector to the SWDCLK, SWDIO and GND header pins on the board, and connect them to the Debug Probe "D" connector using the alternate 3-pin JST connector to 0.1-inch header (female) cable included with the Debug Probe.



Installing Tools

Edit this on GitHub

Before we get started we need to install some tools.

Installing OpenOCD

You need to install OpenOCD.

NOTE

SMP support for debugging on both RP2040 cores is not yet available in the release version of `openocd`. However, support is available in the `rp2040` branch and will be enabled if you build from source.

Linux (and Raspberry Pi)

On Raspberry Pi OS you can install `openocd` directly from the command line.

```
$ sudo apt install openocd
```

You need to be running OpenOCD version 0.11.0 or 0.12.0 to have support for the Debug Probe. If you're not running Raspberry Pi OS, or your distribution installs an older

version, or require SMP support, you can build and install `openocd` from source. Start by installing needed dependencies,

```
$ sudo apt install automake autoconf build-essential texinfo libtool libftdi-de
v libusb-1.0-0-dev
```

and then build OpenOCD.

```
$ git clone https://github.com/raspberrypi/openocd.git --branch rp2040 --depth=
1 --no-single-branch
$ cd openocd
$ ./bootstrap
$ ./configure
$ make -j4
$ sudo make install
```

NOTE

If you are building on a Raspberry Pi you can also pass `--enable-sysfsgpio --enable-bcm2835gpio` when you `./configure` to allow bit-banging SWD via the GPIO pins. See Chapters 5 and 6 of Getting Started with Raspberry Pi Pico for more details.

macOS

Install Homebrew if needed,

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/instal
l/master/install.sh)"
```

and then OpenOCD.

```
$ brew install open-ocd
```

Alternatively you can install needed dependencies,

```
$ brew install libtool automake libusb wget pkg-config gcc texinfo
```

and build OpenOCD from source.

```
$ cd ~/pico
$ git clone https://github.com/raspberrypi/openocd.git --branch rp2040 --depth=
1
$ cd openocd
$ export PATH="/usr/local/opt/texinfo/bin:$PATH"
$ ./bootstrap
$ ./configure --disable-werror
```

```
$ make -j4
$ sudo make install
```

NOTE

If you are using a newer Arm-based Mac, the path to `texinfo` will be `/opt/homebrew/opt/texinfo/bin`.

NOTE

Unfortunately `disable-werror` is needed because not everything compiles cleanly on macOS

MS Windows

A standalone OpenOCD Windows package is available for download. Alternatively it will be installed as part of our Pico setup for Windows installer package.

But, if you want to, you can also build OpenOCD from source using MSYS2. Go ahead and download and run the MSYS2 installer, and then update the package database and core system packages,

```
$ pacman -Syu
```

NOTE

If MSYS2 closes, start it again (making sure you select the 64-bit version) and run `pacman -Su` to finish the update.

Install required dependencies,

```
$ pacman -S mingw-w64-x86_64-toolchain git make libtool pkg-config autoconf aut
omake texinfo
mingw-w64-x86_64-libusb
```

Pick all when installing the `mingw-w64-x86_64` toolchain by pressing ENTER.

Close MSYS2 and reopen the 64-bit version to make sure the environment picks up GCC,

```
$ git clone https://github.com/raspberrypi/openocd.git --branch rp2040 --depth=
1
$ cd openocd
$ ./bootstrap
$ ./configure --disable-werror
$ make -j4
```

NOTE

Unfortunately `disable-werror` is needed because not everything compiles cleanly on Windows

NOTE

Manual installation of `openocd` on MS Windows is not recommended.

Installing GDB

We also need to install the GNU debugger (GDB).

Linux (and Raspberry Pi)

Install `gdb-multiarch`.

```
$ sudo apt install gdb-multiarch
```

macOS

Install Homebrew if needed,

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/instal
l/master/install.sh)"
```

then install `gdb`.

```
$ brew install gdb
```

NOTE

You can safely ignore the request for "special privileges" messages on installation.

IMPORTANT

If you have an Arm-based (M1-based) Mac `gdb` is not available via Homebrew, instead you will either have to install it from source or use `lldb` instead of `gdb`. At this time there is no official support from the developers for running GDB on Arm-based Macs. Support for GDB can be found on the GDB mailing list on Sourceware.org. `lldb` is installed as part of the Xcode Command Line Tools.

MS Windows

GDB is available as part of our Pico setup for Windows installer. It is also included in the Arm GNU Toolchain Downloads.

Alternatively information about manual installation can be found in Chapter 9 and Appendix A of our Getting Started with Raspberry Pi Pico book.

NOTE

Manual installation of GDB on Windows is not recommended.

Serial Wire Debug (SWD)

Edit this on GitHub

Serial Wire Debug (SWD) is a two-pin interface (SWDIO and SWCLK) alternative to the JTAG four- or five-pin debugging interface standard.

Uploading new programs to your Pico

The Pico Debug Probe will let you load binaries via the SWD port and OpenOCD: you will not need to unplug, and then push-and-hold, the BOOTSEL button every time you push a new binary to your Pico. Using the Debug Probe uploading new binaries is an entirely hands off affair.

Once you have built a binary:

```
$  sudo openocd -f interface/cmsis-dap.cfg -f target/rp2040.cfg -c "adapter spe
ed 5000" -c "program blink.elf verify reset exit"
```

NOTE

When you use the Debug Probe to upload a binary the ELF version of the file is used, not the UF2 file that you would use when you drag-and-drop.

Debugging with SWD

It'll also let you use `openocd` in server mode, and connect GDB, which gives you break points and "proper" debugging.

IMPORTANT

To allow debugging, you must build your binaries as `Debug` rather than `Release` build type, e.g.

```
$ cd ~/pico/pico-examples/
$ rm -rf build
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
```

```
$ cd blink
$ make -j4
```

In a debug build you will get more information when you run it under the debugger, as the compiler builds your program with the information to tell GDB what your program is doing.

See Chapter 6 of Getting started with Raspberry Pi Pico for more information.

Run OpenOCD 'server' ready to attach GDB to:

```
$ sudo openocd -f interface/cmsis-dap.cfg -f target/rp2040.cfg -c "adapter spee
d 5000"
```

Then open a second terminal window, switch to the directory containing your built binary, and start GDB.

```
$ gdb blink.elf
> target remote localhost:3333
> monitor reset init
> continue
```

NOTE

On non-Raspberry Pi Linux platforms you should invoke GDB by using `gdb-multiarch blink.elf`.

NOTE

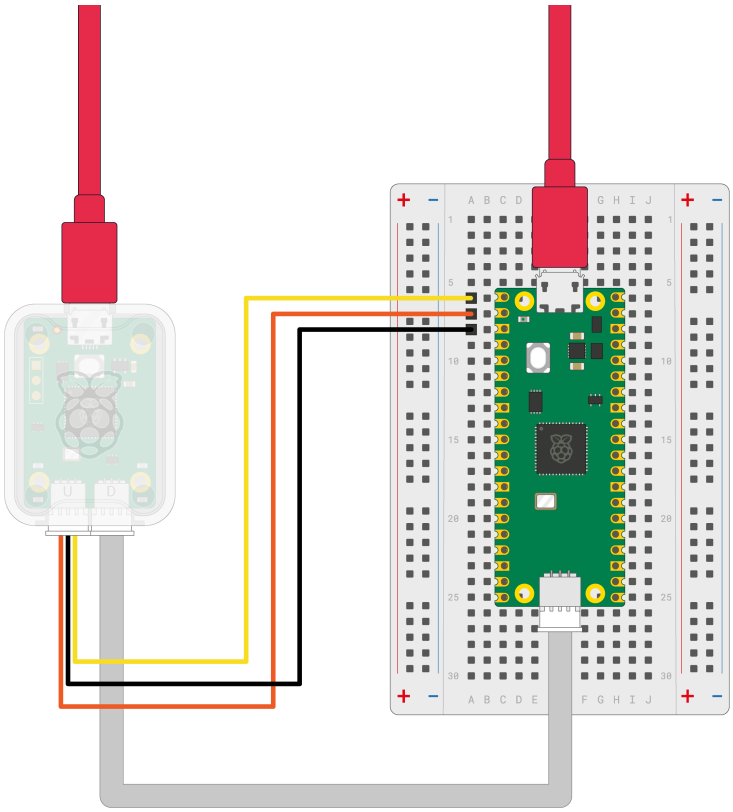
If you are on an Arm-based (M1) Mac without `gdb`, you can make use of `lldb`, which is installed along with the XCode Command Line Tools. The syntax used by `lldb` is slightly different to `gdb`.

```
$ lldb blink.elf
(lldb) gdb-remote 3333
(lldb) process plugin packet monitor reset
(lldb) continue
```

Serial connections

Edit this on GitHub

Ensure that the Debug Probe is connected to the UART pins of your Raspberry Pi Pico.



The default pins for Raspberry Pi Pico UART0 are as follows:

Default UART0	Physical Pin	GPIO Pin
GND	3	N/A
UART0_TX	1	GP0
UART0_RX	2	GP1

Once connected, traffic over the Raspberry Pi Pico's UART will be relayed to your computer by the Debug Probe and exposed as a CDC UART. On a Raspberry Pi this will show up as `/dev/ttyACM0`; on other platforms this serial port will show up differently (e.g. on macOS it will appear as `/dev/cu.usbmodemXXXX`).

If you have not already done so you should install minicom:

```
$ sudo apt install minicom
```

and open the serial port:

```
$ minicom -b 115200 -o -D /dev/ttyACM0
```

TIP

To exit `minicom`, use CTRL-A followed by X.

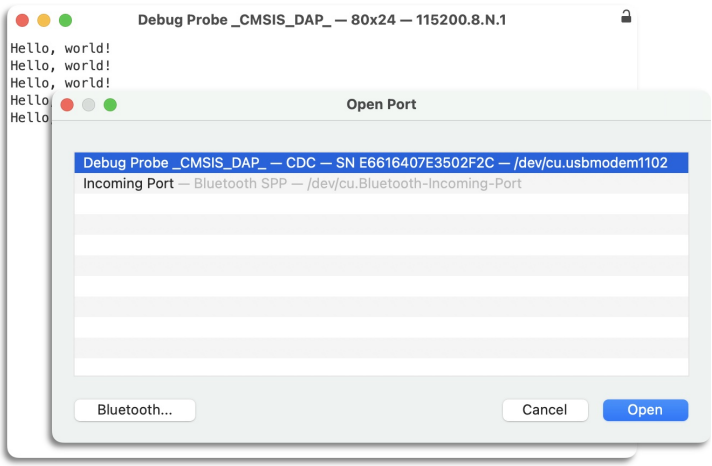
To test serial communication you can build and upload the "Hello World" example application.

Change directory into the `hello_world` directory inside the `pico-examples` tree, and run `make`. Afterwards, you can upload it to your Raspberry Pi Pico using `openocd`. For a full walkthrough of building the `hello_serial` example program, see Chapter 4 of Getting started with Raspberry Pi Pico.

```
$ cd pico-examples
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake ..
$ cd hello_world/serial
$ make -j4
$ sudo openocd -f interface/cmsis-dap.cfg -f target/rp2040.cfg -c "adapter spee
d 5000" -c "program hello_serial.elf verify reset exit"
$ minicom -b 115200 -o -D /dev/ttyACM0
```

On opening `minicom` you should see "Hello, world!" printed to the console.

For terminal programs that support it, a description of the USB serial UART is advertised in the USB device description.



The unique serial number in this description means that on Windows your COM port numbering is "sticky" per device, and will allow you to write `udev` rules to associate a named device node with a particular Debug Probe.

Updating the firmware on the Debug Probe

Edit this on GitHub

Edit this on GitHub

NOTE

There is currently no newer version of the firmware. The firmware running on your Debug Probe is the latest available. If you have accidentally overwritten the firmware on your Debug Probe, the latest release of the firmware can be found on Github.

From time to time you may need to update the Debug Probe firmware. New firmware for the debug probe will be made available as a UF2 file distributed by Raspberry Pi.

Pinch to remove the top of the Debug Probe enclosure, then push and hold the BOOTSEL button as you plug the Debug Probe into your computer. This will mount an RPI-RP2 volume on your desktop. Drag-and-drop the firmware UF2 onto the RPI-RP2 volume. The firmware will be copied to the Debug Probe and the volume will dismount.

Your Debug Probe will reboot. You are now running an updated version of the Debug Probe firmware.

Schematics

Edit this on GitHub

Schematics and mechanical drawing of the Debug Probe are available:

- Schematics (PDF)
- Mechanical Diagram (PDF)

The test point (TP) shown on the schematics are located as shown in the diagram below.