

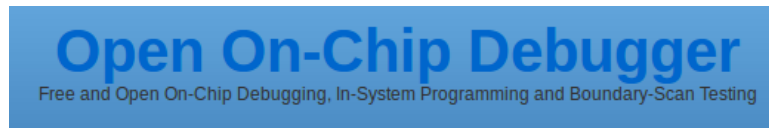
# OpenOCD for RISC-V

2019-10-14

## 1 About

OpenOCD is a software that provides on-chip debugging, in-system programming and boundary-scan testing tools.

The official website is [openocd.org](https://openocd.org)



A fork with RISC-V support is available on <https://github.com/riscv/riscv-openocd>. It supports versions 0.11 and 0.13 of the official RISC-V external debug specification.

OpenOCD is free software and is licensed under the **GNU General Public License v2.0**

This document presents a way of using OpenOCD to debug a RISC-V platform using a JTAG connection.

## 2 Position in the debug system

OpenOCD can be used as a translator between GDB and a RISC-V platform, as can be seen in the RISC-V external debug specification.

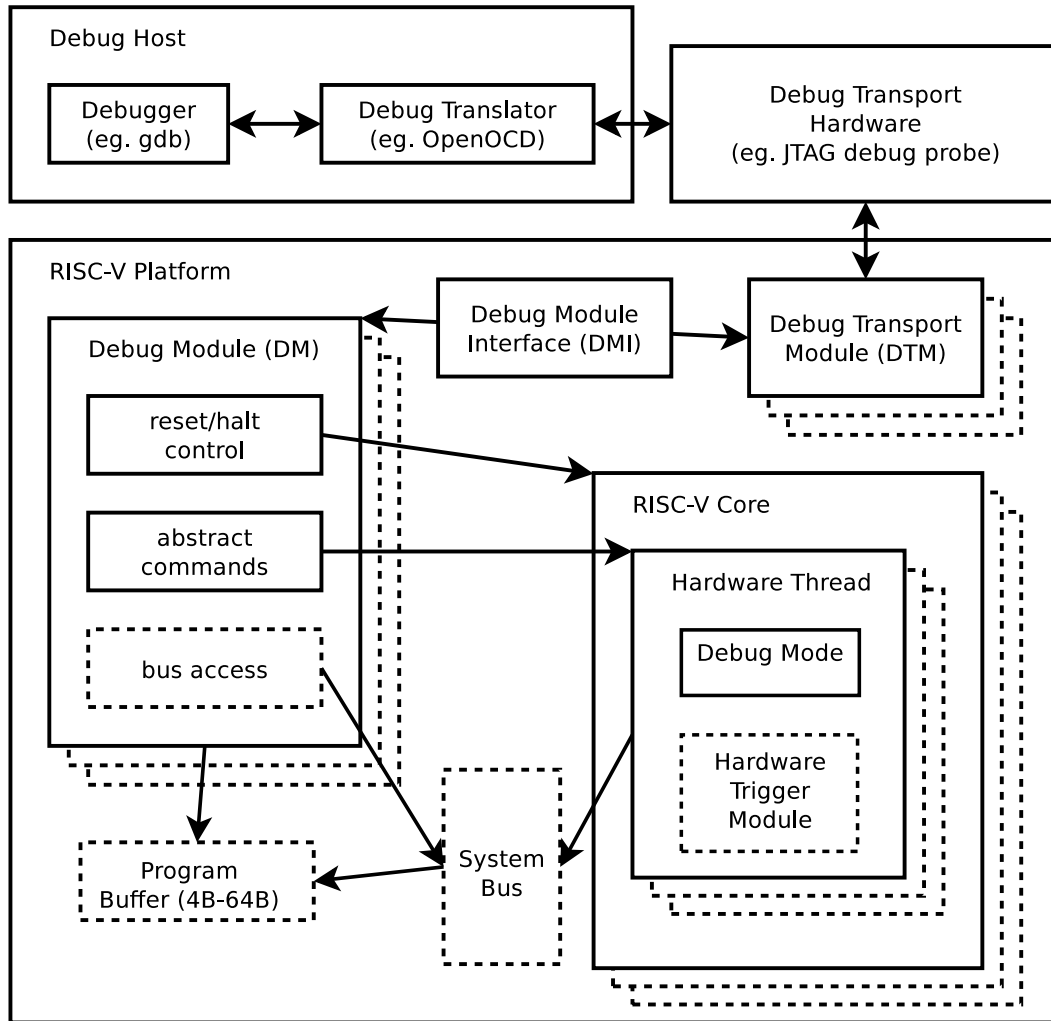


Figure 1: RISC-V Debug System Overview

OpenOCD can connect to GDB in two ways: with a TCP/IP socket or with pipes (stdin/stdout).

More information here: <http://openocd.org/doc/html/GDB-and-OpenOCD.html>

A debug adapter is needed to connect OpenOCD to the RISC-V platform. This document focuses on using JTAG transport hardware.

## 3 Config file

<http://openocd.org/doc/html/OpenOCD-Project-Setup.html>

A typical usage of OpenOCD is:

```
$ openocd -f config_file.cfg
```

A config file contains commands that OpenOCD will execute using its Jim-Tcl interpreter.

A single file can be broken into several ones. The command line call would then list every file in the correct order:

```
$ openocd -f config_file_1.cfg config_file_2.cfg config_file_3.cfg
```

It is also possible to launch a single command:

```
$ openocd -c "a command"
```

To have more information about what OpenOCD is doing, use the -d option.

```
$ openocd -f config_file.cfg -d
```

### 3.1 Interface

<http://openocd.org/doc/html/Debug-Adapter-Configuration.html>

The interface configuration tells OpenOCD how to use the transport hardware.

Config files for a lot of debug adapters can be found where OpenOCD was installed. The path should be `<build>/share/openocd/scripts/interface`.

Otherwise, refer to examples and the official documentation to see how to use the various interface drivers.

### 3.2 TAP declaration

<http://openocd.org/doc/html/TAP-Declaration.html>

A device with a JTAG interface means the device has a Test Access Port (TAP). You need to set up the active TAPs of the device by declaring them inside a configuration file.

Typically, this is achieved in a few lines:

```
set _CHIPNAME riscv
jtag newtap $_CHIPNAME cpu -irlen 5 -expected-id 0x12345678
```

The -irlen parameter is the length in bits of the TAP instruction register and is mandatory.

You can give the IDCODE which is expected in the TAP, but this is optionnal.

### 3.3 CPU configuration

<http://openocd.org/doc/html/CPU-Configuration.html>

This step gives information about the debug target.

```
set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -chain-position $_TARGETNAME
```

The target refers to a TAP with a *dotted.name* and its type is precised (riscv for RISC-V).

### 3.4 Starting the debug session

<http://openocd.org/doc/html/Server-Configuration.html>

<http://openocd.org/doc/html/General-Commands.html>

An end of the config file could be:

```
init
halt
```

**init** end the configuration stages and enters the run stage.

**halt** sends a halt request to the target.

After these commands, the target is halted and OpenOCD is waiting for a GDB connection.

### 3.5 Other useful commands

<http://openocd.org/doc/html/index.html>

All commands are detailed in the official documentation. From experience, the following commands have been useful.

Set up a maximum speed for the JTAG interface:

```
adapter_khz 1000
```

Enable error reports from GDB:

```
gdb_report_data_abort enable
gdb_report_register_access_error enable
```

Display the TAPs in the scan chain and their status:

```
scan_chain
```

Prefer to use the System Bus Access to access memory rather than the Program Buffer:

```
riscv set_prefer_sba on
```

By default, OpenOCD will only listen on the loopback network interface (localhost). If the network environment is safe, you can cover all available interfaces with:

```
bindto 0.0.0.0
```

### 3.6 RISC-V commands

Commands specific to the RISC-V architecture can be found in this section of the documentation:

[http://openocd.org/doc/html/Architecture-and-Core-Commands.html#RISC\\_002dV-Architecture](http://openocd.org/doc/html/Architecture-and-Core-Commands.html#RISC_002dV-Architecture)

## 4 Config file examples

### 4.1 remote\_bitbang interface

<https://github.com/riscv/riscv-isa-sim#debugging-with-gdb>

The RISC-V ISA simulator Spike can be interfaced with OpenOCD and GDB. The corresponding config file is given in the project *README.md*

```
interface remote_bitbang
remote_bitbang_host localhost
remote_bitbang_port 9824

set _CHIPNAME riscv
jtag newtap $_CHIPNAME cpu -irlen 5 -expected-id 0x10e31913

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -chain-position $_TARGETNAME

gdb_report_data_abort enable

init
halt
```

The interface driver is **remote\_bitbang**. It connects to a remote process over a socket connection and performs bitbang requests. This driver is useful for debugging a simulated processor.

You need to specify a port number and a hostname to the driver.

### 4.2 sysfsgpio interface

[http://openocd.org/doc/doxygen/html/sysfsgpio\\_8c.html#details](http://openocd.org/doc/doxygen/html/sysfsgpio_8c.html#details)

The **sysfsgpio** interface driver implements a bitbang JTAG interface using system GPIOs.

```
adapter_khz 10000

interface sysfsgpio

# gpio numbers: tck tms tdi tdo
sysfsgpio_jtag_nums 507 508 509 510

# gpio number: trst
sysfsgpio_trst_num 511

set _CHIPNAME riscv
jtag newtap $_CHIPNAME cpu -irlen 5

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -chain-position $_TARGETNAME

gdb_report_data_abort enable
bindto 0.0.0.0

init
halt
```

### 4.3 ftdi interface and several TAPs

<https://github.com/pulp-platform/pulpiissimo>

The pulpiissimo project provides several examples of using the **ftdi** interface driver and setting up a scan chain with two TAPs.

An example for the Olimex ARM-USB-OCD debug adapter is:

```
adapter_khz 1000

# Olimex ARM-USB-OCD-H
interface ftdi
ftdi_device_desc "Olimex OpenOCD JTAG ARM-USB-OCD-H"
ftdi_vid_pid 0x15ba 0x002b

ftdi_layout_init 0x0908 0x0b1b
ftdi_layout_signal nSRST -oe 0x0200
ftdi_layout_signal nTRST -data 0x0100
ftdi_layout_signal LED -data 0x0800

set _CHIPNAME riscv

jtag newtap $_CHIPNAME unknown0 -irlen 5 -expected-id 0x10102001
jtag newtap $_CHIPNAME cpu -irlen 5 -expected-id 0x249511C3

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -chain-position $_TARGETNAME -coreid 0x3e0

gdb_report_data_abort enable
gdb_report_register_access_error enable

riscv set_reset_timeout_sec 120
riscv set_command_timeout_sec 120

# prefer to use sba for system bus access
riscv set_prefer_sba on

# dump jtag chain
scan_chain

init
halt
echo "Ready for Remote Connections"
```

The interface part of the configuration can also be found in the scripts provided by OpenOCD in your build directory: **<build>/share/openocd/scripts/interface/ftdi/olimax-arm-usb-ocd.cfg**

There are two TAPs in the scan chain, but only one is compatible with OpenOCD. Both are described in the config file, but only one is linked with a target.