RISC-V Console: A Containerized RISC-V Based Game Console Emulator for Education

Christopher Nitta
Department of Computer Science
University of California, Davis
Davis, CA, USA
cjnitta@ucdavis.edu

Aaron Kaloti
Department of Computer Science
University of California, Davis
Davis, CA, USA
apksingh@ucdavis.edu

Shuotong Wang
Department of Electrical and Computer Engineering
University of California, Davis
Davis, CA, USA
vstwang@ucdavis.edu

ABSTRACT

The rapid transition to online education due to the COVID-19 pandemic left many instructors needing to redesign their course projects as students no longer had access to physical hardware. This paper describes the development of an open-source containerized RISC-V based game console emulator that replaced physical hardware for use in course projects. The tool was initially designed and used in a graduate operating systems course and then subsequently used in a lower division computer organization and machine-dependent programming course. The container provides a full toolchain with gcc compiler, RISC-V game console emulator with integrated debugger, example program, and input recording/auto-run tool designed for auto-grading. The use of a container reduced the barrier to entry for the students allowing them to get up and running in a relatively short period of time. Given the successful deployment of the tool in the previous courses, the tool was used both again in the lower division course and in the upper division undergraduate operating systems course this past fall.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Computational science and engineering education**; • **Computing methodologies** → **Simulation tools**.

KEYWORDS

RISC-V, emulator, autograder

ACM Reference Format:
Christopher Nitta, Aaron Kaloti, and Shuotong Wang. 2022. RISC-V Console: A Containerized RISC-V Based Game Console Emulator for Education. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 1 (ITiCSE 2022), July 8–13, 2022, Dublin, Ireland*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3502718.3524791

1 INTRODUCTION

The rapid transition to online education due to the COVID-19 pandemic left many instructors needing to redesign their course projects as students no longer had access to physical hardware on campus. Given the availability of relatively inexpensive hardware such as the Rasperry Pi one might expect an instruction could request the students to purchase their own hardware; the fact that students were spread throughout the world during the pandemic removed the student purchase of hardware as a viable option. Many full system emulators exist such as QEMU [1], VirtuaBox, and even full system simulators such as gem5 [7] are available; however, each system comes with their disadvantages, such as speed of simulation, lack of desired I/O, etc. It was the challenges presented from online education combined with a lack of an ideal solution that led to the development of the RISC-V Console Emulator (RVCE).

Since the development of the RVCE it has been used in multiple offerings of a lower division computer organization and machine-dependent programming course as well as upper division and graduate level operating systems courses at our institution. Currently, there are plans to continue using the RVCE in future offerings of the courses. The details of the RVCE design and capabilities are provided in this paper. The remainder of this paper is organized as follows. Section 2 provides some background on the rationale for the development of the RVCE, which is followed by the work most related to this effort in Section 3. A description of the RVCE design and features appears in Section 4. Section 5 discusses course experiences with the RVCE. We conclude in Section 6 and provide comments on planned feature improvements.

2 BACKGROUND

In preparing for ECS 251 Operating Systems, we looked to revamp a term long group project using a Rasperry Pi with one that used only software tools and could be completed individually. The project was reworked from having a group of students develop an OS and network connected application to a project where individual students would design their own OS for a theoretical cartridge based game console. In an effort to promote open systems and to align with other courses offered in the department we decided that the emulated processor should be RISC-V [12] based. RISC-V was designed with research and education in mind, but is growing in industry use. Faculty in our department have been adopting RISC-V for their courses when appropriate as the base Instruction Set Architecture (ISA) is small and real systems are being built using the RISC-V ISA. Initially QEMU[1] was planned for the basis of the game console emulator; however, after further investigation this plan was abandoned in favor of developing emulated hardware from scratch. The design of QEMU (as well as other emulators) is to support more general purpose systems that have Memory

ITiCSE 2022, July 8–13, 2022, Dublin, Ireland.

ACM ISBN 978-1-4503-9201-3/22/07.
https://doi.org/10.1145/3502718.3524791

Management Units (MMUs) and to support real existing hardware. We desired to have a system that only supported machine mode, and had non-standard hardware such as a palletized video controller. The desire to support non-standard simplified hardware was driven by a concern that real hardware would likely push students to just design a simplified Linux in order to be able to reuse existing open-source drivers. In addition, since the system was designed to have a single application running at a time removing the MMU would simplify the OS implementation, a concern due to the shorter 10-week term at our institution.

3 RELATED WORK

The work that is most related to RVCE that we are aware of differ from our work in one of several ways: it emulates a processor other than RISC-V, it was not designed with the intent of primarily being educational use, or its design has a different purpose. There have been many emulators designed and used over the decades; however, we will briefly discuss the differences between the μARM [9] and μMPS [4–6] line of work. The μARM emulator is an ARM7tdmi-based system emulator/architecture that was designed for the purpose of being appropriate for undergrad education. JaeOS [9] was presented along with μARM as a system framework students to work on their own operating systems. Kaya [6] is an OS designed for teaching OS and runs on μMPS a MIPS R3000 emulator. Support for multiple processors was added to μMPS2 [5], and more recently the μMPS3 [4] (a non-backward compatible redesign) was released along with the publication of the Pandos Project. RVCE differs from μARM and μMPS most obviously in that they emulate ARM and MIPS respectively as opposed to RISC-V, but also RVCE is focused on more of an embedded system instead of a general purpose machine.

As stated previously QEMU/RISC-V [10] was considered as the basis for the emulator. The existing support for RISC-V on QEMU is focused on the products available from SiFive Inc. Since the existing configurations were designed with the actual hardware boards in mind, we believed that modifying QEMU/RISC-V to meet our vision would have been a greater development effort than our ground up solution. The RISC-V support on QEMU was also not designed with students being the expected user (unlike the μMPS work). The RVCE was designed with students in mind.

There are at least two other RISC-V emulators/simulators designed for education; however, the two that we are aware of WebRISC-V [3] and DINO CPU [8] are not suitable for needs of ECS 251. WebRISC-V is a web-based pipeline simulation of a RISC-V implementation. The microarchitectural details provided by WebRISC-V are unnecessary for needs of ECS 251, and are better suited for a computer architecture course. Along the lines of WebRISC-V, the DINO CPU is a pipelined implementation of RISC-V designed for teaching computer architecture courses. While the DINO CPU has a single cycle version, the implementation in Chisel and the amount of simulated detail of the DINO CPU makes the performance of emulating a system similar to RVCE infeasible.

4 RISC-V CONSOLE EMULATOR DESIGN

The RVCE was designed with the goal of providing a cross-platform compatible game console emulator with students being the primary user. The RVCE is completely open-source and is available at https://github.com/UCDClassNitta/riscv-console. Considering that the RVCE was designed to be used by students to develop software, a full development toolchain was also necessary to be packaged in with the repository. This remainder of this section describes the RVCE emulated hardware, the container and toolchain packaging, and the auto-grading support.

4.1 Hardware Emulator

The RVCE is designed to compile and run on a Linux environment that supports GTK+3 [11]. The RVCE code is written in C++ using the 2014 standard and gtk+-3.0 is the only package dependency at this time. While the RVCE source may be compiled directly on OS-X, Windows, Cygwin or other systems if the appropriate packages are installed, it will compile and run within the Docker container described later in Section 4.2. The RVCE can be run in one of two modes, either normal or debug mode. In normal mode, the RVCE provides only the video display, controller buttons, power/reset buttons, and buttons to load firmware and cartridge Executable and Linkable Format (ELF) files. In debug mode, the RVCE also provides displays for CPU registers, Control and Status Registers (CSRs), instruction disassembly, and memory window. Figure 1 shows a screenshot of the RVCE in debug mode. The black box with "`Hellow World!X`" in it in the upper left is the video display, directly below it are the buttons for the multi-button control pad (`w`, `x`, `a`, `d` for up, down, left, and right, and `u`, `i`, `j`, `k` for buttons 1 - 4), and the power/reset and load buttons. Figure 2 illustrates what the multi-button controller might look like if fabricated; the letters on the RVCE buttons are the currently mapped keys so that the keyboard keys can be utilized as buttons instead of requiring the mouse the press the buttons. The left side of the window constitutes what is displayed in normal mode. The addition debug section on the right is only available in debug mode. The CPU registers are displayed at the top right-hand side of the window with the instruction disassembly, and CSRs below. The memory window is on the bottom right with buttons to quickly reposition to the base of the firmware (`FW` button), cartridge (`CTR` button), chipset (`CS` button), video memory (`VID` button), global pointer (`GP` button) or the top of the stack (`SP` button). In debug mode breakpoints can be set/cleared by double-clicking on an instruction in the disassembly window. The `Run` toggle button will run the system, this button is also responsible for stopping the system when it is running. The debugger allows for single stepping via the `Step` button, and also allows for recording of all controller inputs via the `Record` button. Recording of inputs was designed with auto-grading in mind, auto-grading is discussed further in Section 4.3.

4.1.1 RISC-V Processor. The main emulated processor of the RVCE is a RISC-V RV32EM [12]. The processor is the same as the base RV32I except that it has only 16 registers instead of 32. In addition, the processor supports the M standard extension Integer Multiplication and Division, and the ZiCSR Control and Status Register extensions. The RVCE supports only Machine privilege mode, so that "game" will be able to directly access all emulated hardware. Currently, the RVCE decodes and executes each instruction for the RV32EM and is capable of emulating in excess of 20MHz on

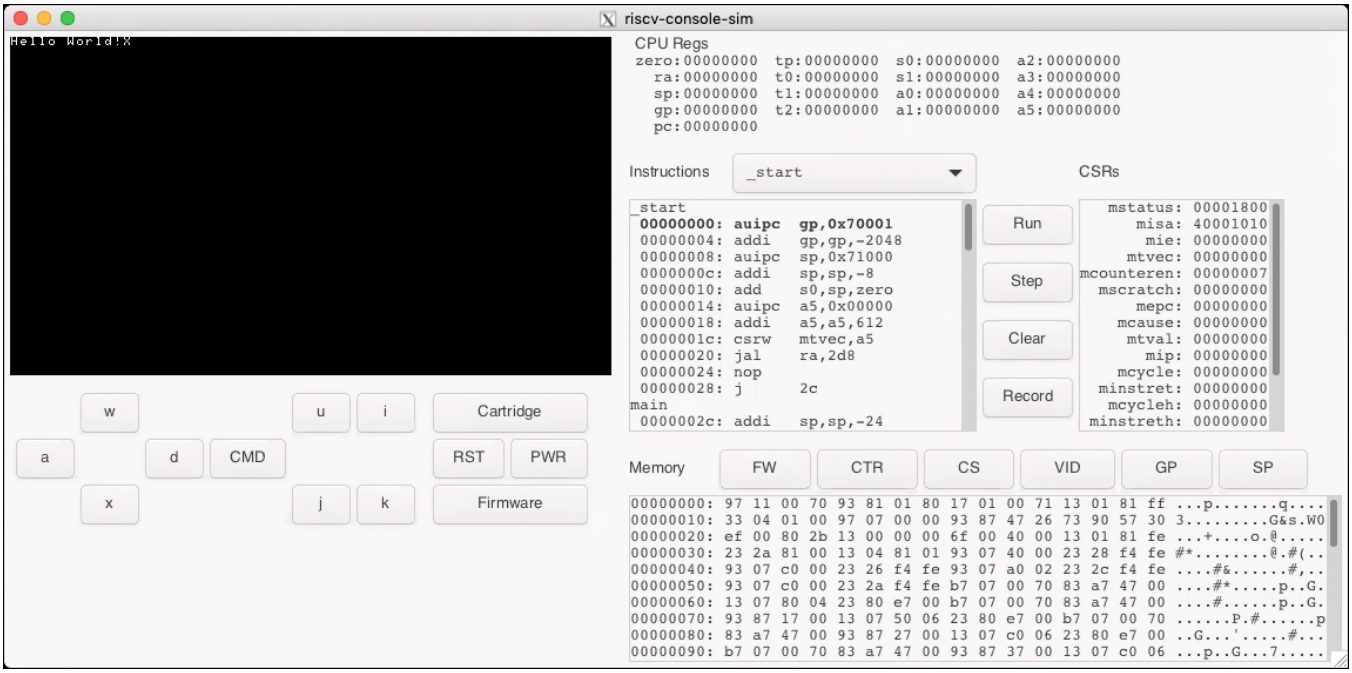

Figure 1: Screenshot of RISC-V Console Emulator in Debug Mode

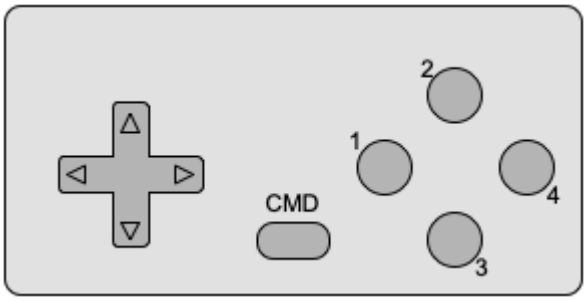

Figure 2: Theoretical Emulated Multi-button Controller

Table 1: RISC-V Console Emulator Memory Layout

Base Address	Size	Description
0x00000000	0x1000000 (16MiB)	Firmware Flash
0x20000000	0x1000000 (16MiB)	Cartridge ROM
0x40000000	0x40 (64B)	Chipset Registers
0x50000000	0x100000 (1MiB)	Video Controller
0x70000000	0x1000000 (16MiB)	RAM

a 2.4Ghz Intel Core i9-9980HK. While the 2.4Ghz Intel Core i9-9980HK may be a relatively high performance processor by today's standards, in our experience students did not have issues with emulation speed on their machines. Dynamic binary translation could dramatically increase the emulated speed; however, for the purposes of the instruction the current emulated speed is more than enough to maintain real-time performance.

4.1.2 Memory. The main components of the memory mapped emulated hardware are the flash memory, RAM, chipset, cartridge and video controller. Table 1 shows the layout of memory within the emulated system. The firmware flash, cartridge ROM, and main memory RAM are all 16MiB in size; while RAM can contain the entirety of either the flash or ROM, execution from RAM is disallowed. The firmware flash is designed to be accessed in a single cycle, so migration of instructions to RAM is not necessary. The game cartridges consist of ROM and will hold the "game" application on them. When inserted into the system (ELF file is loaded) the ROM is mapped directly into memory and can be read in a single cycle just as flash memory can. The insertion or removal of a cartridge can generate an interrupt through the chipset, and the current status can be determined through a memory mapped register.

4.1.3 Chipset. The console has a chipset that controls the interrupts and system timer as well as provides access to read the status of the controller buttons. Table 2 shows the layout of registers within the chipset. The Machine Time and Machine Time Compare are 64 bit registers (mtime and mtimecmp) defined in the RISC-V Privileged Spec [13]. The Cartridge Status register specifies if the cartridge is inserted or not, and contains the entry point to the cartridge application; this means that the ELF e_entry [2] can be used instead of requiring that the entry point always be at a fixed location. The console chipset has two integrated DMA channels that are each capable of transferring up to 16,777,215 bytes in a transfer request. Once initiated the DMA channel will transfer up to 32-bits per CPU cycle until the transfer has been completed. In addition to the DMA channels, the Machine Time clock as well

Table 2: Chipset Memory Layout

Base Address	Size	Description
0x40000000	0x4 (4B)	Interrupt Enable Register
0x40000004	0x4 (4B)	Interrupt Pending Register
0x40000008	0x8 (8B)	Machine Time
0x40000010	0x8 (8B)	Machine Time Compare
0x40000018	0x4 (4B)	Controller Status Register
0x4000001C	0x4 (4B)	Cartridge Status Register
0x40000020	0x20 (32B)	DMA Registers
0x40000024	0x4 (4B)	Machine Clock Period Register
0x40000028	0x4 (4B)	Video Clock Period Register

as the video controller clock period can be controlled through the chipset registers.

4.1.4 Video Controller. The video controller is responsible for rendering the graphics for the console. The video controller has 1MiB of memory and renders a 512 × 288 (16:9 aspect ratio) screen. The video controller has two modes: a text mode and a graphics mode. The default of the video controller is to start in text mode that provides 64 × 36 characters each of 8 × 8 pixels. The video controller has a built-in MSX font that is loaded into the font memory upon reset. The text mode is capable of rendering up to 256 different characters, but by default the MSX font only supports the printable ASCII characters from '!' to '~'. The graphics mode provides support for five "background" full resolution images, 64 large sprites (each up to 64 × 64 pixels in size), and 128 small sprites (each up to 16 × 16 pixels in size). All images support 256 unique colors from a 32-bit RGBA palette. There are four background palettes and four sprite palettes, so all images displayed do not need to share a global palette.

4.2 Container and Toolchain

The RVCE and toolchain have been set up to run within a Docker container. The decision to containerize the project was driven by a desire to have students up and running with very few commands; in addition, containerizing the toolchain allows for a common platform for all students to work within. In order to get up and running one will need Docker, X-11 support and a bash shell on their machine. PowerShell on Windows is also supported for those that don't have a bash shell. Using a single bash script `rvconsole.sh` (or `rvconsole.ps1` for PowerShell) one can start the entire process. During the first run of the script, the base RISC-V Docker image `riscv_base` that has the RISC-V build toolchain will be pulled. This process can take a noticeable amount of time, potentially on the order of minutes as it is pulling several GB; fortunately, this should only have to be done once. Once the base image is pulled the RISC-V development environment Docker image `riscv_console_dev` will be built. The `riscv_console_dev` image builds upon the `riscv_base` image and should build quickly. The `riscv_console_dev` image adds the necessary packages for building the RVCE within the container. Once the `riscv_console_dev` image is built the script will launch a container name `riscv_console_run`. When the container is launched the current directory is mounted as the `/code` directory of the container allowing for source files to be edited on the host machine as well as within the container. Once the container is up and running, the user will be a bash shell prompt within the container and can exit using `exit` command. Subsequent runs of `rvconsole.sh` will restart the `riscv_console_run` container assuming no changes have been made to the Docker files.

The Docker container that is created with `rvconsole.sh` script is designed to have all packages necessary to compile the RVCE. The script is also designed to allow X11 forwarding out of the container so that the RVCE can be compiled and run inside the container. Once at the `/code#` prompt, one can launch the RVCE by running the `runsim.sh` bash script within the container. If the RVCE has not already been built, the script will build the application.

4.3 Auto-grading Support

The RVCE supports auto-grading using two main parts: input recording, and auto-running. The purpose of the input recording allows for an instructor to run a working example, and to record the inputs while running the example. The auto-runner allows for running the RVCE using inputs that were previously recorded, this allows for testing of students code.

4.3.1 Input Recording. The first is the recording of input described earlier in Section 4.1. The recorder is integrated into the RVCE debug mode and can be activated by clicking the `Record` toggle button. Once pressed, the RVCE will be switched into recording mode and will store all button actions along with the absolute cycle number since `Run` toggle button was pressed. When the `Record` toggle button is depressed again the user will be prompted of where to save the file. The input recording is output in JSON format.

4.3.2 Auto-Runner. Auto-Runner allows for the RVCE to run on previously recorded inputs. The auto-runner does not launch the GUI, but runs as a command line program only. The auto-runner takes in two optional arguments: input JSON file path, and output JSON file path. The auto-runner program by default will use `input.json` and `output.json` file in current directory if no argument is specified. In addition to supporting all inputs, the auto-runner also supports outputting the status of registers and memory. While it is possible to manually construct an input JSON file since they human-readable, we envision that most instructors would opt to record input and potentially make small alterations if needed. The ability to output registers, and sections of memory (including video memory) allows for instructors to compare the resulting state of the RVCE after running a student's program without needing to manually load each student's program through the RVCE GUI one at a time.

5 COURSE EXPERIENCE

As stated previously while preparing for ECS 251 we looked to replace the group project that used actual hardware with one that could be completed individually. The RVCE was developed so that students in ECS 251 could design their own embedded OS to support future cartridge based games. Despite development of RVCE continuing into the term in which it was first being used, we found that the development environment turned out to be quite reliable

considering the wide range of systems students were using. In addition, the feedback from the students were quite favorable with course evaluations showing comments such as “Project was a great way to get us to understand OS”, “Project helped me learn a lot about operating systems”, and “project is fun”.

Given the successful deployment of the RVCE in ECS 251, we decided to use it in ECS 50 Computer Organization and Machine-Dependent Programming for one of the assignments the following term. Initially the plan had been to cover more on RISC-V and to use the RVCE for multiple assignments; however, due to too much time being spent on x86-64 we ended up using RVCE for only a single I/O assignment. The students were tasked with making several modifications to the C code example provided in the repository. To make the modifications, the students had to sufficiently understand how the example program worked as they were tasked with changing the control from the direction buttons to the numbered buttons. We felt that it provided a great way for students to experience writing I/O related code in a “high-level” language.

This past fall we focused primarily on RISC-V and then switched to x86-64 later on in ECS 50. This change meant that the students received much more exposure to I/O and interrupts using the RVCE. The two initial RISC-V assignments were small, basic assignments that had the students write small programs or functions. The third RISC-V assignment utilized I/O and interrupts allowing for more exposure to hardware that students had received prior to the use of RVCE. Initially the plan was to have the students implement a game; however, the assignment was not quite ready in time for use this past fall. Given the flexibility of the RVCE, many games could be the basis for assignments and we plan to develop multiple for use in future courses.

In addition to using RVCE in ECS 50 this past fall, we also used it in our upper division undergraduate ECS 150 Operating Systems and Systems Programming course. The previous assignments that built a “VM” layer on top of Linux were reworked to run on the RVCE. The previous assignments utilized multiple processes in order to provide the illusion of hardware working in parallel; however, this approach limited the ability to autograde the assignments. The previous assignments also were limited in that they did not have the students interact with hardware. The reworked assignments for ECS 150 had the students develop support for threads, synchronization mechanisms, memory management of the heap, and ultimately support for the graphics hardware.

In both ECS 50 and ECS 150 there was a desire to increase the exposure to interfacing with hardware and to increase a knowledge of RISC-V. To assess the student’s perception of the RVCE we surveyed both courses about the assignments, the RVCE, and their confidence of working on real hardware. There were 90 and 115 respondents to the survey, corresponding to response rates of 79% and 85% of the enrolled students in ECS 50 and ECS 150 respectively. The survey questions utilized a five point Likert scale ranging from Strongly Disagree to Strongly Agree. Figure 3 shows the results of the survey statement “My knowledgeable of RISC-V has improved because of the assignments that used the RVCE.” Overall the students agreed that their knowledge increased because of the assignments using RVCE. Figure 4 shows the results for the survey statement “The RVCE aided in my understanding of RISC-V.” Like the previous statement the students agreed that the RVCE

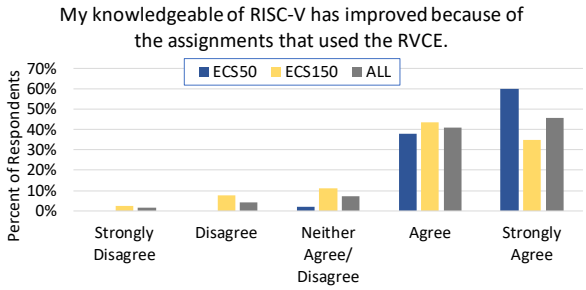

Figure 3: ECS 50 & ECS 150 Survey Results (Assignments)

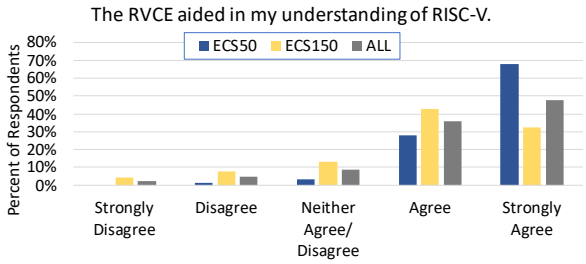

Figure 4: ECS 50 & ECS 150 Survey Results (RVCE)

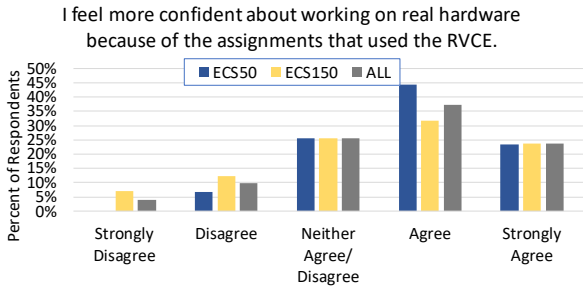

Figure 5: ECS 50 & ECS 150 Survey Results (Hardware)

aided in their understanding; however, the ECS 50 students found that it aided them more than the ECS 150 students. The fact that ECS 50 is a requirement for ECS 150 it is not necessarily surprising that ECS 150 students agreed less with the statement as they may have learned more about RISC-V during their ECS 50 course. Figure 5 shows the results of the survey statement “I feel more confident about working on real hardware because of the assignments that used the RVCE.” Students agreed that they felt more confident about working on real hardware; however, the strength of the agreement was not as strong as for the other two statements. The survey results indicate that the students generally believe that the assignments utilizing the RVCE increased their comfort with working on hardware and aided in increasing their knowledge of RISC-V.

6 CONCLUSIONS AND FUTURE WORK

The successful deployment of RVCE in multiple courses and spanning from lower division to graduate level leads us to believe that is a suitable for university level computer science education. As

stated previously we plan to continue using it in future offerings of our courses and have identified features for further expansion. The features that we have identified for future expansion and believe are worthy of mentioning here fall into three categories: increased auto-grading support, increased platform support, and additional emulated hardware.

The first and most likely highest priority feature that we would like to provide is to add a GUI dialog to specify any output commands. Currently, the output commands specified in the input JSON files must be added manually after recording. Being able to quickly select the registers and/or memory sections that the instructor would like to compare would greatly aid in the workflow. Along the lines of auto-grading support we believe that adding a Dockerfile and associated scripts to aid in Gradescope programming assignment integration would be greatly beneficial to the community. The Gradescope integration is likely slated to be developed during the fall term as the current plan is to utilize the RVCE in more courses. The last feature associated with auto-grading is to add support for outputting screen rendering in graphics mode. The actual image rendered to the screen can be accomplished in many ways, for example background 0 or 1 could be used with the other disabled and the resulting image would be the same with the contents of the video memory being vastly different. If an instructor is attempting to assess the contents actually displayed on the screen the current output capabilities would make that difficult. Additionally, we have considered if the screen rendering output should be as hex in the output JSON file, or if the image should be rendered to a PNG. We plan to further investigate this over the coming year.

Currently, Windows, OS-X, and Linux are supported through the use of a Docker container; however, Docker requires superuser access on Linux in order to run, likely preventing students from running it on shared instructional systems. We plan to add support through a Singularity container as well; this would provide the ability for students to launch the container on shared systems in which they have limited access rights. In addition, it could be highly beneficial to add support for a native GUI on Windows and OS-X so that students may directly install the RVCE.

The other features we would like to add are related to additional emulated hardware. The first piece of hardware we believe could be beneficial is a UART type device. By providing a UART interface that was connected to a standard in and out would provide users another debug interface that could be utilized during development. In addition, by having a UART type interface that is connected to the host system, users of the RVCE could in theory network instantiations and actually have multiplayer games. The final piece of emulated hardware that we see would improve the RVCE is to add audio support. Due to difficulties in connecting audio to Docker containers on some platforms, this remains a larger development effort that it may initially appear; however, we hope to eventually support this and the other features.

7 ACKNOWLEDGMENTS

We would like to acknowledge the 45 graduate students in ECS 251, 222 undergraduate students in ECS 50, and 136 undergraduate students in ECS 150 who were the first users of RVCE. Specifically we would like to thank Xiaoyi “Eric” Li who discovered and fixed so many of RVCE’s early bugs.

REFERENCES

[1] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *2005 USENIX Annual Technical Conference (USENIX ATC 05)*. USENIX Association, Anaheim, CA. https://www.usenix.org/conference/2005-usenix-annual-technical-conference/qemu-fast-and-portable-dynamic-translator

[2] TIS Committee. 1995 [Online].. *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification.* https://refspecs.linuxfoundation.org/elf/elf.pdf

[3] Roberto Giorgi and Gianfranco Mariotti. 2019. WebRISC-V: A Web-Based Education-Oriented RISC-V Pipeline Simulation Environment. In *Proceedings of the Workshop on Computer Architecture Education (WCAE’19)*. Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/3338698.3338894

[4] Mikey Goldweber, Renzo Davoli, and Mattia Biondi. 2021. The Pandos Project and the μMPS3 Emulator. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE ’21)*. Association for Computing Machinery, New York, NY, USA, 122–128. https://doi.org/10.1145/3430665.3456331

[5] Michael Goldweber, Renzo Davoli, and Tomislav Jonjic. 2012. Supporting Operating Systems Projects Using the μMPS2 Hardware Simulator. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE ’12)*. Association for Computing Machinery, New York, NY, USA, 63–68. https://doi.org/10.1145/2325296.2325315

[6] Michael Goldweber, Renzo Davoli, and Mauro Morsiani. 2005. The Kaya OS Project and the μMPS Hardware Emulator. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE ’05)*. Association for Computing Machinery, New York, NY, USA, 49–53. https://doi.org/10.1145/1067445.1067462

[7] Jason Lowe-Power, Abdul Mutaal Ahmad, et al. 2020. The gem5 Simulator: Version 20.0+. arXiv:cs.AR/2007.03152

[8] Jason Lowe-Power and Christopher Nitta. 2019. The Davis In-Order (DINO) CPU: A Teaching-Focused RISC-V CPU Design. In *Proceedings of the Workshop on Computer Architecture Education (WCAE’19)*. Association for Computing Machinery, New York, NY, USA, Article 2, 8 pages. https://doi.org/10.1145/3338698.3338892

[9] Marco Melletti, Michael Goldweber, and Renzo Davoli. 2015. The JaeOS Project and the μARM Emulator. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE ’15)*. Association for Computing Machinery, New York, NY, USA, 3–8. https://doi.org/10.1145/2729094.2742596

[10] SiFive. 2019 [Online].. *Simulating with QEMU.* https://sifive.github.io/freedom-e-sdk-docs/userguide/qemusimulation.html

[11] GTK Development Team. 2018 [Online].. *Gtk – 3.0: The GTK toolkit.* https://docs.gtk.org/gtk3/

[12] Andrew Waterman and Krste Asanović. 2019 [Online].. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA.* https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

[13] Andrew Waterman and Krste Asanović. 2019 [Online].. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture.* https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMFDQC-and-Priv-v1.11/riscv-privileged-20190608.pdf