ECE 544 Final Project

Chuck Faber and Brett Thornhill
Spring 2021

Project Overview

For this project, we tested the RVfpga-SoC curriculum, making comments on the instructions and testing the viability of the system as applied to ECE 544 in the future. Our comments were added directly on the Installation and Lab Instruction documents each in our own separate folders in the Git repo.

- ece544_final_project (root folder)
  - brett
    - RVfpgaSoC-May21_2021
      - RVfpgaSoC
        - Labs
          - LabInstructions
            - **Lab 1.docx**
            - **Lab 2.docx**
            - **Lab 3.docx**
            - **Lab 4.docx**
            - **Lab 5.docx**
        - **RVfpgaSoC Installation Guide.docx**
  - chuck
    - *Same as Brett's Folder*

A compiled list of our comments along with the associated documents and page numbers are also provided as an appendix to this report for easier review.

Both Brett and Chuck independently ran through Labs 1 through 5 in an attempt to get a feel for the RVfpga SoC system, and the additions of ZephyrRTOS, and TensorFlow.

Assumptions

In order to draw our conclusions about the potential for RVfpgaSoC use in ECE 544 in the future, we had to make the assumption that the course outcomes for ECE 544 remain the same in the future. In particular, that students will:

1. Gain an understanding of embedded system design methods: pwm detection, closed loop digital control, sensors, actuators, and displays.
2. Gain a basic understanding of programming real-time operating systems
3. Gain a basic understanding of common embedded system communication buses including I2C, SPI, and CAN.

4. Gain a basic understanding of networking and wireless networks
5. Practice writing and debugging embedded system applications and drivers
6. Practice designing implementing and testing SoC systems using the Xilinx platform development tool
7. Practice writing good technical documentation and well-structured code.

Impressions of Installation Guide

The installation guide was mostly clear, but there were cases of some instructions we were asked to do that didn't make sense. Better explanation of each step to the student making it clear why each step is being taken would be helpful. The student would have to have some prior experience with installing similar programs via command line if using Linux.

Impressions of Lab 1

This lab was by far one of the most tedious. Unlike with using the Microblaze core, every connection had to be manually wired, and there were hundreds and hundreds of connections to be made. The block diagram itself was difficult to traverse because of how much space it took up. The majority of the time was spent at this stage of the labs.

We'd recommend that the RVfpga program put some effort into getting the connection automator to work for their system. Or for instructors, to provide an already mostly connected system or fully connected system.

Hours were spent on this, and any small error made here would affect students greatly later in the future. The time spent manually connecting wires here at this stage could be better used learning how to expand the system and learning how to apply the concepts taught in the class to this system.

At the very least, the provided block diagrams which are zoomed in on particular portions of the overall block diagram and highlight the required connections are very useful.

Impressions of Lab 2

We ran the simulator and the hardware tests for this lab to generate a blinking LED or simulate a simple ALU operation. This lab was very similar to one we've done in ECE 540.

Overall this lab was simple and easy to accomplish should lab 1 have been completed correctly. However we believe several improvements can be made. The verilator simulation and the use of GTK wave could be more useful in 544 and 540, however its use wasn't really emphasized in ECE 540. I think this tool should be highlighted a little more as one possible option to debug projects and designs.

In addition, and this is a problem for several of the following labs, the blinking light is a nice little hello world, but there is no opportunity for expansion here. In ECE 540, the labs often had a set of tasks or challenges provided to students to extend the concepts of the lab further. Maybe implementing the use of the switches, or buttons. This opportunity is completely ignored. Once the student gets the light to blink, the lab abruptly ends.

Impressions of Lab 3

This lab apparently replaces what was done in Lab 1 by automatically running the connections we made manually in Vivado using FuseSoC. This kind of makes us question the purpose of creating a block diagram entirely. Especially as some of the benefits of working with the Vivado's IP Integrator with Microblaze are conspicuously absent for RVfpgaSoC.

For example, adding new IP blocks to the RVfpgaSoC design for PMODs and other peripherals is simply not allowed without major adjusting of the base Verilog code, likely similar to how we had to edit it in ECE 540. But if we have to manually edit the Verilog code in wb_interconnect, and on forward to the intcon_wrapper.v, like we did in 540, what is the point of using the IP Integrator at all.

It almost seems like using the IP Integrator in Vivado was an afterthought for this project, and is not intended to be used long term. Especially with how efficient and fast the FuseSoC building process is.

This begs the question of what does the instructor want to focus on teaching in ECE 544? Is learning how to use the Vivado IP Integrator and implement new and custom IP a major learning objective in the class? Because if so, RVfpga isn't the best tool to teach it. The student will be doing a lot of the same Verilog coding and connecting as we did in ECE 540.

Again the lab abruptly ends after getting a light to blink. This should be extended. Maybe having the student make changes to the RTL and show how efficiently FuseSoC can be used to generate these targets. The folder structure is very different than in Lab 1 so I'm unsure even how to edit this.

The OCD (Open On Chip Debugger) is introduced here. I would like to have learned more about how to use it and the possible commands. It seems to allow sending directly JTAG commands. This could be a useful debugging tool.

Impressions of Lab 4

Both Brett and Chuck ran into similar problems in this lab where it seemed that the first steps would not work because fusesoc couldn't recognize that the swervolf core was already installed or it assumed that the swervolf core was a dependency of the swervolf core, so perhaps it was a circular dependency issue.

We wondered if it was the later half of Lab 3 which might have caused that problem since after re-doing the first half, removing the SweRVolf folder and re-installing the fusesoc libraries seemed to fix it.

In this lab we implemented the ZephyrRTOS onto the SweRVolf SoC. Overall, it was straight forward. The simulation example was interesting. The hardware example again was just a blinking light. Like the other labs, this one needs to be extended and further challenges given. There were a few errors in the instructions given (i.e. they copy-pasted part of the earlier part lab and forgot to change some of the terminal commands).

Sample programs other than blinky require altering device tree files so it would be helpful if the lab had a section on how to update these files and explain what exactly you are doing in each step. The readme files for each of the samples gives hints on what to update and where but the level of information varies between readme files so if you don't look at all them, it is unlikely you will have any direction on what to do.

Impressions of Lab 5

Again this lab was fairly straight-forward. We set up a Python Tensorflow environment and ran another Hello-World program. It was unclear to me exactly what this Hello World program was supposed to be demonstrating. Apparently Tensorflow allows for the development of machine learning applications. I would have liked a more thorough explanation of what machine learning concepts were being demonstrated in this hello_world application. And again, like with the other labs, I would have liked to see how it could be extended.

Overall Impressions

Overall, while there were bugs and some problems, the labs themselves ran much more smoothly than we expected them to. For what these labs do, which is to guide the student into setting up the very basics of the RVfpgaSoC system, they are well documented, and provide helpful guidance where needed. Many of the bugs have already been anticipated and there are clear troubleshooting messages where appropriate. The block diagram PDFs for lab 1 were very helpful, and a good example of a way to generate useful block diagram connections for students in the future.

However, while it is a good set of instructions to get the system up and running, we would consider this whole set of 5 labs, not as labs, but as simply one large "Installation Guide". They walk the student through how to install each of these features, but no further information is given on how to extend, customize, or build upon these tools.

Heavy guidance would need to be given to students to take them to the next stage after this "installation". This might be the development of a larger example project like the one given to us in the

Getting Started guide for the Microblaze projects this term. It would certainly involve the front-loading of how to customize the hardware and software through coding demonstrations in class.

Final Recommendations

As the RVfpgaSoC system currently stands, we don't believe it is yet ready to replace the Xilinx/Microblaze/Vitis regime currently in place for 544. At least not without a lot of extra work by either the RVfpga folks, or by the instructors of ECE 544 or both. The system has potential, but there are some key improvements that need to be made both to the RVfpgaSoC curriculum and to the ECE 544 course structure:

1. Get the Lab 1 connections working with the connection automator OR provide the students with the already connected block diagram OR get rid of it entirely and just use the FuseSoC builder.
2. All labs need to include ways to extend the "installation" steps they provide, and provide more examples of how to do so for the standalone, RTOS, and Tensorflow applications.
    a. Run through more examples in each lab.
    b. How does one customize the hardware?
    c. How does one customize the software?
    d. How does one apply the RTOS constructs to create new applications?
    e. How does one apply TensorFlow to fully use its capabilities?
3. Review the debugging tools like GTKWave/VerilatorSIM in more detail and demonstrate how it can be used to debug designs.

The above items and questions need to be addressed either in the labs or during instruction for this system to have much use in meeting the goals and expectations for its use in ECE 544. Otherwise we believe students will be frustrated and unable to move forward with developing the projects and deliverables expected of this course.

Also we'd recommend to students that if doing this, they should definitely be running this in an Ubuntu environment. Labs 4 and 5 cannot be run in anything but an Ubuntu environment at present anyway. If they need to create a VM as both Brett and Chuck have done, we recommend giving it at least 80GB of disk space, and 8GB of memory.

Stretch Goals

Our stretch goals were to connect and demonstrate the functionality of a PMOD on the RVFPGA environment. As well as to implement Roy's original test code in the RVFPGA environment with the encoder and the OLED display. For implementing the PMOD, there was very little instruction or guidance on how to edit the given hardware. It became apparent that altering the hardware was far more complex and required more guidance than was available. Even though the PMOD did not get implemented, it did make it very obvious there was a large gap of education that needs to be filled after one goes through the RVfpga labs and before customizations can be made. Hopefully this gap can be filled for future students.

Since we weren’t able to customize the hardware, the stretch goal for software development was updated to create a more sophisticated zephyr program that expanded on the blinky sample code. This also ran into issues with a lack of guidance. Several of the sample programs required editing device tree files. There was some documentation but since the editing varies depending on what board you were using, it was vague. The readme file for each sample varied in the amount of information provided as well. It would have been very helpful to have a next step section in lab 4 that explained the device tree files and how to edit them.

Responsibilities

Both Chuck and Brett followed the installation and labs independently, providing feedback and recommendations. As next steps after the labs, Chuck worked on adding a PMOD to the hardware and Brett worked on running a more complex zephyr software program.

Conclusion/Summary

Overall we enjoyed working through the RVfpga-SoC curriculum. It has a lot of potential and we are looking forward to seeing the next release. The curriculum provided a good start to building a RISC-V SoCs. Most of our feedback centered on expanding what is currently provided to allow students to customize and move to the next stage.

Our main conclusion in recommending it for future ECE 544 classes is that a choice needs to be made between using Vivado/IP Integrator or using FuseSoC/Rvfpga. One has the benefit of being used in the industry today and the other introduces students to systems that are up and coming in the near future.

Appendix: Detailed Lab Instruction Document Feedback

Overall:

- For all the instruction guides the system of numbering steps was inconsistent. It was difficult to refer to specific steps. Some steps were letters and others numbers while some had no label for sub steps and others did.

Installation Guide:

- Overall
    - It would be helpful to have more explanations for all the commands that are being executed. There are the general headlines like “Install Zephyr SDK” but if you don’t have experience using linux, it isn’t clear why you need to run commands like chmod.
- Step 5. 1. (pg 10)
    - It is confusing to have one command be on separate bullets. It was not clear that sudo apt install is needed for several of the instructions
- Step 5. 2. (pg 10)
    - It was not obvious the commands needed to be run together. Some of the bullets needed to be part of the same command and some didn’t.

Lab 1:

- Step 4 (starting page 26)
    - The process of connecting all the modules by hand is very tedious. A lot of time is wasted that the student could be spending elsewhere. I don’t know if it is possible to set up the IP so that the connection wizard works well with it. But it might be something worth looking into.
- Step 6. 3. (pg 42)
    - “Now add the following four paths”. Only three paths are listed
    - LabProjects should be changed to LabResources
    - There is a syntax error in pic_map_auto.h
- Page 45
    - Vivado was run in a virtual machine and crashed at this point with very little to go off of in error messages. It was resolved by adding more memory to the virtual machine.

Lab 2:

- Step 4. 10 (pg 10)
  - The filter field at the bottom can be helpful for finding the signals quickly
- Step 4. 11 (pg 12)
  - It is unclear if and when you are supposed to be executing the program
- Step 4. 12 (pg 12)
  - It would be helpful to provide information on how to zoom into 10,100ns
- Step 5. g. (pg 14)
  - The file path is incorrect. LabProjects should be changed to LabResources
- Step 5. 5. (pg 17)
  - The instruction is a store word (sw) instead of sb.
- End of page 17
  - Looks like the rest of the file got cut off. It stops mid sentence "In the next lab, we will introduce"

Lab 3:

- Step 5. 1. (pg 7)
  - There is no SweRVolf directory already built. It is unclear if that was missing or the user needs to create it themselves.
- Step 7 (pg 12)
  - The command vivado swervolf.0.7.3 xpr should be vivado swervolf_0.7.3.xpr
- Step 8. 7 (pg 14)
  - LabProjects should be changed to LabResources
- Step 10. Hardware. 3
  - If experiencing issues later if could be possible the export of the SWERVOLF_ROOT variable didn't take

Lab 4:

- Step 5. 1. (page 5)
  - Suggestion to add "Can do 'printenv VARIABLENAME' to see if these were successful
- Step 6. 5
  - Ran into many issues at this point.
  - Received error "ERROR: 'swervolf' or any of its dependencies requires 'swervolf' but this core was not found"
  - Also "ERROR: conflicting requirements: Requirements: 'swervolf'==0.7.3-0' <- 'fusesoc_utis_generators>0-0'+swervolf-0.7.3-0 was ignored because it depends on missing packages"

    - This issue was resolved by rerunning lab 3 but we aren’t sure why it fixed the issue
    - The App.hex folder directory was too long. It was getting cut off and resulting in an error. It had be cut and pasted into an earlier directory and point the command to that for it to work
- Step 8. 3 (pg 15)
    - There are two step 3s
- Step 8 at the end (pg 16)
    - How do you exit the program? Are there more commands we can run at this stage?
- Step 10. 4. (pg 18)
    - Can also do ‘code main.c’ and edit in VSCode
- Step 10. 13 (pg 24)
    - Don’t we want to run our new app? Get rid of /basic/blinky/ and replace with “my_first_app”
- Step 10. 14 (pg 25)
    - Have to use ‘sudo putty’ or the connection will fail

Lab 5:

- Step 6. 4. (pg 9)
    - Got error “No Module named ‘fusesoc’”. Had to leave virtualenv by closing terminal