SiFive Freedom Studio Manual

June 16, 2017

SiFive Freedom Studio Manual

Release Information

Version	Date	Changes
v1p0	May 6, 2017	N/A
v1p1	June 16, 2017	Change doc name to from Quick Start Guide to Manual Updated plugin versions Documentation now matches new ilg plugins Added Known Issues Section Troubleshooting section changed to OS Specific Board Setup Appendix

Contents

SiFive Freedom Studio Manual **i**

1 Introduction **1**

1.1 Freedom Studio Introduction 1

1.2 Product Overview 1

1.2.1 Eclipse 1

1.2.2 RISC-V GCC 2

1.2.3 OpenOCD 2

2 Freedom Studio Set Up **3**

2.1 Download and Install 3

2.2 Hardware Setup 3

2.3 Freedom Studio Contents 3

2.4 Environment Setup 4

3 Freedom Studio Environment **7**

3.1 Workspace 7

3.2 Perspectives 7

3.2.1 C/C++ Perspective 8

3.2.2 Debug Perspective 10

3.2.3 Additional Debug Views 11

4 Getting Started **13**

4.1 Import the Bundled Examples 13

4.1.1 Bundled Examples Step by Step 13

4.2 Import Freedom-E-SDK Examples 15

4.2.1 Freedom-E-SDK Examples Step by Step 15

4.3 Creating a New Project From Scratch 16
4.4 Debug 17

5 Known Issues **19**

A Linux OS Board Setup **21**
A.1 Hardware Setup 21
A.2 Linux Serial Terminal Setup 22

B macOS Board Setup **23**
B.1 Hardware Setup 23
B.2 macOS Serial Terminal Setup 23

C Windows Board Setup **25**
C.1 Hardware Setup 25
C.2 Windows Serial Terminal Setup 25

List of Figures

2.1 First Run Dialog 4

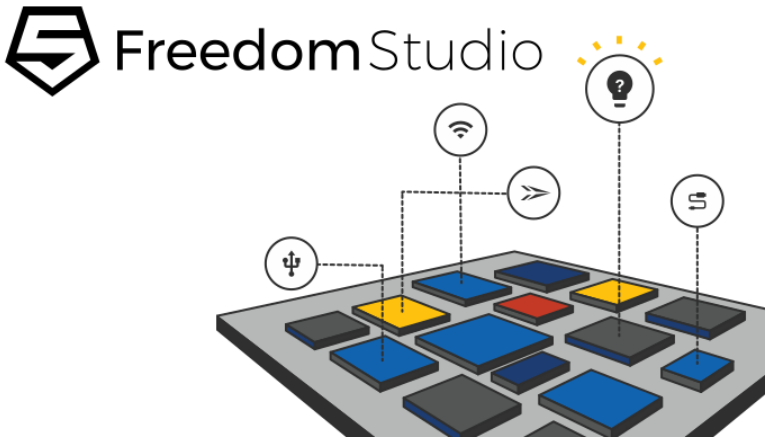
3.1 C/C++ Perspective 8
3.2 Build Settings Menu 9
3.3 Debug Perspective 10
3.4 Expression View 11

4.1 Import Bundled Examples 14
4.2 Import Freedom-E-SDK Examples 16

C.1 Windows Putty Setup 26

Chapter 1

Introduction



Freedom Studio Introduction

Freedom Studio is an integrated developer environment which can be used to write and debug software targeting SiFive based processors. Freedom Studio is based on the industry standard Eclipse platform and is bundled with a pre-built RISC-V GCC Toolchain, OpenOCD, example programs, and documentation.

Product Overview

This section will describe the individual components used in Freedom Studio release Beta 2.

Eclipse

The major versions of the Eclipse plugins are as follows:

- Eclipse - Neon.2 (4.6.3)
- CDT - 9.2.1.2017062208
- egit - 4.4.1.201703071140

- GNU MCU Eclipse RISC-V Cross - 2.5.1.2017.06121008
- GNU MCU Eclipse OpenOCD - 4.1.5.2017.06121008

RISC-V GCC

RISC-V GCC was built from source using the following repository and commit hash:

- Repository: https://github.com/riscv/riscv-gnu-toolchain
- Commit hash: f5fae1c27b2365da773816ddcd92f533867f28ec

This build of GCC is able to target RV32 and RV64 based processors.

OpenOCD

Freedom Studio uses OpenOCD for hardware debug. OpenOCD was built from source using the following repository and commit hash:

- Repository: https://github.com/riscv/riscv-openocd
- Commit hash: 5747469eae56101bd167b450bc4d802162b113c0

Chapter 2

Freedom Studio Set Up

Download and Install

Freedom Studio can be downloaded from the SiFive website at the following address: https://www.sifive.com/products/tools/

Unzip the download to a directory on your computer. To start the Eclipse environment under Windows and Linux, double-click the executable in the eclipse folder ("eclipse/FreedomStudio"). On MacOS simply click the "FreedomStudio" launcher in the root FreedomStudio folder.

In Windows, you can create a desktop shortcut by right-clicking on the FreedomStudio.exe, and selecting ***Send To – Desktop***.

Hardware Setup

This Manual assumes that you have already completed the Getting Started guide for your particular hardware, including the OpenOCD setup. If you have not, please refer to the Appendix of this manual or the Getting Started guide for your particular hardware and ensure that you are able to connect a serial terminal to your board and upload applications.

Hardware documentation can be found in the "Freedom Studio/SiFive/Documentation" directory and also online at: https://www.sifive.com/documentation/.

Freedom Studio Contents

The Freedom Studio directory contents are as follows:

- FreedomStudio - Root directory
  - eclipse (Windows and Linux only)- Directory containing the eclipse environment
  - SiFive - SiFive files
    - Documentation - documentation delivered with Freedom Studio.
    - Examples - Zip files containing example projects for a given board.
    - Misc - Directory containing miscellaneous files such as OpenOCD config files, drivers, and Linux OpenOCD udev rules

    - OpenOCD - Directory containing the OpenOCD build described in Section 1.2.
    - riscv64 - Directory containing the RISC-V GCC build described in Section 1.2.
  - Build Tools (Windows Only) - Tools which allow eclipse CDT to function in a Windows environment such as make, echo, etc...
  - jre (Windows and Linux Only) - The Java Run Time Environment (JRE). On macOS the JRE is located under the FreedomStudio.app bundle.

Environment Setup

The first time the Freedom Studio environment is run, the user will be prompted to set up paths to the toolchain and OpenOCD via the dialog shown in Figure 2.1. These paths will set the global defaults used by Freedom Studio.

The Freedom Studio bundle includes a pre-built toolchain and OpenOCD in the location described in Section 2.3. The pre-built toolchain and OpenOCD have been tested working with this release of Freedom Studio and are the recommended default. Be sure to select the “bin” directory which contains the actual binaries.

These paths can be changed at any time by using the preferences menu as follows:

- Windows and Linux - ***Window – Preferences – MCU***
- MacOS - ***Freedom Studio – Preferences – MCU***

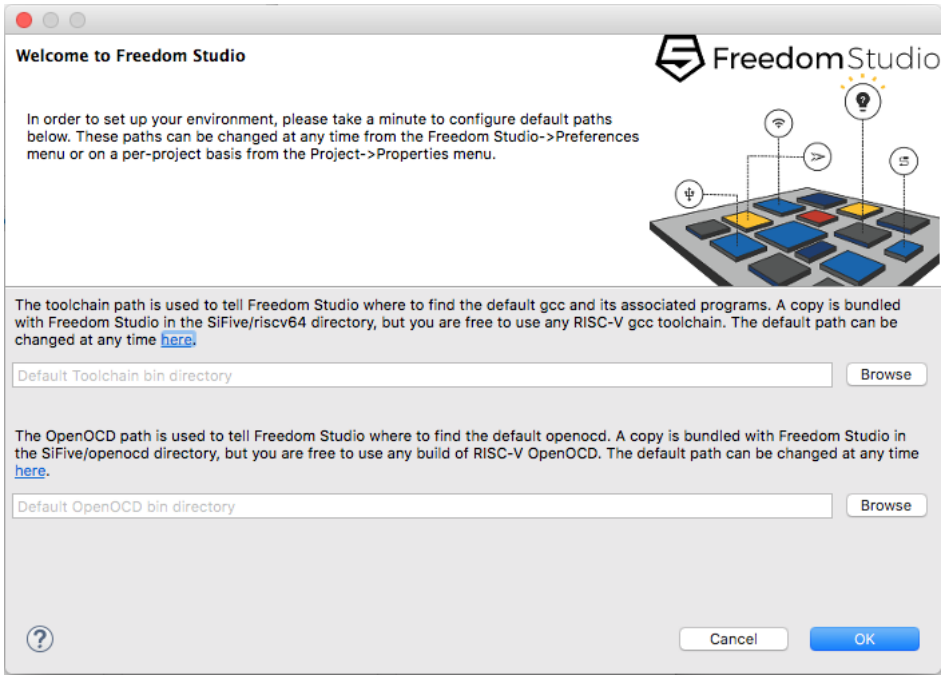


Figure 2.1: First Run Dialog

The environment preferences can be set at 3 different scopes: Global, Workspace, and Project. Global scope sets the default for the installation and is the lowest priority. Workspace scope allows

you to set the toolchain preferences specific a a given Workspace, and will override the Global setting. And finally Project scope, which can be set by right clicking a project in your workspace and selecting ***Properties – MCU***, allows you to set preferences on a per project basis. Project scope always takes priority over Global and Workspace.

This flexibility allows the user to easily work with a number of different toolchains installed on the same system, such as one built from source using Freedom-E-SDK, while still maintaining project portability.

Chapter 3

Freedom Studio Environment

Workspace

Eclipse uses workspaces to group together a set of related projects. Eclipse workspaces allow for a lot of flexibility in how one organizes their projects. For example, it is possible to have a workspace which contains only a single project. It is also possible to have a workspace which contains multiple related projects such as a library project and an application which depends on that library.

Switching between workspaces is accomplished by selecting ***File – Switch Workspace***.

Perspectives

Eclipse uses perspectives to group certain windows together which are used for the same tasks. Freedom Studio currently has 3 main perspectives: C/C++ , Debug, and Git. From Eclipse, you can change perspectives by clicking ***Window – Perspectives — Open Perspective***.

Perspectives are fully user customizable and persistent to a workspace. The sections below describe the default perspectives.

C/C++ Perspective

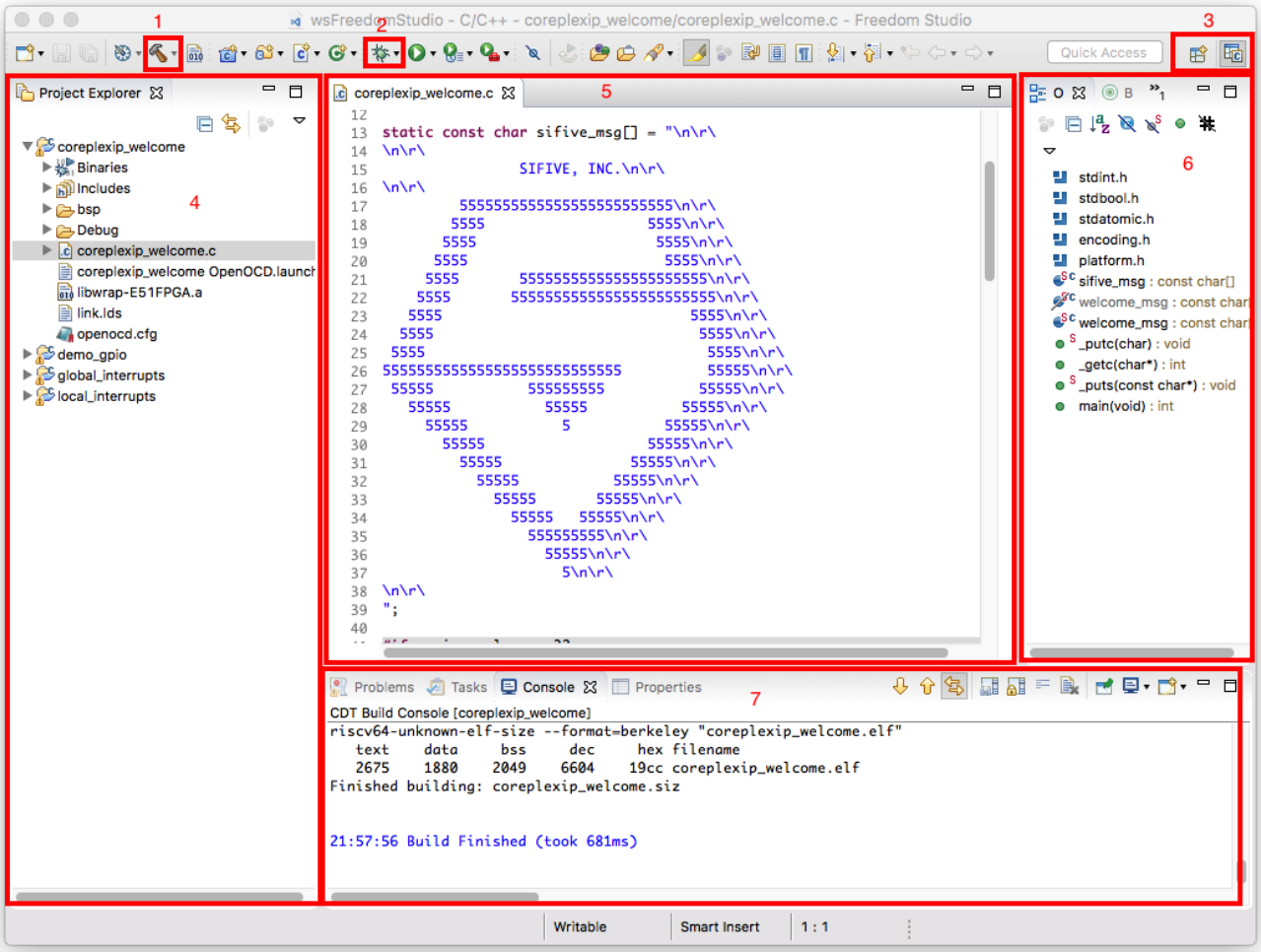


Figure 3.1: C/C++ Perspective

1. Build Toolbar Button.
2. Debug Toolbar Button. The down arrow next to the bug lets you pick a particular configuration.
3. Perspective Toolbar - allows you to quickly change perspectives.
4. Project Explorer - Displays all projects in the workspace.
5. Editor - Main view for editing source files.
6. Outline View - Displays the source code Outline of the active editor file.

7. Console - Displays useful information from project builds.

The build settings for a particular project can be viewed by right-clicking on a project in the Project Explorer, and selecting ***Properties – C/C++ Build – Settings***.

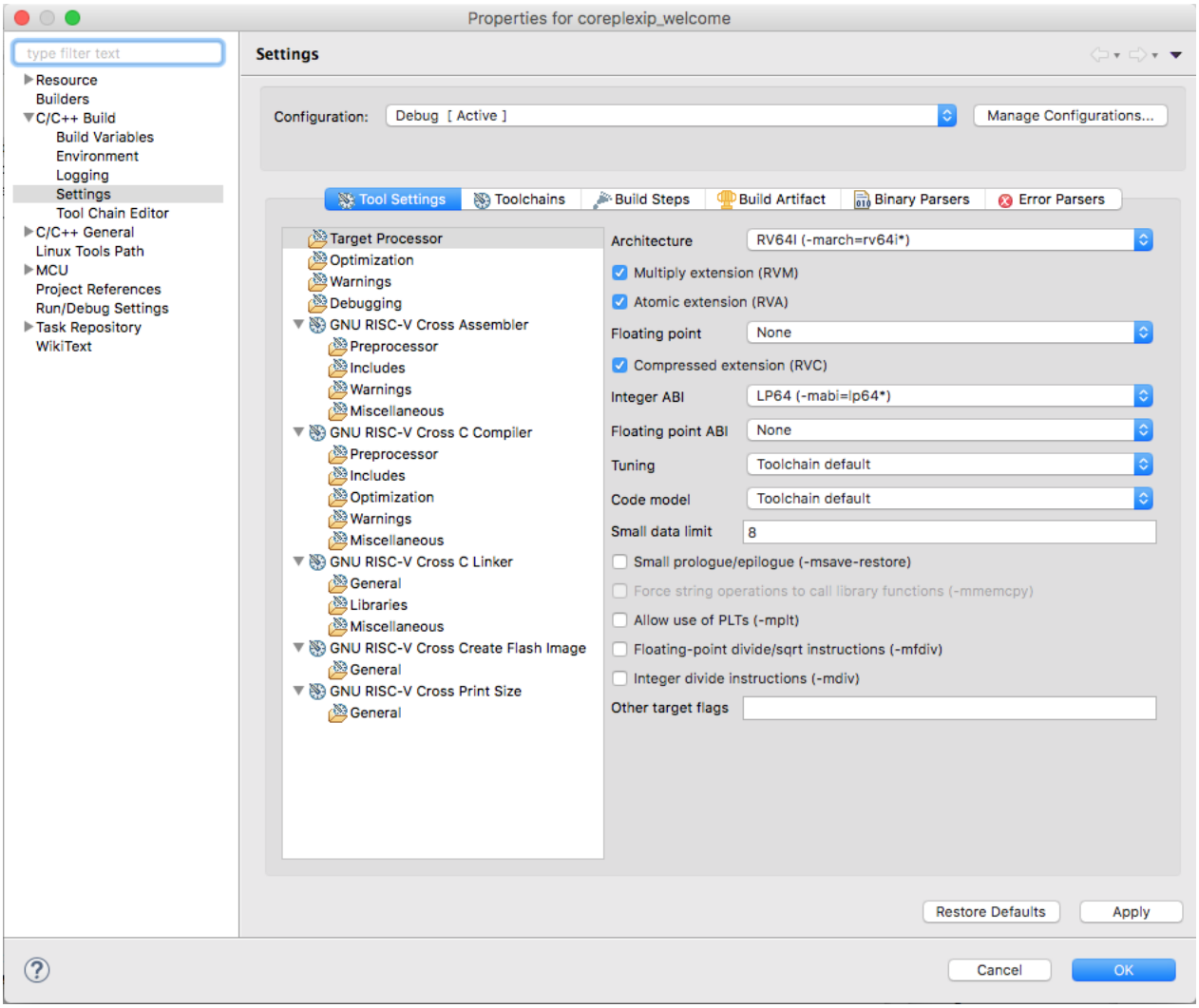


Figure 3.2: Build Settings Menu

Debug Perspective

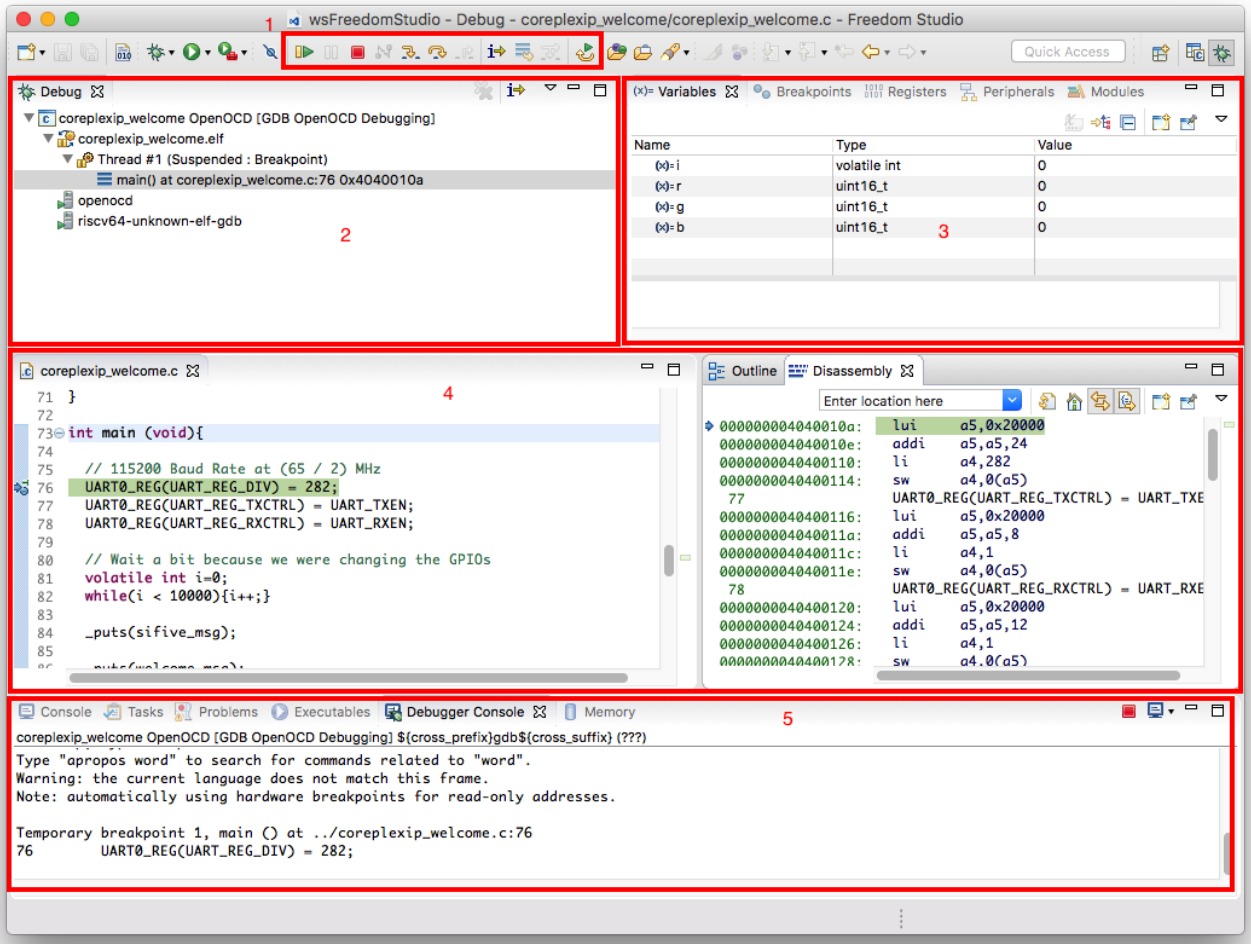


Figure 3.3: Debug Perspective

1. Runtime Controls - (from left to right) Start, Halt, Stop, Step Into, Step Over, Step Out, Step Instruction, Restart.

2. Runtime Context - Displays current context information on a per-thread basis.

3. Debug Views - Variables, Core Registers, Breakpoints.

4. Source View - Displays current PC in source file and in dissasembly.

5. Consoles - Displays GDB and OpenOCD console output.

Additional Debug Views

There are other useful Debug views which are not displayed in the default Debug Perspective, specifically the Disassembly View and Expressions View. In order to add these views to the open Debug Perspective click ***Windows – Show View*** and select the view from the list.

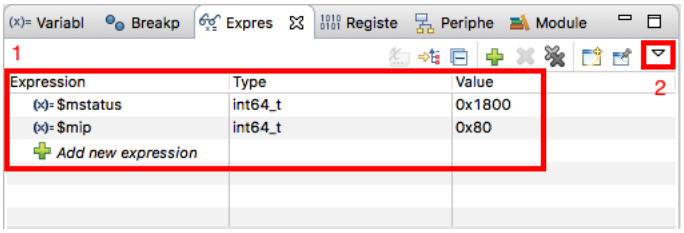


Figure 3.4: Expression View

The Expression view allows you to view any variable within scope. In addition to variables, it is possible to use this view to see the current value of CSRs on your device. The Expression view, along with other eclipse views which display variables and memory, allows you to change the format of the displayed value. The format can be changed by clicking the down arrow marked with “2” in Figure 3.4.

Chapter 4

Getting Started

Import the Bundled Examples

Freedom Studio bundles several examples for each SiFive development platform which are suitable starting a new project. These are the same examples from the Freedom-E-SDK but packaged to where each example is self-contained and not reliant on other projects. Specifically this means that relevant files in the *bsp* directory have been copied into the project folder as well as *libwrap.a*, and Include Paths have been changed accordingly.

Bundled Examples Step by Step

After opening Freedom Studio and selecting a workspace (any location is OK), the procedure is as follows:

- From Freedom Studio click ***File – Import***.
- ***General – Existing Projects into Workspace***.
- ***Select Archive File***.
- Browse to your *Freedom Studio/SiFive/Examples* directory and select the zip file for your platform.
- Select as many of the examples as you like, and click ***Finish***.

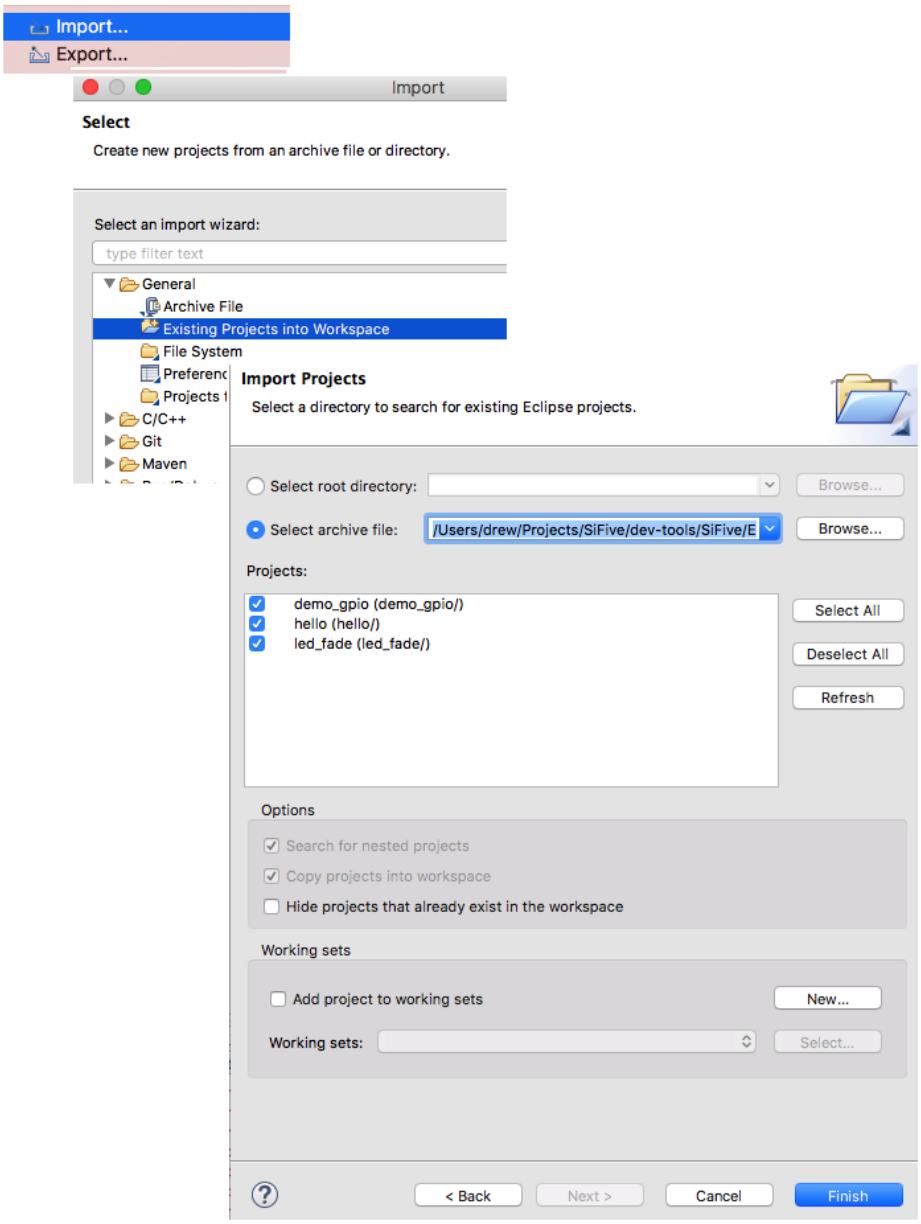


Figure 4.1: Import Bundled Examples

The projects should now be copied into your workspace. Pressing the Build (hammer) icon will build the currently selected project or Control+B (Command+B on macOS) will build the entire workspace.

If switching to a new platform (for example from E31FPGA to E51FPGA), it is reccomended that you also switch to a new workspace.

Import Freedom-E-SDK Examples

Freedom-E-SDK also includes Freedom Studio example projects in the *Freedom-E-SDK/FreedomStudio* directory. These projects are set up to link to the files in the rest of the Freedom-E-SDK so as the same source files can be built from both Freedom Studio and the classic Freedom-E-SDK makefiles.

Freedom-E-SDK is maintained on github and will therefore always have the latest versions of the example projects compared the bundled examples which are only updated on new releases of Freedom Studio.

Freedom-E-SDK Examples Step by Step

After opening Freedom Studio and selecting a workspace (any location is OK), the procedure is as follows:

- From Freedom Studio click ***File – Import***.
- ***General – Existing Projects into Workspace***.
- ***Select Root Directory***.
- Browse to your *freedom-e-sdk/FreedomStudio* directory and select the folder which matches your platform.
- Select the wrap-boardname project, and as many of the other projects as you like.
- Make sure that “Copy projects into workspace” is not checked, and click ***Finish***.

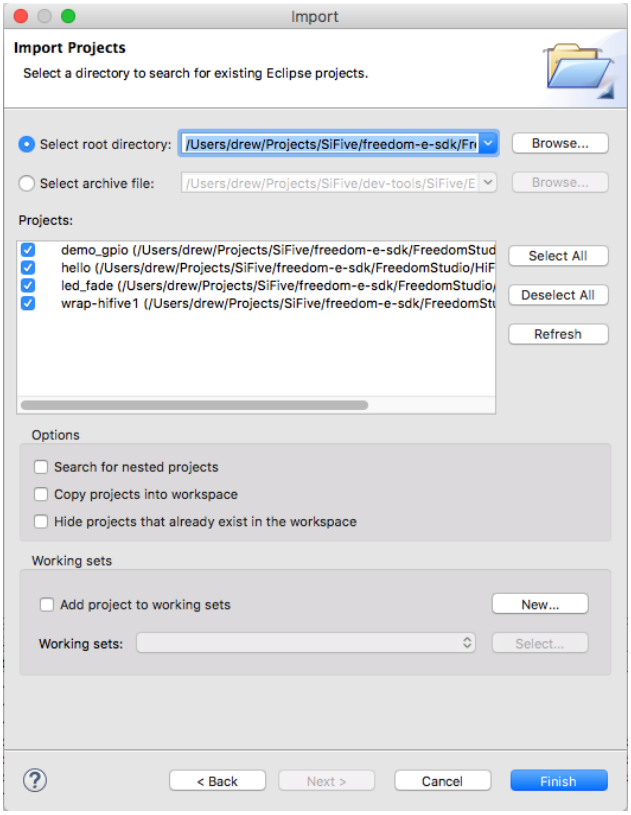


Figure 4.2: Import Freedom-E-SDK Examples

This should result in the selected projects being imported into your workspace. Note that the actual source files for the project remain in the Freedom-E-SDK folder, and any modifications you make to them in Freedom Studio will modify the original Freedom-E-SDK files.

If switching to a new platform (for example from E31FPGA to E51FPGA), it is recommended that you also switch to a new workspace as most of the projects share the same names between the platforms.

Creating a New Project From Scratch

It is also possible to create a new project from scratch.

- After opening Freedom Studio, select ***File – New***.
- Select the Project Type: C, C++.
- Under *Project Type* select Executable or Shared Library, and select *RISC-V Cross GCC* as the Toolchain.
- Give the project a name and select ***Finish***.

From here you will have an empty project which you can begin development. Toolchain settings can be found by right-clicking on the project and selecting ***Properties – C/C++ Build – Settings*** as described in Section 3.2.1.

Debug

All of the examples include launch files which tell Freedom Studio how to start a debug connection with the target platform. All launch configurations are set to load the program into the address to which they were linked (defined in the linker file), and to halt execution at the *main* function.

To launch a Debug session, click ***Run – Debug Configurations*** and then ***GDB OpenOCD*** followed by the desired program. By default this opens the Debug perspective which contains lots of useful information for debugging. If a particular window of interest is not in the view, click ***Window – Show View*** for the full list of windows.

Some useful debug controls are described in Figure 3.3.

Chapter 5

Known Issues

Freedom Studio is still considered Beta software and as such there are a number of known, but minor, issues which are described below.

If you come across other issues not reported here, please let us know on our forum: https://forums.sifive.com/.

Will my Freedom Studio Beta 1 projects work with Freedom Studio Beta 2
Unfortunately not. Freedom Studio moved to a new CDT build plugin for Beta 2. It is recommended to create a new project as described in Section 4.3 and to copy your source files and linker scripts into the new project. We do not plan to change build plugins again anytime soon and apologize for the inconvenience.

When the debugger first connects I receive a message saying "No source available for address"
This occurs when instructing the debugger to halt immediately after connecting to the target. It is safe to ignore this message. Stepping/Running the target will work as expected from this point.

The Registers view displays registers which are not on my target, such as floating point registers and the entire CSR list
This is a known issue and something which will be addressed in the future. It should not interfere with the debug connection.

Upon starting a debug connection, the Console prints out a lot of text regarding CSRs in red font
This issue is related the Registers view issue described above. While red font can be scary, it is generally benign debugging output. This issue will be addressed in a future release.

Sometimes the Stack Frame Back Trace is displayed incorrectly.
This is a known issue and something which will be addressed in the future. It should not interfere with the debug connection.

Appendix A

Linux OS Board Setup

This section will describe how to connect SiFive development boards to your Linux computer.

Hardware Setup

By default, most Linux distributions do not give users permissions to access USB devices. One either needs root access or to be given the appropriate permissions.

Below are steps you can follow to access your development kit without `sudo` permissions:

1. With your board's debug interface connected, make sure your device shows up with the `lsusb` command:

   ```
   > lsusb
   ...
   Bus XXX Device XXX: ID 0403:6010 Future Technology Devices
   International, Ltd FT2232C Dual USB-UART/FIFO IC
   ```

2. Set the udev rules to allow the device to be accessed by the `plugdev` group:

   For your convenience the *99-openocd.rules* file is included with Freedom Studio in the *FreedomStudio/SiFive/Misc* directory.

   ```
   > sudo cp 99-openocd.rules /etc/udev/rules.d/
   ```

3. See if your board shows up as a serial device belonging to the `plugdev` group:

   ```
   > ls /dev/ttyUSB*
   /dev/ttyUSB0 /dev/ttyUSB1
   ```

   (If you have other serial devices or multiple boards attached, you may have more devices listed). For serial communication with the UART, you will always want to select the higher number of the pair, in this example `/dev/ttyUSB1`.

```
> ls -l /dev/ttyUSB1
crw-rw-r-- 1 root plugdev 188, 1 Nov 28 12:53 /dev/ttyUSB1
```

4. Add yourself to the `plugdev` group. You can use the `whoami` command to determine your user name.

```
> whoami your_user_name > sudo usermod -a -G plugdev your_user_name
```

5. Log out and log back in, then check that you're now a member of the plugdev group:

```
> groups
...  plugdev ...
```

   If you are not part of the plugdev group, perform a full reset.

Now you should be able to access the serial (UART) and debug interface without sudo permissions.

Linux Serial Terminal Setup

Now that you have permissions to access the device, use a terminal emulator such as GNU screen to open a console connection from the host computer to the development board.

Set the following parameters:

Speed	115200
Parity	None
Data bits	8
Stop bits	1
Hardware Flow	None

For example, on Linux using GNU Screen:

```
screen /dev/ttyUSB1 115200
```

You should now have a working serial connection to the development board.

You can use `Ctrl-a k` to "kill" (exit) the running screen session.

Appendix B

macOS Board Setup

This section will describe how to connect SiFive development boards to your macOS computer.

Hardware Setup

By default, macOS has the standard FTDI driver installed while OpenOCD expects to communicate over USB using libusb. In order to allow OpenOCD to communicate with the SiFive development boards, it is necessary to unload the FTDI driver from macOS.

- Open *Applications/Utilities/Terminal*
- Paste in the following command:
  `sudo kextunload -p -b com.apple.driver.AppleUSBFTDI`
- Paste in the following command:
  `sudo kextutil -b com.apple.driver.AppleUSBFTDI -p AppleUSBEFTDI-6010-1`

Note: This is not a permanent solution and after logging out of you computer it is necessary to issue the above commands above.

To avoid having to issue these commands on every log-in, it is possible to add the above commands to your user's *~/.bash_profile*. By doing so, the above commands will be issued automatically every time your user logs in.

macOS Serial Terminal Setup

Now that you are able to access the device, use a terminal emulator such as GNU screen to open a console connection from the host computer to the development board.

```
screen /dev/tty.usbserial-210319A28DF68 115200
```

You should now have a working serial connection to the development board.

You can use `Ctrl-a k` to "kill" (exit) the running screen session.

Appendix C

Windows Board Setup

This section will describe how to connect SiFive development boards to your Windows computer.

Hardware Setup

By default, Windows has the standard FTDI driver installed while OpenOCD expects to communicate over USB using libusb. In order to allow OpenOCD to communicate with the SiFive development boards, it is necessary to instruct Windows to load the libusb driver.

In the Windows Freedom Studio bundle there are drivers included to do this. Install the drivers by navigating to the *FreedomStudio/SiFive/Misc* folder and double-clicking the appropriate driver for your board.

Note: This is a permanent change and the drivers must be uninstalled in order return to the original functionality. This can be done through Windows Add/Remove programs.

Windows Serial Terminal Setup

Now that the correct platform drivers are installed, use a terminal emulator, such as putty (https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html), to open a console connection from the host computer to the development board.

First, determine the COM port of the board by right-clicking on the Windows icon, and selecting ***Device Manager***. In the Device Manager look for ***Ports (COM and LPT)*** and take note of the USB Serial Port COM number.

Open putty, select ***Serial***, type in the COM port number and 115200 baud as shown in Figure C.1 and click ***Open***. You should now have an open serial terminal which can be used for serial communication to your SiFive board.

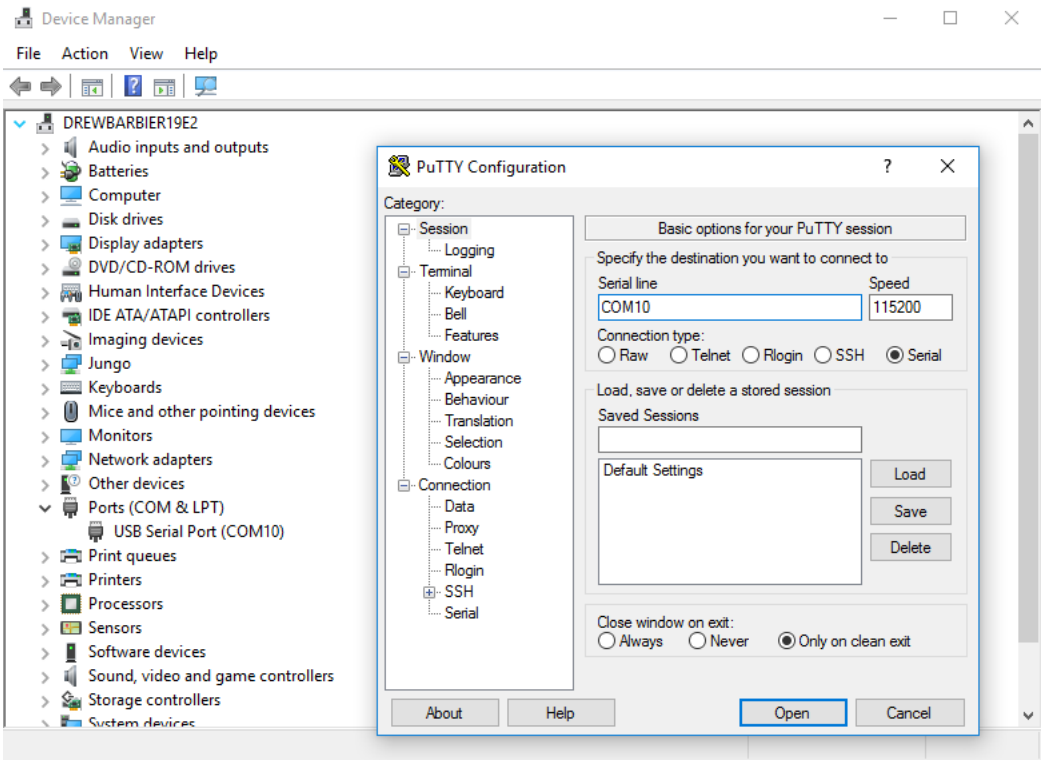


Figure C.1: Windows Putty Setup