

Building riscv-openocd

Armand Bouvier-Neveu – Thales Research and technology France

2019-09-12

1 About

OpenOCD is a free and open-source software distributed under the GPL-2.0 license. It provides on-chip programming and debugging support with a layered architecture of JTAG interface and TAP support.

A fork with RISC-V support has been developped and is available on GitHub at <https://github.com/riscv/riscv-openocd>

This guide aims at providing help about building OpenOCD for various platforms. More details, guides and manuals about OpenOCD can be found in the repository's README.md

2 Dependencies

2.1 Compiler

GCC or Clang is currently required to build OpenOCD. On a Linux workstation, GCC should be available by default or in the distribution's repositories.

For other architectures, cross-compiling is possible. Linaro provides pre-built GNU cross-toolchain binary archives at <https://releases.linaro.org/components/toolchain/binaries/>

The `uname` command can help finding the architecture of a target:

```
$ uname --m
```

2.2 Tools

Other tools are also needed:

- make
- libtool
- pkg-config ≥ 0.23
- autoconf ≥ 2.64
- automake ≥ 1.14
- texinfo

On Ubuntu, ensure that everything is installed with:

```
$ sudo apt install make libtool pkg-config autoconf automake texinfo
```

2.3 Libraries

Depending on the compilation options, some libraries might be needed.

2.3.1 libusb

If the target is the workstation (i.e. not cross-compiling), it is likely that libusb is already installed or included in the distribution's repositories. For Ubuntu:

```
$ sudo apt install libusb-1.0-0-dev
```

To cross-compile, or to use the latest version from source:

- Obtain the source. The latest tarball is available on <https://libusb.info>
- Extract the source

```
$ tar -zxvf libusb-X.X.XX.tar.bz2
```

- Make a build directory somewhere, inside the libusb root directory for example

```
$ cd libusb-X.X.XX  
$ mkdir build
```

- If cross-compiling, set the compiler. Ensure that the given compiler is in the \$PATH

```
$ export CC=arm-linux-gnueabi-gcc
```

- Use the *configure* script. You can see all the options with:

```
$ ./configure --help
```

- For example, these options set the install directory, set the host for cross-compiling and disable udev:

```
$ ./configure --prefix=<build dir> --exec-prefix=<build dir> --host=arm-linux-  
gnueabi --disable-udev
```

- Compile and install files

```
$ make  
$ make install
```

- The produced library files are in the *build* directory. They need to be added to the compiler to be used.

2.3.2 libftdi

If the target is the workstation (i.e. not cross-compiling), it is likely that libusb is already installed or included in the distribution's repositories. For Ubuntu:

```
$ sudo apt install libftdi1-dev
```

To cross-compile or to get the latest version, the source is available at <https://www.intra2net.com/en/developer/libftdi/download.php>

- Some tools are needed. On Ubuntu:

```
$ sudo apt install build-essential cmake
```

- Libusb is also required.
- Download and extract the tarball
- Inside the directory, make a build directory

```
$ cd libftdiX-X.X  
$ mkdir build  
$ cd build
```

- Create a cmake toolchain file and fill it with the target informations

toolchain-arm-linux-gnueabihf.cmake

```
SET(CMAKE_SYSTEM_NAME arm-linux-gnueabihf)
SET(CMAKE_C_COMPILER arm-linux-gnueabihf-gcc)
SET(CMAKE_CXX_COMPILER arm-linux-gnueabihf-g++)
SET(LIBUSB_LIBRARIES <...>/gcc-linaro-7.4.1-2019.02-x86_64_arm-linux-gnueabihf/lib/
  libusb-1.0.so.0)
SET(LIBUSB_INCLUDE_DIR <...>/gcc-linaro-7.4.1-2019.02-x86_64_arm-linux-gnueabihf/
  include/libusb-1.0)
```

- Make sure the compilers are in the \$PATH and use cmake. The install directory can be controlled with the prefix options.

```
$ cmake -DCMAKE_TOOLCHAIN_FILE=<cmake toolchain file> -DCMAKE_INSTALL_PREFIX=$(pwd)
  -DLIBFTDI_LIBRARY_DIRS=$(pwd) ..
```

- Compile and install the files

```
$ make
$ make install
```

- The produced library files are in the *build* directory. They needed to be added to the compiler to be used.

3 OpenOCD

- Download the sources

```
$ git clone https://github.com/riscv/riscv-openocd
$ cd riscv-openocd
```

- Prepare a *build* directory

```
$ mkdir build
```

- If the aim is to use OpenOCD for PULP applications, a small patch is required. Find `src/target/riscv/riscv.h` and modify the following definition :

```
-#define RISC_V_MAX_HARTS 32
+#define RISC_V_MAX_HARTS 1024
```

- If cross-compiling, set the compiler. Ensure that the given compiler is in the \$PATH

```
$ export CC=arm-linux-gnueabihf-gcc
```

- Launch the bootstrap script

```
$ ./bootstrap
```

- Review the possible configuration options

```
$ ./configure --help
```

- In particular, `-enable-sysfsgpio`, `-enable-ftdi` and `-enable-remote-bitbang` must be added to use the corresponding features.
- The `-host=` option must be provided to cross-compile
- The `-prefix=` and `-exec-prefix=` options control the installation directory

- Use configure with all the chosen options

```
$ ./configure --enable-sysfsgpio --host=arm-linux-gnueabihf --prefix=<build dir> --  
exec-prefix=<build dir>
```

- Some problems may be encountered with an aarch64-linux-gnu host. Please review <https://github.com/riscv/riscv-openocd/issues/17>
- Compile and install files

```
$ make  
$ make install
```

A single *openocd* binary should be available in *build*