Freedom Studio Build Settings

Version 1.0

Freedom Studio Build Settings

Proprietary Notice

Release Information

Version	Date	Changes
V1.0	August 21, 2019	• Initial release

Contents

Chapter 1

Freedom Studio Build Settings

1.1 Introduction

SiFive's integrated IDE is an Eclipse-based tool called Freedom Studio which includes a pre-built RISCV GCC Toolchain, OpenOCD for debug, and the freedom-e-sdk repo which contains software examples and an API layer called freedom-metal.

The compile and link options within Freedom Studio reside in Makefiles which are included when a new project is created. This applies to all Freedom Studio versions released in 2019 and beyond. Versions released in 2018 and earlier have build settings buried in menus and are no longer supported.

1.2 How to Create a New Project

To create a new project, simply click the **New Project** button, or use the **SiFive Tools** menu option:

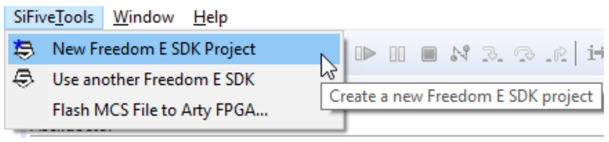
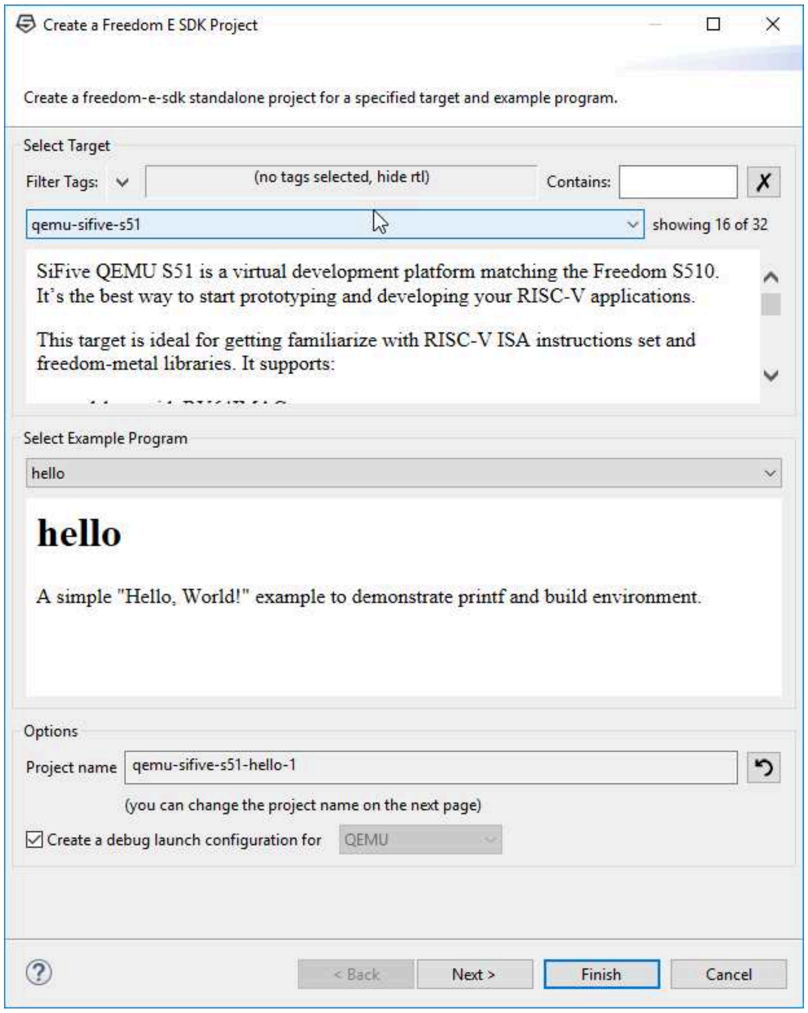

SiFiveTools
Window
Help
New Freedom E SDK Project
Use another Freedom E SDK
Flash MCS File to Arty FPGA...
Create a new Freedom E SDK project

Figure 1: New Project Menu Option

1.2.1 New Project Selections

This will open up a dialogue box where the hardware target and the example program can be selected. Note that the default target options only include SiFive's development boards and QEMU, but this can be changed to include custom designs.


Create a Freedom E SDK Project

Create a freedom-e-sdk standalone project for a specified target and example program.

Select Target

Filter Tags: (no tags selected, hide rtl) Contains:

qemu-sifive-s51 showing 16 of 32

SiFive QEMU S51 is a virtual development platform matching the Freedom S510. It's the best way to start prototyping and developing your RISC-V applications.

This target is ideal for getting familiarize with RISC-V ISA instructions set and freedom-metal libraries. It supports:

Select Example Program

hello

hello

A simple "Hello, World!" example to demonstrate printf and build environment.

Options

Project name: qemu-sifive-s51-hello-1

(you can change the project name on the next page)

☑ Create a debug launch configuration for QEMU

< Back | Next > | Finish | Cancel

Figure 2: New Project Dialogue Box

How to Update New Project Target List

Custom cores are shipped with a board support package (bsp) which can be used to create new software projects specific to the design. To create a project for a custom core, use the SiFive Tools menu option to select the freedom-e-sdk project contained in the design tarball.

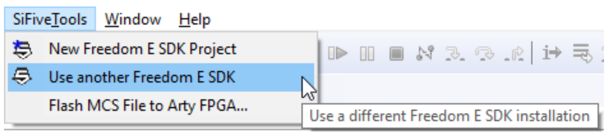
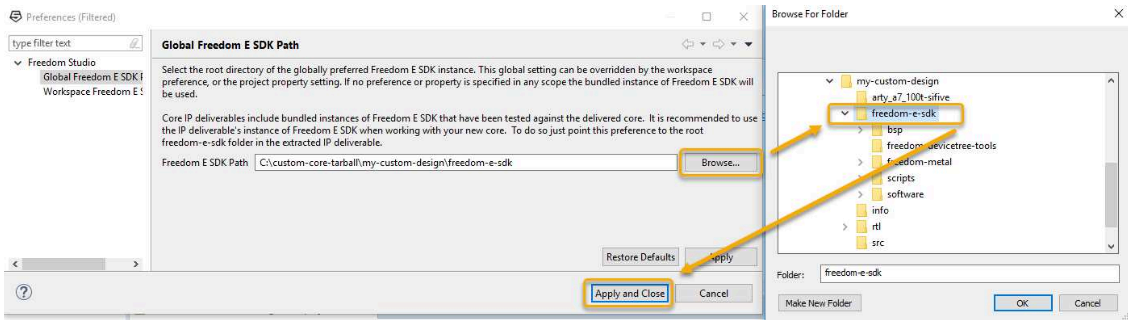

SiFiveTools
Window
Help
New Freedom E SDK Project
Use another Freedom E SDK
Flash MCS File to Arty FPGA...
Use a different Freedom E SDK installation

Figure 3: Use a Custom BSP

Then select the freedom-e-sdk root folder:


Preferences (Filtered)

Global Freedom E SDK Path

Select the root directory of the globally preferred Freedom E SDK instance. This global setting can be overridden by the workspace preference, or the project property setting. If no preference or property is specified in any scope the bundled instance of Freedom E SDK will be used.

Core IP deliverables include bundled instances of Freedom E SDK that have been tested against the delivered core. It is recommended to use the IP deliverable's instance of Freedom E SDK when working with your new core. To do so just point this preference to the root freedom-e-sdk folder in the extracted IP deliverable.

Freedom E SDK Path: C:\custom-core-tarball\my-custom-design\freedom-e-sdk

Browse For Folder

Folder: freedom-e-sdk

Figure 4: New freedom-e-sdk root folder

Click OK, and then revisit the **New Project** button. Now, the custom core will be available in the pull-down menu in the **Select Target** box.

The project will begin to build automatically after **Finish** is clicked in the **Create a Freedom E SDK Project** dialogue box. Once the build is complete, a debug configuration window will pop up. Click cancel for now, and view your new project in the Project Explorer Window to navigate the build settings.

1.2.2 Project Explorer Window

The Project Explorer Window shows a full view of the project including source files, include files, and makefiles required for building.

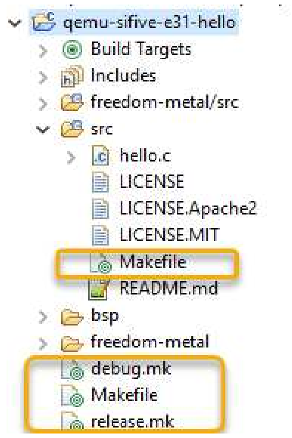


Figure 5: Project Explorer Window

Note that there two files named `Makefile`. The root folder Makefile specifies compiler and linker options. The /src filder Makefile specifies compiler optimizations and debug flags. Additionally, `debug.mk` and `release.mk` specify compiler optimizations. These are discussed in detail below.

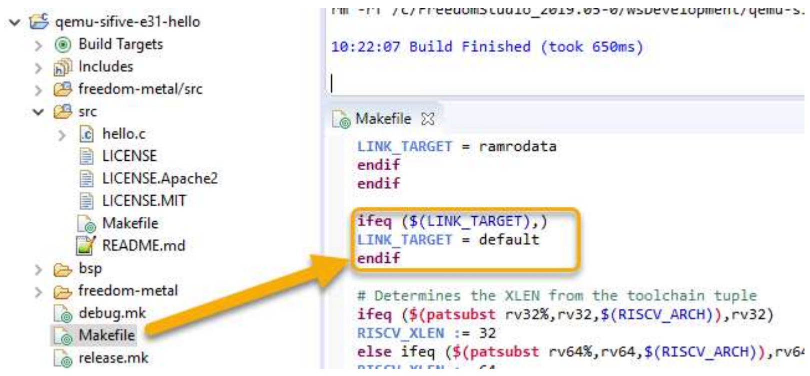
1.3 Makefile Linker Options

The root path Makefile defines which linker file to use when building your project.

1.3.1 How to Specify the Linker File

LINK_TARGET options include the following:

- **LINK_TARGET = default** → Code + Data in SPI Flash
- **LINK_TARGET = ramrodata** → RO Data in RAM, Code in Flash
- **LINK_TARGET = scratchpad** → All Code + Data in RAM



```
rm -rf /c/FreedomStudio_2019.05-0/wsDevelopment/qemu-s

10:22:07 Build Finished (took 650ms)
```

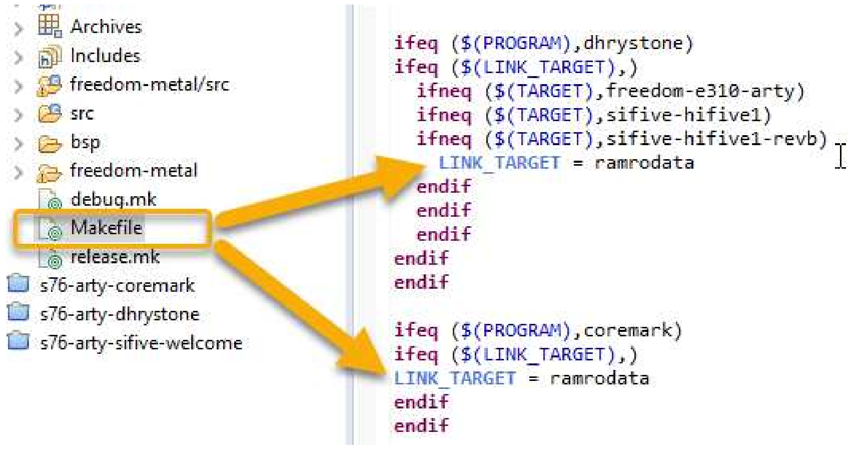
```
LINK_TARGET = ramrodata
endif
endif

ifeq ($(LINK_TARGET),)
LINK_TARGET = default
endif

# Determines the XLEN from the toolchain tuple
ifeq ($(patsubst rv32%,rv32,$(RISCV_ARCH)),rv32)
RISCV_XLEN := 32
else ifeq ($(patsubst rv64%,rv64,$(RISCV_ARCH)),rv6
```

Figure 6: Makefile Linker Option Description

Note

.bss, .stack, and .heap always reside in RAM for all configurations above.

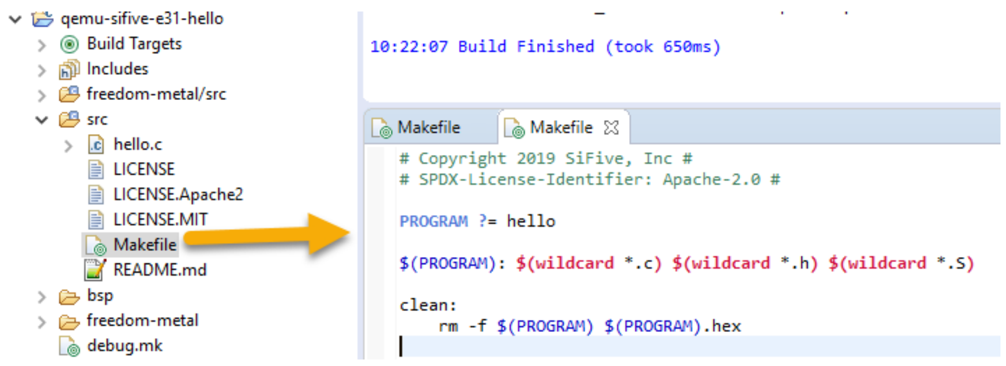
The **ramrodata** linker option is used for performance benchmarks such as Dhrystone and Coremark on FPGA and RTL Simulation platforms.



```
ifeq ($(PROGRAM),dhrystone)
ifeq ($(LINK_TARGET),)
  ifneq ($(TARGET),freedom-e310-arty)
  ifneq ($(TARGET),sifive-hifive1)
  ifneq ($(TARGET),sifive-hifive1-revb)
    LINK_TARGET = ramrodata
  endif
  endif
  endif
endif
endif

ifeq ($(PROGRAM),coremark)
ifeq ($(LINK_TARGET),)
LINK_TARGET = ramrodata
endif
endif
```

Figure 7: Benchmark Linker Selection

The src/Makefile shows the following:



```
10:22:07 Build Finished (took 650ms)
```

```
# Copyright 2019 SiFive, Inc #
# SPDX-License-Identifier: Apache-2.0 #

PROGRAM ?= hello

$(PROGRAM): $(wildcard *.c) $(wildcard *.h) $(wildcard *.S)

clean:
	rm -f $(PROGRAM) $(PROGRAM).hex
```

Figure 8: Source Makefile Content

> **Note**
>
> The src/Makefile contents may vary based on the software example

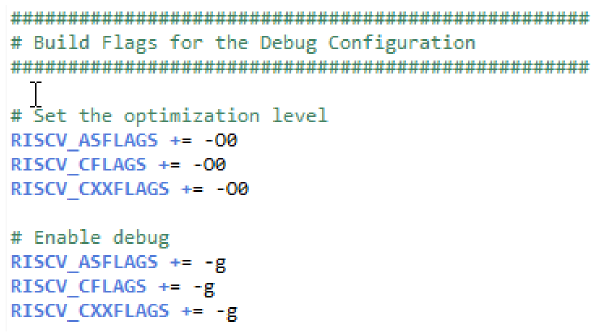
The wildcard option provides a flexible method for the make process to pick up new files as they are added to the project. Adding new source code is as easy as dragging and dropping them into the Project Explorer window. Or, right click on the /src folder and select New → Source File.

Additionally, this file can be modified to add paths to include additional sub folders for custom projects.

1.4 Compiler Options

he **debug.mk** and **release.mk** files contain the compiler optimization flags.

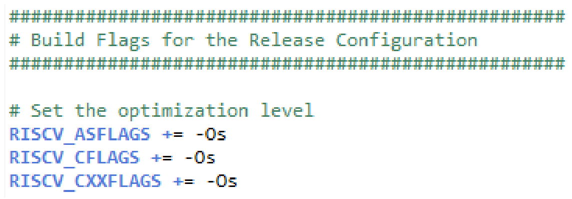
Example content for debug.mk:



```
####################################################
# Build Flags for the Debug Configuration
####################################################

# Set the optimization level
RISCV_ASFLAGS += -O0
RISCV_CFLAGS += -O0
RISCV_CXXFLAGS += -O0

# Enable debug
RISCV_ASFLAGS += -g
RISCV_CFLAGS += -g
RISCV_CXXFLAGS += -g
```

Figure 9: debug.mk file

Example content for release.mk:



```
#####################################################
# Build Flags for the Release Configuration
#####################################################

# Set the optimization level
RISCV_ASFLAGS += -Os
RISCV_CFLAGS += -Os
RISCV_CXXFLAGS += -Os
```

Figure 10: release.mk file

To toggle between **release** and **debug**, use the pull-down arrow next to the hammer icon:

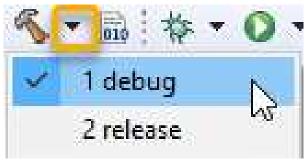


Figure 11: Debug or Release Selection

Note

Ensure the **release** build option is selected when running a benchmark example, or the optimizations may not get set correctly. This could cause issues with code size and performance, possibly resulting in a lower than expected benchmark score.

The Makefile process will include the appropriate *.mk file based on this selection. If a different compiler optimization is required, it can be changed here by directly modifying the file itself.

1.5 Debug Options

The debug options are described in the Debug Configurations menu.

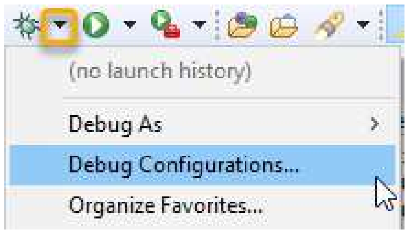
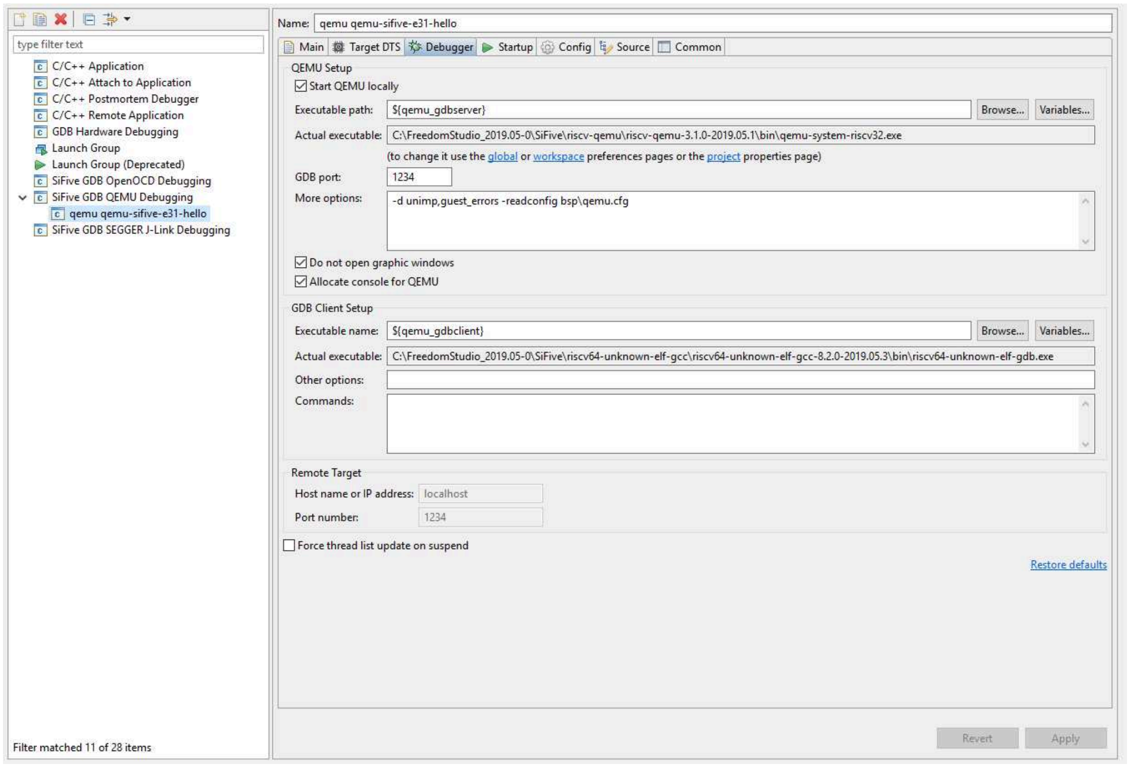


Figure 12: Debug Configuration Selection

In the **Debugger** tab shows the openOCD and GDB information.

For QEMU targets, our host is a virtualized environment, so we use the GDB server included in the QEMU path.


type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
GDB Hardware Debugging
Launch Group
Launch Group (Deprecated)
SiFive GDB OpenOCD Debugging
SiFive GDB QEMU Debugging
qemu qemu-sifive-e31-hello
SiFive GDB SEGGER J-Link Debugging
Filter matched 11 of 28 items
Main
Target DTS
Debugger
Startup
Config
Source
Common
Browse...
Variables...
Browse...
Variables...
Restore defaults
Revert
Apply

Name: qemu qemu-sifive-e31-hello

QEMU Setup

☑ Start QEMU locally

Executable path: ${qemu_gdbserver}

Actual executable: C:\FreedomStudio_2019.05-0\SiFive\riscv-qemu\riscv-qemu-3.1.0-2019.05.1\bin\qemu-system-riscv32.exe

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 1234

More options: -d unimp,guest_errors -readconfig bsp\qemu.cfg

☑ Do not open graphic windows

☑ Allocate console for QEMU

GDB Client Setup

Executable name: ${qemu_gdbclient}

Actual executable: C:\FreedomStudio_2019.05-0\SiFive\riscv64-unknown-elf-gcc\riscv64-unknown-elf-gcc-8.2.0-2019.05.3\bin\riscv64-unknown-elf-gdb.exe

Other options:

Commands:

Remote Target

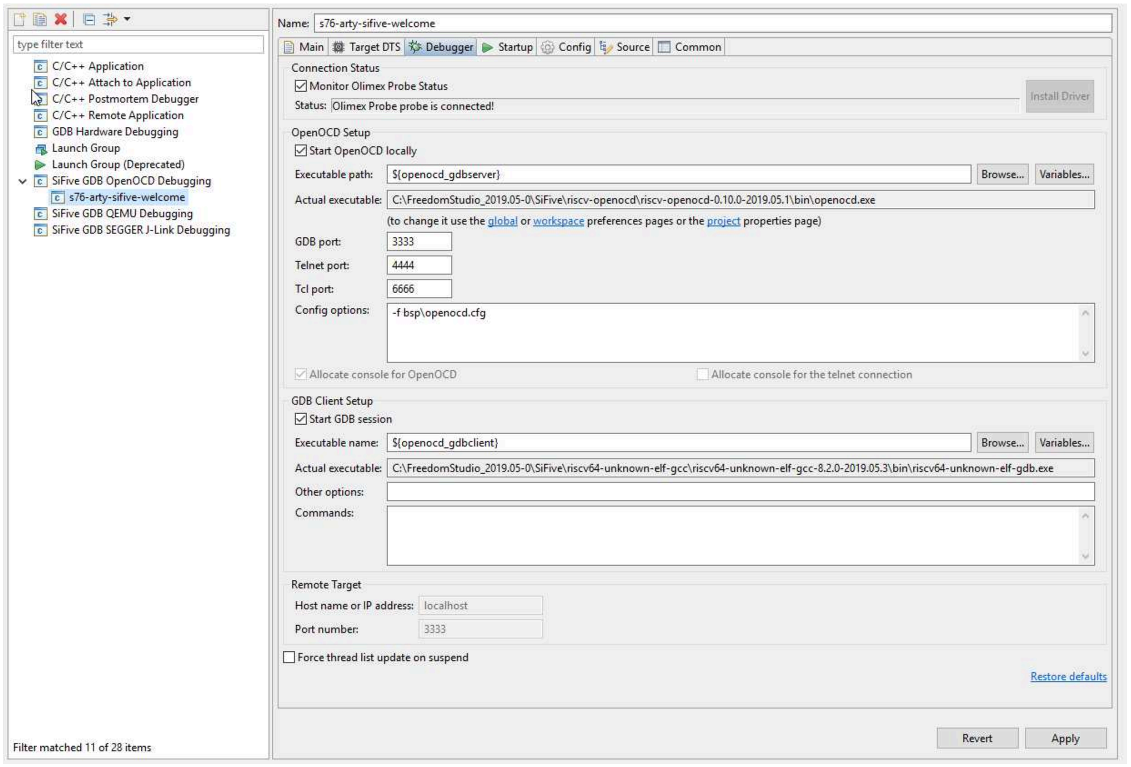
Host name or IP address: localhost

Port number: 1234

☐ Force thread list update on suspend

Figure 13: QEMU Debug Configuration

For hardware targets such as the Arty 100T FPGA board, we use the openOCD toolchain path, located in our Freedom Studio bundle.



Name: s76-arty-sifive-welcome

Connection Status
- [✓] Monitor Olimex Probe Status
- Status: Olimex Probe probe is connected!

OpenOCD Setup
- [✓] Start OpenOCD locally
- Executable path: ${openocd_gdbserver}
- Actual executable: C:\FreedomStudio_2019.05-0\SiFive\riscv-openocd\riscv-openocd-0.10.0-2019.05.1\bin\openocd.exe
- (to change it use the global or workspace preferences pages or the project properties page)
- GDB port: 3333
- Telnet port: 4444
- Tcl port: 6666
- Config options: -f bsp\openocd.cfg
- [✓] Allocate console for OpenOCD
- [ ] Allocate console for the telnet connection

GDB Client Setup
- [✓] Start GDB session
- Executable name: ${openocd_gdbclient}
- Actual executable: C:\FreedomStudio_2019.05-0\SiFive\riscv64-unknown-elf-gcc\riscv64-unknown-elf-gcc-8.2.0-2019.05.3\bin\riscv64-unknown-elf-gdb.exe
- Other options:
- Commands:

Remote Target
- Host name or IP address: localhost
- Port number: 3333

[ ] Force thread list update on suspend

Figure 14: ARTY Debug Configuration using Olimex Hardware

Navigate the other tabs to see additional configuration options.

Note

When using the Digilent Arty FPGA board along with either the Olimex or J-Link debug hardware, no changes are required to any of the debug configuration tabs when using a standard JTAG connection. This applies to new software project created using the Freedom Studio New Project Method described earlier, where the custom core design being debugged was generated through SiFive.com.

1.5.1 Multi-core OpenOCD Configuration

In multi-core designs, the JTAG connectivity is often daisy-chained so there is only one set of JTAG pins exposed in the design. Daisy chaining is supported for up to 20 TAP controllers.

JTAG	Core_0	Core_1	Core_n
TMS	TMS	TMS	TMS
TCK	TCK	TCK	TCK
TDI	TDI	Core_0.TDO	Core_1.TDO
TDO	Core_1.TDI	Core_n.TDI	TDO

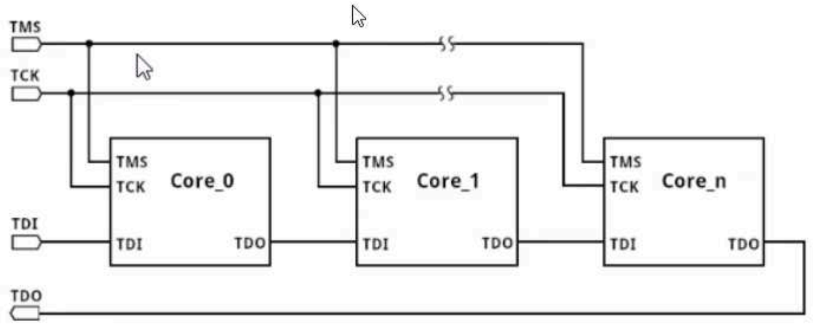


Figure 15: Multi-core JTAG Connectivity

If the target hardware is a multi-core design from SiFive, then changes to the openocd.cfg file are required. This file is used by Freedom Studio and defined by navigating to Debug Configurations → Debugger tab.

Multi-core openocd.cfg Example

The standard configuration below explicitly tells OpenOCD there's exactly one hart, so that is all that is exposed by OpenOCD to gdb.

```
set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -chain-position $_TARGETNAME
$_TARGETNAME configure -work-area-phys 0x80000000 -work-area-size 10000 \
-work-area-backup 1
```

Let's take a multi-core example containing five SiFive CPUs. In this case, the above example should be replaced with the following:

```
set _TARGETNAME_0 $_CHIPNAME.cpu0
set _TARGETNAME_1 $_CHIPNAME.cpu1
set _TARGETNAME_2 $_CHIPNAME.cpu2
set _TARGETNAME_3 $_CHIPNAME.cpu3
set _TARGETNAME_4 $_CHIPNAME.cpu4
target create $_TARGETNAME_0 riscv -chain-position $_CHIPNAME.cpu -rtos hwthread
target create $_TARGETNAME_1 riscv -chain-position $_CHIPNAME.cpu -coreid 1
target create $_TARGETNAME_2 riscv -chain-position $_CHIPNAME.cpu -coreid 2
target create $_TARGETNAME_3 riscv -chain-position $_CHIPNAME.cpu -coreid 3
target create $_TARGETNAME_4 riscv -chain-position $_CHIPNAME.cpu -coreid 4
target smp $_TARGETNAME_0 $_TARGETNAME_1 $_TARGETNAME_2 $_TARGETNAME_3 $_TARGETNAME_4
```

1.5.2 cJTAG Configuration

For smaller designs, a 2-pin JTAG configuration is available where pin count is critical. A hardware adapter can be used in between the debug hardware and the ribbon cable as shown here.

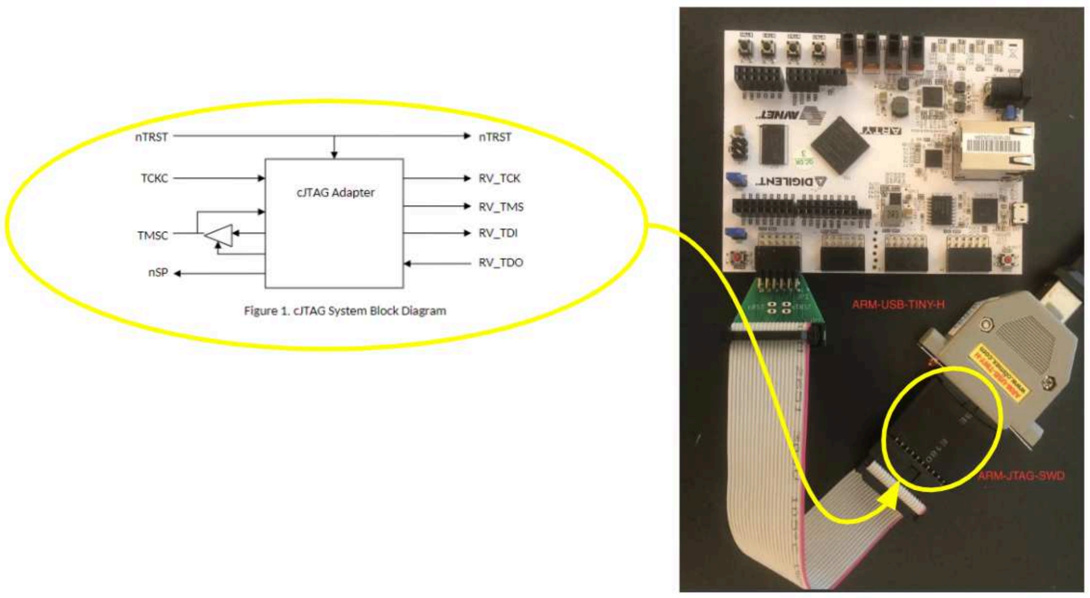


Figure 16: ARTY cJTAG Configuration

cJTAG openocd.cfg Example

A different openocd configuration file is required if using cJTAG. An example openocd.cfg file that supports cJTAG is shown here.

```
adapter_khz     10000

#source [find interface/ftdi/olimex-arm-usb-tiny-h.cfg]

interface ftdi
ftdi_oscan1_mode on
ftdi_device_desc "Olimex OpenOCD JTAG ARM-USB-TINY-H"
ftdi_vid_pid 0x15ba 0x002a

ftdi_layout_init 0x0808 0x0a1b
ftdi_layout_signal nSRST -oe 0x0200
# oscan1_ftdi_layout_signal nTRST -data 0x0100 -oe 0x0100
ftdi_layout_signal LED -data 0x0800

# These signals are used for cJTAG escape sequence on initialization only
ftdi_layout_signal TCK -data 0x0001
ftdi_layout_signal TDI -data 0x0002
ftdi_layout_signal TDO -input 0x0004
ftdi_layout_signal TMS -data 0x0008
ftdi_layout_signal JTAG_SEL -data 0x0100 -oe 0x0100

set _CHIPNAME riscv
jtag newtap $_CHIPNAME cpu -irlen 5
```

```
set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -chain-position $_TARGETNAME
$_TARGETNAME configure -work-area-phys 0x40000000 -work-area-size 10000
-work-area-backup 1

#flash bank my_first_flash fespi 0x40000000 0 0 0 $_TARGETNAME 0x20004000
init
#reset
if {[ info exists pulse_srst]} {
  ftdi_set_signal nSRST 0
  ftdi_set_signal nSRST z
}
halt
#flash protect 0 64 last off
```

1.6 Porting

Porting software projects into Freedom Studio simply involves creating a new project, then just adding the source code. This assumes we use the embedded GCC toolchain and libs, and use OpenOCD/GDB or Segger JLink for debugging.

Here is a basic outline of what needs to be done to port a project over:

1. Download and extract the latest Freedom Studio from SiFive.com Boards page.
    - Scroll down a little bit on that page to see the Freedom Studio links.
    - Extract the Freedom Studio package and launch Freedom Studio from the root directory.
2. Follow the new project wizard procedure mentioned previously, or refer to the the documentation here.
    - An example like return-pass or the basic hello will provide a simple template for adding source code.
    - If a different project or file name is desired, right click and change the file name, or right click and change the project name.
        - Keep in mind the debug settings also have to reflect these changes since the Debugger needs to know where the .elf file resides.
        - Go to Run → Debug Configurations and update paths and/or filenames as necessary if a file or project name was changed.
3. Add source code files to the /src path of the new project and they get picked up automatically by the build process.
    - For a hierarchy structure, i.e., multiple folders within /src, right click /src → New → Folder

- The root path Makefile will also need to be updated to point to the new paths if they contain source code to be built

1.6.1 More Information

For more information, click Help → Browse Tools Docs on the Freedom Studio Menu bar.