

East Tennessee State University

Digital Commons @ East Tennessee State University

Undergraduate Honors Theses

Student Works

5-2022

Development of Classroom Tools for a RISC-V Embedded System

Lucas Phillips

Follow this and additional works at: <https://dc.etsu.edu/honors>



Part of the [Systems Architecture Commons](#)

Recommended Citation

Phillips, Lucas, "Development of Classroom Tools for a RISC-V Embedded System" (2022). *Undergraduate Honors Theses*. Paper 718. <https://dc.etsu.edu/honors/718>

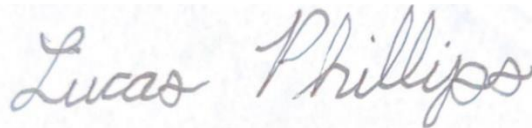
This Honors Thesis - Open Access is brought to you for free and open access by the Student Works at Digital Commons @ East Tennessee State University. It has been accepted for inclusion in Undergraduate Honors Theses by an authorized administrator of Digital Commons @ East Tennessee State University. For more information, please contact digilib@etsu.edu.

Development of Classroom Tools for a RISC-V Embedded System

By

Lucas Scott Phillips

An Undergraduate Thesis Submitted in Partial Fulfillment
of the Requirements for the
University Honors Scholars Program
Honors College
and the
Honors-in Computing Program
College of Business and Technology
East Tennessee State University



Lucas Phillips

April 14, 2022

Date



Prof. Matthew Harrison, Thesis Mentor

April 14, 2022

Date



Dr. Ghaith Husari, Reader

April, 14, 2022

Date

Abstract

RISC-V is an open-source instruction set that has been gaining popularity in recent years, and, with support from large chip manufacturers like Intel and the benefits of its open-source nature, RISC-V devices are likely to continue gaining momentum. Many courses in a computer science program involve development on an embedded device. Usually, this device is of the ARM architecture, like a Raspberry Pi. With the increasing use of RISC-V, it may be beneficial to use a RISC-V embedded device in one of these classroom environments. This research intends to assist development on the SiFive HiFive1 RevB, which is a RISC-V embedded device. This device was chosen because of its ease of use, functionality-rich API, and affordability. In order to make developing with this board very approachable for a student, this research involved the development of a small suite of tools. These tools support common functionality like: building a source file into an executable ELF file, converting that ELF executable into an Intel HEX executable format that is required to run on the device, uploading the Intel HEX executable onto the device, and attaching a debug session to the program that is running on the device. With the help of this toolchain, developing on this RISC-V embedded device should be very approachable for most students.

Contents

Abstract	2
1 Background.....	4
1.1 What is RISC-V?.....	4
1.2 The Importance of an Open-source ISA	4
1.3 Momentum Behind RISC-V.....	5
2 Why the HiFive1 RevB Board?.....	6
2.1 Advantages of SiFive	6
2.2 Connecting to the Board.....	7
2.3 Standard Output with the Board.....	8
2.4 Uploading and Debugging	8
2.5 The Freedom Metal API.....	9
2.6 Board Affordability	10
3 Toolchain.....	10
3.1 Installing the Toolchain.....	11
3.2 Building an ELF Executable	12
3.3 Converting an ELF Executable to an Intel HEX Executable	13
3.4 Uploading an Intel HEX Executable to the Board	14
3.5 Debugging a Program on the Board.....	14
4 Opportunities for Future Development.....	14
5 Conclusion.....	15
6 References	16

1 Background

RISC-V is an open-source instruction set architecture that has been in development for over a decade (RISC-V International, n.d.). It has been steadily gaining a market over the past few years and is projected that RISC-V processors will have a revenue of more than a billion dollars by 2025 (Stewart, Kodama, Bucaille, & Crossan, 2021). Also, with support from large chip manufacturers like Intel, widespread adoption of RISC-V is clearly on the minds of one of the largest chip manufacturers in the world (Morra, 2022).

1.1 What is RISC-V?

RISC-V is a free and open-source instruction set architecture, or ISA. (RISC-V International, n.d.). Development of RISC-V began in May of 2010 at the University of California, Berkeley. After several years of development, the RISC-V Foundation was created to help to foster a community around and drive early adoption of the ISA. In March of 2020, the RISC-V International Association was incorporated in Switzerland and is the organization that manages the RISC-V ISA to this day (RISC-V International, n.d.).

1.2 The Importance of an Open-source ISA

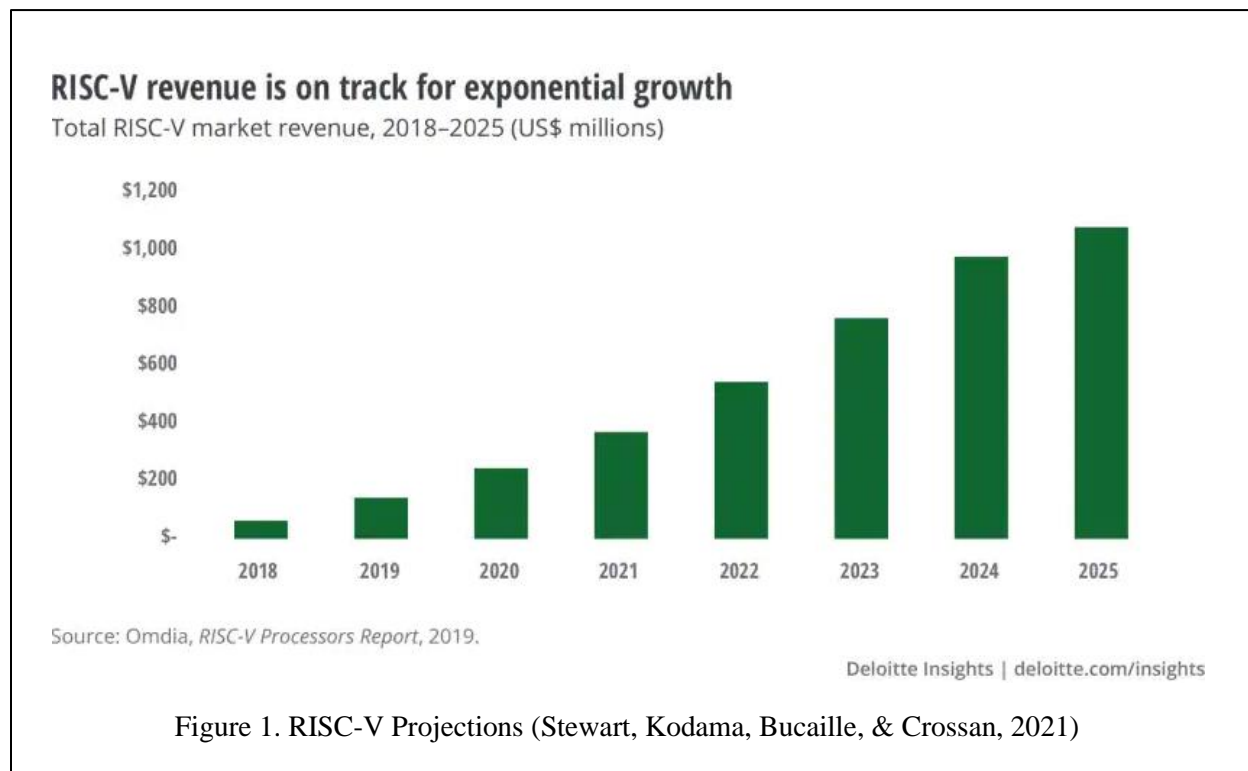
Currently, two of the most popular ISAs are ARM and Intel's x86 architecture. However, these two architectures have one major thing in common: they both require a manufacturer to purchase a license in order to develop with that ISA, leading to every board ARM or x86 computer manufactured incurring an additional licensing cost. In contrast, RISC-V is an open source platform that is free for any manufacturer to use.

Since RISC-V is open source, manufacturers can add or extend the existing ISA's functionality to meet their needs (Cording, 2021). There is space reserved within the base RISC-V ISA for any custom instructions that a manufacturer may want to incorporate. Also, in the

spirit of open source, if the manufacturer deems that their change could be beneficial to the architecture as a whole, they could consider becoming a member of RISC-V International and proposing their changes to the organization (RISC-V International, n.d.).

1.3 Momentum Behind RISC-V

CPUs that use RISC-V had a revenue of less than \$200 million in 2019, and, in 2021, they had a revenue of over \$400 million (Stewart, Kodama, Bucaille, & Crossan, 2021). Deloitte Insights projects that RISC-V will continue this growth for the next few years, with them projecting a revenue of over \$1 billion in 2025. RISC-V is here to stay and is only going to continue to chip away at a market that has been controlled by x86 and ARM for years.



In addition to general momentum that has been growing around RISC-V, the ISA is also looking to receive a bump from one of the largest players in chip manufacturing: Intel. In February of 2022, Intel announced a plan to try increase domestic production of chips with

several large investments (Morra, 2022). Part of this investment strategy is a \$1 billion investment into companies that are trying to strengthen the ecosystem around RISC-V and encourage its adoption (Morra, 2022).

2 Why the HiFive1 RevB Board?

The HiFive1 RevB was selected for this research because of its simple hardware interface, convenient API, reasonable price, along with other benefits. The manufacturer of the board, SiFive, provides a lightweight API, Freedom Metal, that enables bare-metal application development (SiFive, 2019). In addition, the board is also equipped with Segger J-Link, which allows for easy debugging (SEGGER, n.d.). Uploading programs, debugging, and powering the board can all be accomplished with a simple micro-USB cable. Finally, the board is priced similarly to other embedded devices like the Raspberry Pi and, because of the benefits previously listed, makes itself a very intriguing option for students and courses that rely on an embedded device for labs and exercises.

2.1 Advantages of SiFive

SiFive was founded by the inventors of RISC-V (SiFive, n.d.). The company was founded in 2015 as a way for the creators of RISC-V to pair their open source architecture with custom boards and software that make developing for RISC-V much easier (SiFive, n.d.). SiFive's close relationship with RISC-V is a major benefit to, not only the boards that they develop, but also to the software and tools that the company develops to make working with their boards much easier. SiFive produces several tools for working with their boards like a GNU embedded toolchain, the Freedom E SDK, and the FreedomStudio IDE.

The GNU embedded toolchain that is provided by SiFive provides many of the common tools that developers of low-level software will be familiar with. The toolchain contains all of the

tools that SiFive has determined are necessary to compile and debug programs on their boards. This includes tools like *gcc*, *g++*, *gdb*, *ld*, *objcopy*, *objdump*, and many more (SiFive, n.d.). This provides developers with an easy source for all of the tools that they should need to work with their board.

If a developer is looking for some useful scripts and example programs for learning about developing with the board, SiFive provides the Freedom E SDK. This SDK, which currently only supports development in Linux environments, removes the need for the developer to even use any of the toolchain tools directly. The SDK handles building, uploading, and debugging with a few commands that accept command line parameters for selecting the program, the target board, and the configuration. In addition, the SDK also includes over 40 example programs that show how to use certain features of the Freedom Metal API and of their board.

While the Freedom E SDK simplifies much of the development process, SiFive provides another tool to tenable student software development against their platform. The FreedomStudio IDE provides all of the features that were present in the SDK, but wraps them in SiFive's own version of the Eclipse IDE. This IDE is available for Windows, Linux, and MacOS and provides simple user interfaces for tasks like building, running, and debugging. It handles standard input and output as well through an integrated serial terminal. It also includes the same example programs as the SDK but with added configuration options while creating a project. The FreedomStudio IDE is truly SiFive's way of making developing with their boards as easy of a process as possible.

2.2 Connecting to the Board

One significant benefit of the HiFive1 RevB over something like a Raspberry Pi is that all interactions with the board can be accomplished with a single cable. The included micro-USB

cable runs from the board to the user's host machine and can handle operations like upload, debugging, serial communications, and powering the board. A Raspberry Pi requires a USB to TTL serial cable adaptor to support serial communications. In addition, the HiFive1 RevB also includes a button to restart the board; developing on the Raspberry Pi without an OS requires frequent restarts of the board, which can only be accomplished by unplugging and plugging back in the power supply. The HiFive1 RevB simplifies communication and uploading programs to the board versus alternative platforms, like the Raspberry Pi.

2.3 Standard Output with the Board

Standard output for the board is handled using the a serial connection via the USB cable. In short, any application that can handle output for a serial connection should be able to show the program's output. Some examples of applications that will work for this purpose are Tera Term and PuTTY.

2.4 Uploading and Debugging

Uploading a program to the HiFive1 RevB is a very easy process and can be accomplished in a couple of ways. The first way to upload a program is to use the Freedom E SDK. The SDK has a simple *upload* command, which will upload the chosen program to the board. The other way to upload is, in fact, even easier than using the SDK. When the HiFive1 RevB is connected to a computer via the micro-USB cable, it will appear in the File Explorer as an external drive. To upload a new program to the board and trigger a restart, a user can simply drag a program that they have built to the external drive labeled *HiFive*. If the program cannot be successfully uploaded, then the board will revert back to the previously installed program and begin executing.

The HiFive1 RevB uses Segger J-Link to support debugging. To utilize this feature, a user must download Segger's suite of tools from their website. This will include both a GUI and CLI version of the J-Link GDB Server tool. After selecting the correct target board in the GUI application, a user can start up a GDB Server for the connected board. Also, the GUI application will provide the user with the command line parameters to use if the user wants to use the CLI application instead for better automation. After the server is running, the user can launch the *gdb* tool found in the GNU embedded toolchain and connect to the server from within *gdb* using the *target extended-remote* command with the local server passed as the command's argument. Once connected, the user can use *gdb* just as they would with a program running on their host machine.

2.5 The Freedom Metal API

The next major benefit of using the HiFive1 RevB is the ability to utilize the Freedom Metal API. The Freedom Metal API allows a developer to easy access to many CPU features and peripherals. There is standard documentation for all of the API's features on the API's GitHub website. In addition, there are more detailed developer guides for more general concepts like exception handling, interrupt handling, and standard I/O.

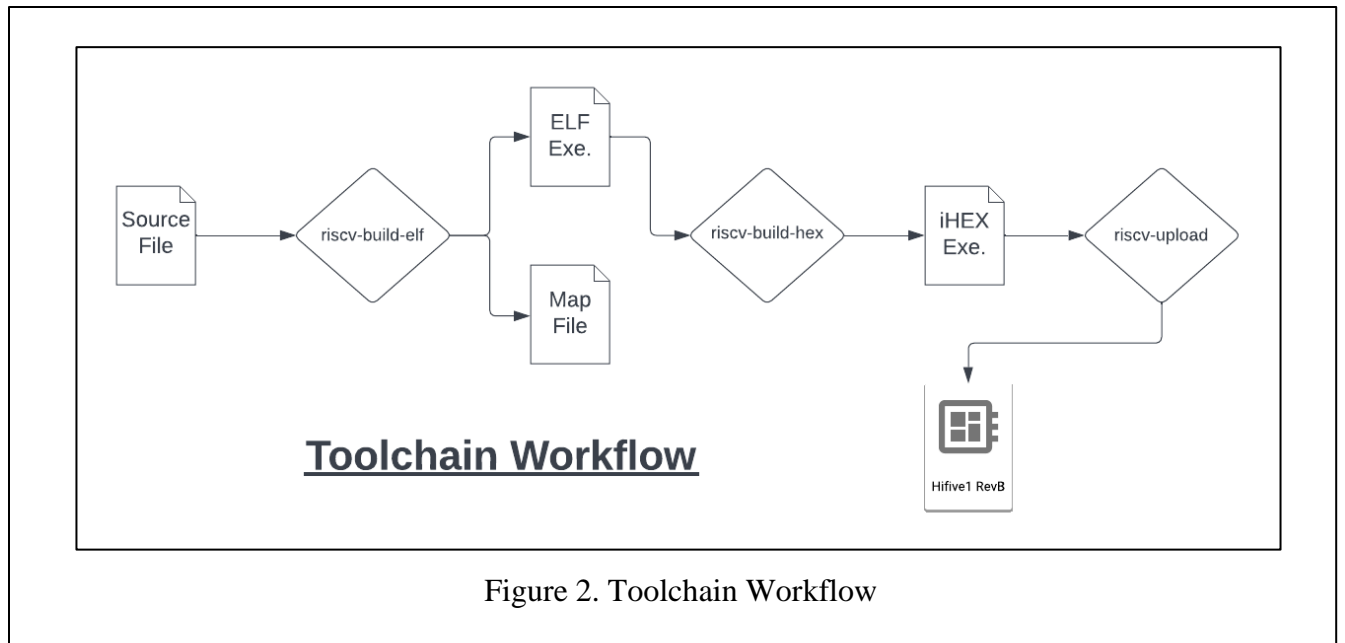
The Freedom Metal API provides access to several commonly used architectural features and communication interfaces. A developer can define metal constructors and destructors, which will execute before and after the program's *main* function, respectively. A developer can also toggle and set the board's integrated LED using functions from the API. A developer can also perform actions on any of the board's GPIO pins. The API provides functions to enable the L2 cache or set the L2 cache's associativity. In addition, there are functions and structs to support interrupts, time, locks, and many more features.

2.6 Board Affordability

One final noteworthy feature of the HiFive1 RevB is its price. The board currently costs only \$59 on SiFive's website. While there may be cheaper embedded devices, many do not have the ease of use and rich feature set that SiFive has provided here. And for a price that is cheaper than most college textbooks, it does not seem to be an expense that would hinder students from using it in a course.

3 Toolchain

While developing with the HiFive1 RevB can be done just by using the tools provided by SiFive, part of this research involved producing several scripts to simplify many of the common actions that users would have to perform while working on the device. PowerShell scripts have been developed for building an executable ELF file for a program, converting an executable ELF file to the Intel HEX format that is required to upload to the board, uploading a file to the board, and debugging the program that is currently running on the device. In addition, there is also an install script which simplifies the process of installing dependencies like the GNU embedded toolchain and the aforementioned scripts.



3.1 Installing the Toolchain

Almost all the tools that are needed to work with the HiFive1 RevB are available in the HiFive1 RevB Development Tools repository (<https://github.com/lucas-phillips/hifive1-revb-development-tools>). To install the tools and dependencies, first download the latest released ZIP file from the *Releases* page on GitHub. Once downloaded, simply extract the archive and open a PowerShell terminal from the new folder that was created. From there, run `./install.sh` and pass it the location you want to install to as the only argument. For example, one could run something like: `./install.sh C:/RISCV-Tools`. This will create the directory if needed and install the tools under it. It will also add the tool scripts to the *Path* environment variable so that they can be referenced from any working directory. This installation process will install the PowerShell scripts, the GNU embedded toolchain, and a version of the Freedom Metal dependencies that is built for the HiFive RevB. Finally, to support debugging on the board, you need to download SEGGER's suite of J-Link tools from their website. Figure 3 shows how this command may execute on a user system.

```
hifive1-revb-development-tools x + v
hifive1-revb-development-tools.git | main | +1 | ./install C:/RISCV-Tools

Directory: C:\

Mode                LastWriteTime         Length Name
----                -
d-----            4/13/2022   9:59 PM             RISC-V-Tools
Folder was created

Directory: C:\RISC-V-Tools

Mode                LastWriteTime         Length Name
----                -
d-----            4/13/2022   9:59 PM             freedom-metal
d-----            4/13/2022   9:59 PM             riscv-toolchain
d-----            4/13/2022   9:59 PM             riscv-scripts

hifive1-revb-development-tools.git | main | +1 |
```

Figure 3. Toolchain Installation

3.2 Building an ELF Executable

This script executes the GNU embedded toolchain's version of the *gcc* command. While it only executes one command, when running *gcc* in FreedomStudio or the Freedom E SDK, the tools pass about 25 flags to the command, so it would be inconvenient for the user to type those flags out for every execution. This script executes the *gcc* command with all of the correct flags for a program on the HiFive1 RevB and only requires the user to input a source file and an output file. The user can optionally also pass any other flags or command line parameters that they want to pass directly to the *gcc* command. An example of this command could be: *riscv-build-elf -sourceFile input.c -outputFile output.elf* (as seen in Figure 2). Figure 4 shows how this command may execute on a user system.

```
ThesisWorkspace x + v
ThesisWorkspace riscv-build-elf -sourceFile sifive-welcome.c -outputFile sifive-welcome.elf
ThesisWorkspace ls

Directory: C:\Users\lucas\OneDrive\Documents\ThesisWorkspace

Mode                LastWriteTime         Length Name
----                -
-a---             10/29/2021  2:02 AM           4613 C sifive-welcome.c
-a---             4/13/2022  10:03 PM          326032 sifive-welcome.elf
-a---             4/13/2022  10:03 PM          338547 sifive-welcome.elf.map

ThesisWorkspace |
```

Figure 4. Build ELF Executable

3.3 Converting an ELF Executable to an Intel HEX Executable

This script executes the GNU embedded toolchain's version of the `ObjCopy` command. In order to install a program onto the HiFive1 RevB, the program must be in the Intel HEX format rather than the ELF format that the `gcc` command produces. Converting the program to this format is easily accomplished using the `objcopy` command and passing it an output target of `ihex`. While this is a fairly simple process, a script is provided to make it even easier. Simply pass the input and output files to the command. For example: `riscv-build-hex -sourceFile input.elf -outputFile output.hex` (as seen in Figure 2). Figure 5 shows how this command may execute on a user system.

```
ThesisWorkspace x + v
ThesisWorkspace riscv-build-hex -sourceFile sifive-welcome.elf -outputFile sifive-welcome.hex
ThesisWorkspace ls

Directory: C:\Users\lucas\OneDrive\Documents\ThesisWorkspace

Mode                LastWriteTime         Length Name
----                -
-a---             10/29/2021  2:02 AM           4613 C sifive-welcome.c
-a---             4/13/2022  10:03 PM          326032 sifive-welcome.elf
-a---             4/13/2022  10:03 PM          338547 sifive-welcome.elf.map
-a---             4/13/2022  10:05 PM           94875 sifive-welcome.hex

ThesisWorkspace |
```

Figure 5. Build iHEX Executable

3.4 Uploading an Intel HEX Executable to the Board

As previously mentioned, there are a few ways to upload a program to the HiFive1 RevB board. The easiest method is likely to simply copy the file onto the HiFive external drive in the File Explorer. However, if the user would prefer to remain in the command line, they can utilize the upload script to accomplish the same thing. They only need to specify the drive letter that is associated with their board and the file they would like to upload. For example: *riscv-upload -sourceFile upload.hex -driveLetter D* (as seen in Figure 2).

3.5 Debugging a Program on the Board

Firstly, make sure you have installed SEGGER's suite of J-Link tools from their website in order to debug the board. This script launches the J-Link GDB Server CLI application in a separate process and passes it the parameters necessary to debug on the HiFive1 RevB. Then, it will launch *gdb* from the GNU embedded toolchain and automatically connect it to the default port on localhost where the server should have launched. If the server launched on something other than the default port, the user may have to manually run: *target extended-remote localhost:<port>* from within their *gdb* session. Once within the *gdb* session, the user should be able to do any debug actions that they would be able to do from a *gdb* session of a program running on their machine. The user may also want to consider loading the symbols for the program by calling the *gdb* command *symbol-file* and passing it the ELF file that was generated.

4 Opportunities for Future Development

One major improvement that could be made to the toolchain is to rework the installation process. Currently, the GitHub repository includes two large zip files that are required for the toolchain to work. These files are currently stored using Git LFS, but GitHub has applied some

limitations to Git LFS which mean that this current implementation will not scale very well if the repository is cloned often. There are a few ways to achieve this, but the most straight forward may just be to host those zip files at another location and ensure that the user downloads them before the run the install script.

Another area of improvement would be to manage source files that are not executables. Currently, the toolchain does not support linking any object files with the executable source code. It would be beneficial to avoid requiring the user to put all of their source code into one file. This would likely only involve adding an extra flag to accept any other files that need to be linked and including those files in the script's invocation of *gcc*.

5 Conclusion

While RISC-V has grown a lot in popularity over the past few years, RISC-V devices still lack many of the conveniences that ARM devices like a Raspberry Pi enjoy. The toolchain developed alongside this research intends to make developing on a RISC-V device as approachable to a student, or any user, as possible. Investments from large chip manufacturers like Intel indicate that now is a perfect time to start getting students accustomed to RISC-V, so that they will be comfortable when they see it after graduation.

6 References

- Cording, S. (2021, April 5). *What Is RISC-V?* Retrieved from Elektor Magazine:
<https://www.elektormagazine.com/articles/what-is-risc-v>
- Giorgi, R., & Mariotti, G. (2019). WebRISC-V: A Web-Based Education-Oriented RISC-V Pipeline Simulation Environment. *Proceedings of the Workshop on Computer Architecture Education*, 1-6. doi:<https://doi.org/10.1145/3338698.3338894>
- Greengard, S. (2020). Will RISC-V revolutionize computing? *Communications of the ACM*, 63(5), 30-32. doi:<https://doi.org/10.1145/3386377>
- Morra, J. (2022, February 14). *Intel Launches \$1 Billion Fund to Build Foundry Ecosystem, Backs RISC-V*. Retrieved from Electronic Design:
<https://www.electronicdesign.com/technologies/embedded-revolution/article/21216435/electronic-design-intel-launches-1-billion-fund-to-build-foundry-ecosystem-backs-riscv>
- RISC-V International. (n.d.). *About RISC-V*. Retrieved from RISC-V: <https://riscv.org/about/>
- SEGGER. (n.d.). *J-Link Debug Probes by SEGGER*. Retrieved from SEGGER:
<https://www.segger.com/products/debug-probes/j-link/>
- SiFive. (2019). *Freedom Metal*. Retrieved from SiFive: <https://sifive.github.io/freedom-metal-docs/introduction.html>
- SiFive. (n.d.). *About SiFive*. Retrieved from <https://www.sifive.com/about>
- SiFive. (n.d.). *Freedom Tools*. Retrieved from GitHub: <https://github.com/sifive/freedom-tools>
- Stewart, D., Kodama, E., Bucaille, A., & Crossan, G. (2021, December 1). *RISC-y business: Could open chip standard RISC-V gain traction against dominant incumbents?* Retrieved

from Deloitte: <https://www2.deloitte.com/xe/en/insights/industry/technology/technology-media-and-telecom-predictions/2022/risc-v-open-source-cpu.html>