S.P.U.D.
Single-core Processing Unit and Display

Initial Divide and Conquer Document

EEL 4914 - Senior Design I

Group C

University of Central Florida

Department of Electrical and Computer Engineering

Authors:

Valentina Terry	Computer Engineer
Mohamed Moussa	Computer Engineer
Gabriel Saborido	Computer Engineer
Frank Laterza	Computer Engineer

Reviewers:

Dr. Sazadur Rahman

Dr. Suboh Suboh

Dr. Saleem Sahawneh

Advisors:

Dr. Mike Borowczak

Chapter 1: Project Description

1.1 Motivation and Background

Our motivation is simple yet bold: to create a CPU that runs RISC-V instructions and to become the best engineers we can possibly be! As a team of four passionate computer engineering students, we see our senior design project as a culmination of everything we have learned over the past four years. Given our team consists of only computer engineers, we feel empowered to take a dive deep into a project that is distinctly rooted in the principles of our field. We want to push the boundaries of what it means to be computer engineers by building something ambitious, innovative, and undeniably "us." That vision takes the form of designing a CPU from the ground up and utilizing open source technology with RISC-V to power our designed logic on an FPGA. We also want our project to have external functionality by connecting it to an LED matrix display allowing us to use our RISC-V software to display images, data, and run simple games like the game of life, pong, and snake. Also our project will connect with a PCB to help demonstrate the inner workings of the CPU.

We are motivated by the opportunity to integrate hardware and software in a way that challenges us to apply the skills we've developed through our coursework and personal projects. For example, in Digital Systems, we dove into Verilog and programming FPGAs to calculate k maps, truth tables, and digital logic. In Computer Organization, we learned about machine code, RISC-like assembly language, instruction types, and CPU schematics. In Computer Architecture, we went over different types of architectures, branch predictions, and cache. From the fundamentals of digital logic design to the intricacies of computer architecture, this project offers us a chance to synthesize everything we've learned while taking on real-world challenges. It’s a bold, hands-on endeavor that highlights our passion for computer engineering: innovation, problem-solving, and the drive to create something extraordinary.

What excites us most is the potential to grow as engineers and push beyond our comfort zones. The basic goal of our project is to design SystemVerilog to run on an FPGA and design testbenches to confirm functionality. However, we wanted to find an interesting way to visualize our work. Using the RISC-V GCC compiler to generate machine code for our CPU, we plan to pair our logic with an interactive display to extend its functionality. If time allows, we would love to explore possible architectural improvements—such as security enhancements or out-of-order execution—and perhaps even running DOOM. These challenges fuel our motivation to build a CPU that is not only functional but also reflects our ambition to learn and innovate.

Ultimately, this project is a tribute to our journey as computer engineers. It’s a chance to prove to ourselves and others that we are ready to take on complex, meaningful challenges. We aren’t just designing a CPU; we are crafting an experience that

combines our technical skills, creativity, and teamwork into a project that's as exciting as it is impactful. For us, this is more than a senior design project—it's a rite of passage and the first step toward becoming the engineers we've always aspired to be.

1.2 Review of Prior Work

Our team is composed of four inspired computer engineering students, each bringing unique skills and experiences that directly support the goals of this project. Together, we aim to leverage our diverse backgrounds to create a CPU that runs RISC-V instructions and powers an interactive screen interface.

Frank brings expertise in low level software development and PCB design. They have hands-on, industry experience with real time systems software development and multithreading design. Completing many projects like self balancing robots, embedded door locks, and flight avionics software he knows the ins and outs of computer functionality and implementation. Their ability to integrate low level software and solve tough problems will ensure thorough high level system logic and smooth integration of RISC-V implementation.

Valentina has extensive experience in ASIC design and verification. Having internship, research, and course experience, she has designed and verified certain designs using UVM and direct testbenches in SystemVerilog. Their familiarity with the language and their ability to debug hardware designs make them an invaluable asset for implementing and testing the FPGA-based CPU.

Gabriel specializes in PCB design, embedded systems, and has a strong foundation in analyzing processor functionality. Their prior work involves designing a custom macropad using Arduino. This demonstrates their ability to create hardware and implement software to complete specific tasks. These skills will be critical for ensuring our CPU supports the RISC-V instruction set via the GNU toolchain.

Mohamed has a strong background in circuit design, having successfully designed and simulated multiple circuits relating to a multitude of projects. PCB, electronic design, digital design, and verification will play a key role in developing the PCB and interfacing it with the FPGA and game display with his varied skill set.

As a team, we are motivated not only by our individual areas of expertise but also by our shared determination to design a cohesive, innovative project. By combining our knowledge of digital logic, computer architecture, embedded systems programming, and hardware design, we are confident in our ability to tackle the challenges of this ambitious project and create a meaningful contribution to the field of computer engineering.

Regarding similar projects, this project was inspired by various creative interpretations of the CPU, including those featured in popular video games that utilize built-in logic gates (Appendix A). We realized that with a sufficient supply of logic units, any digital logic circuit can be replicated. Building on these video game implementations, we also explored relevant literature on the process of designing such systems, which will serve as references as we incorporate our own unique twist (Appendix A).

1.3 Overview of Project Goals

Our basic goals are to get an FPGA to run digital logic to design our own CPU to run program instructions. Including verifying our design with direct testbenches (if time allows, we would ideally like to use formal verification methodologies, or UVM-based testbenches instead). While this sounds simple, this could potentially take up the bulk of our project, as it requires us to use all our knowledge from past courses involving digital systems and computer architecture. To extend the functionality of our project we also plan to add an LED matrix panel designed for FGPAs.

Once the CPU architecture is tested and functional we move onto our advanced goals. To take our project to the next level our goal is to start writing software programs for the CPU to crunch on. These programs will start small, but for demonstration day we'd like to make impressive demonstrations to put our work to the test! Since we'll be adding a screen to our project, we can utilize that to show code executions and display visuals.

At this point, the main goals for our project are complete. Moving forward we will reach the stretch goals to improve upon our machine by implementing new effectiveness. We plan to make improvements to the CPU architecture features that would benefit our project such as the out-of-order execution or the security. In terms of CPU power, we'd like to run DOOM and display it on the LED matrix with minimal issue.

Table 1: Project Goals

Goal Type	Description
Basic	Design and run simulations of CPU logic in modular architecture. Test through direct testbenches. Create example programs to confirm logic. Get the FPGA to run our CPU with precoded instructions. Design PCB to interface with FPGA for an LED matrix.
Advanced	Utilize the RISCV-gcc compiler to create machine code to run on CPU, use UVM-based testbench to verify designs
Stretch	Make improvements to the RISC-V architecture. Improve CPU capabilities and implement DOOM onto the LED matrix display.

1.4 Overview of Objectives

For this project we have a set of objectives relating to multiple levels of design for the CPU. To break up the tough challenge we will delegate logical blocks into separate projects completed by members allowing us to work in parallel. The start of our design will be leveraged based on existing FPGA design. This will give us guidance to move forward with our design. We will focus on functionality and loading software then approach our own custom design as needed. After designing the CPU we will carefully calculate the inputs and outputs of each module so we can easily work on each section separately. This will also make running test cases on our design much easier. Our plan is to work from the high level design and adhere to the **RV32I** instruction types to work with simple lines of machine code. Understanding the architecture at a high level and making a good plan will increase workflow and focus our team.

We will start our journey with simulation of the digital logic to give us better insights into the behavior of our CPU and turn over rapid SystemVerilog code. After successfully confirming the logic and implementing it into the FPGA, we will move onto our advanced goal that will really make this project our own. Utilizing the RISC-V GNU toolchain, we can bring the project to the next level by compiling C code into machine code to run on the CPU.

We also plan to make our computer interactive to show our programs functionality. Creating a custom PCB for logic LEDs and User inputs and adding a 32x32 LED matrix connected to the FPGAs IO will allow us to extend the functionality of our CPU. We plan to map the physical bits in our 32 x 32 bit register architecture to the LED grid. Allowing us to physically see what's inside the CPU as well as any general purpose debugging interface. As a bonus this display can serve as a simple screen to run visual demonstrations to run complex programs or demos.

1.5 Project Features and Functionalities

Once our CPU has been completed, this opens lots of functionality and possibilities. A completed CPU can be fed instruction to run any program we desire. Developing in assembly, we can load data, store data, load rediters, use the ALU on registers and immediate values, branch on condition, jump, halt, and more. But using assembly is slow to develop with. To make development more rapid we can use a program to generate assembly/instructions for us. This is exactly what the compiler will do for us. Utilizing the RISC-V GNU tool chain we can use C code to further abstract resulting in accelerated development. This will take our CPUs functionality to the next level.

Crunching numbers is fun, but not very exciting. To extend the functionality of our CPU we are adding a 32x32 LED matrix screen to display anything we can imagine (at a very low resolution). This will also make developing more visually appealing and help to demonstrate the power of our project. We plan on running simple programs, simulations, and even games! Some examples include A perlin noise generator, a simulation of the game of life, Conway's game of life, and we can even use the buttons on the controller to play games.

1.6 Requirement Specifications

Table 2: Specifications

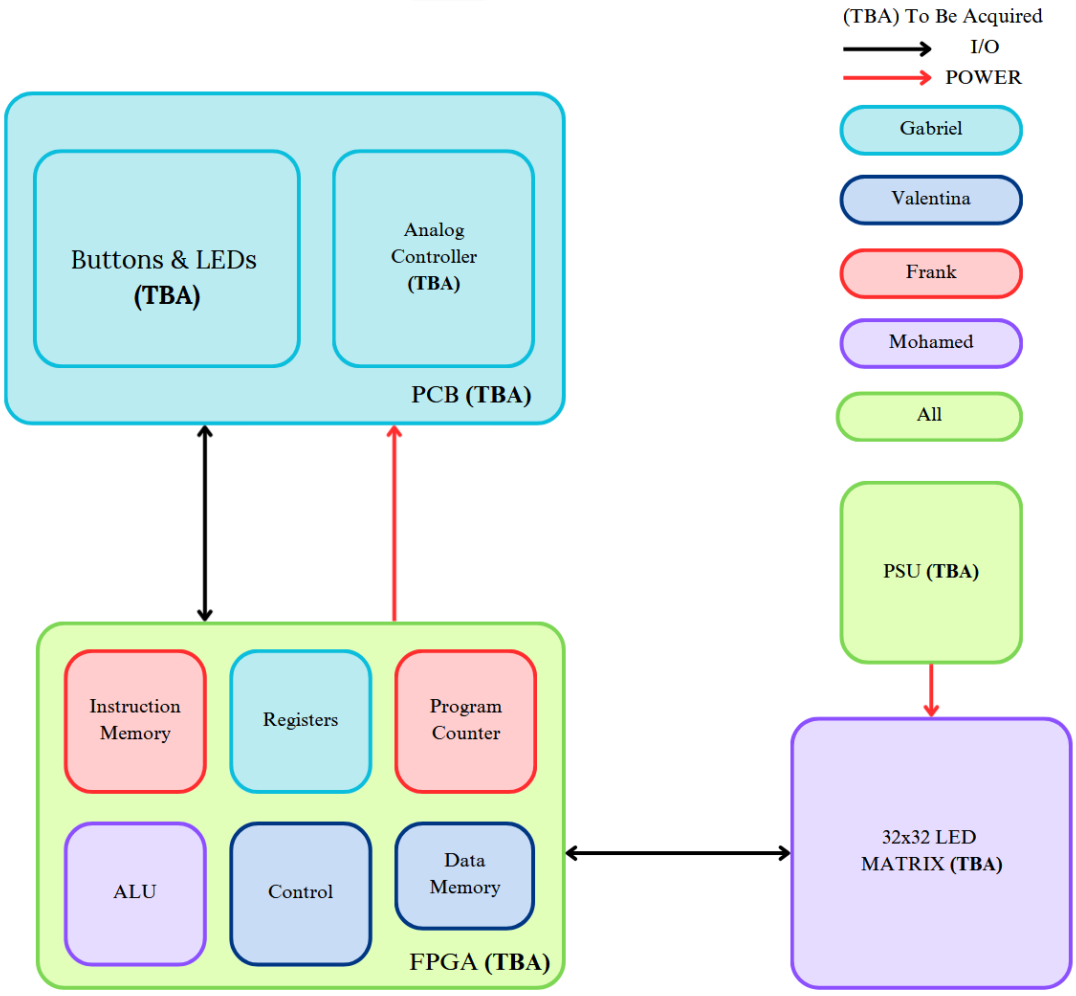
Specifications	
Performance	The CPU is expected to run at a frequency of 50 MHz and be able to drive a 30 Hz LED matrix. The processor will run with 10 MIPS.
Functionality	The system is able to execute multiple RISC-V instructions including instructions relating to arithmetic operations, jump operations, and memory operations. The system is also capable of executing simple programs including simple implementations of video games and basic algorithms.

Energy	The FPGA board draws 36 watts of power from a 12V, 3A supply. The LED matrix draws 20 watts from a 5V, 4A supply. The other LEDs would take around 0.8W. Total power draw = 57W.
Environmental	The project has a modest power draw and no other external fuel consumption.
Health and Safety	The project poses no health and safety dangers.
Maintainability	The project is open to further modifications and upgrades as long as said upgrades are not beyond the allotted logic units.
Manufacturability	The project requires around 300-500, making it somewhat expensive to mass produce, however, if mass produced with custom boards/led matrices it could be within a reasonable price range.
Operational	Ideally, the project is running in a secure, dry, room temperature environment. Adjustments can be made to prove it against more adversary environments but said protections are currently not implemented.
Reliability	The functional coverage for this design will be 90% or higher.
Usability	The external peripherals and some of the calculations can be easily accessible by end users. Implementing additional programs might be difficult for the end user and might only be reserved to our team.

Chapter 2: Project Visualization

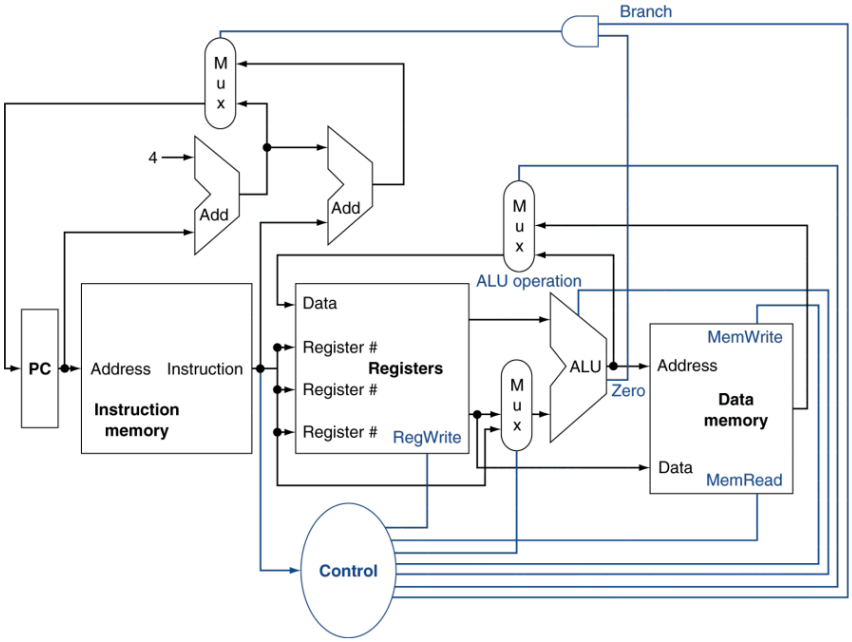
2.1 Hardware Block Diagram

Diagram 2.1.1: Hardware Overview



This diagram shows the physical hardware layout of our project, including color coded delegation.

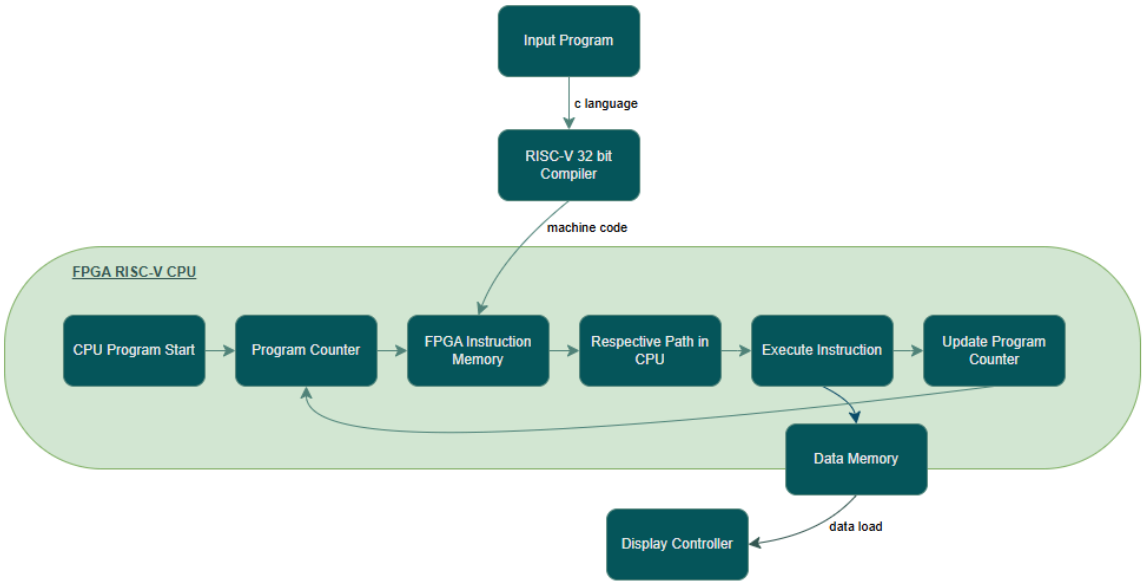
Diagram 2.1.2: CPU Architecture within FPGA



The High Level architecture of our CPU

2.2 Software Flowchart

Diagram 2.2.1: Software Flowchart



Note that our flowchart does not have different colored blocks. We purposefully did this because our project is right on the border of hardware *and* software. Thus, all of the tasks delegated in the hardware block diagram are also the software tasks (we will be writing the SystemVerilog ourselves). The software flowchart, therefore, represents the

path of the program which then breaks down into the path of each instruction. Starting with the input program, which is compiled through the RISC-V toolchain and converted into machine code, each instruction gets loaded into the instruction memory of the CPU. Once the CPU program starts, the program counter (PC) begins fetching instructions from memory. The CPU then decodes and executes the instruction, updating the PC as necessary. Additionally, the execution stage may involve interacting with data memory on the FPGA, which then sends data to the display controller. This final stage controls the LED matrix for visual feedback, completing the cycle of input, processing, and output. Finally, regarding testing/verification, Valentina and Frank will be responsible for writing the testbenches for each module and then for the CPU as a whole.

Table 2.2.1: RISC-V Instruction Set

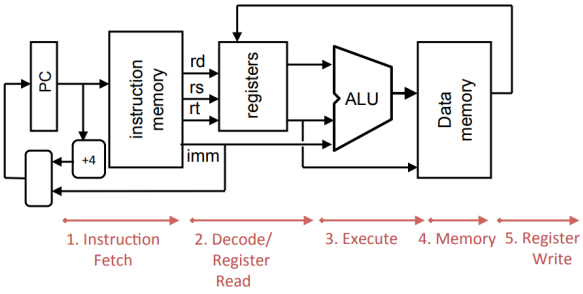
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Instruction Set

imm[31:12]					rd	0110111	LUI
imm[31:12]					rd	0010111	AUIPC
imm[20|10:1|11|19:12]					rd	1101111	JAL
imm[11:0]			rs1	000	rd	1100111	JALR
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]		rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]		rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]		rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]		rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]		rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]			rs1	000	rd	0000011	LB
imm[11:0]			rs1	001	rd	0000011	LH
imm[11:0]			rs1	010	rd	0000011	LW
imm[11:0]			rs1	100	rd	0000011	LBU
imm[11:0]			rs1	101	rd	0000011	LHU
imm[11:5]		rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]		rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]			rs1	000	rd	0010011	ADDI
imm[11:0]			rs1	010	rd	0010011	SLTI
imm[11:0]			rs1	011	rd	0010011	SLTIU
imm[11:0]			rs1	100	rd	0010011	XORI
imm[11:0]			rs1	110	rd	0010011	ORI
imm[11:0]			rs1	111	rd	0010011	ANDI
0000000		shamt	rs1	001	rd	0010011	SLLI
0000000		shamt	rs1	101	rd	0010011	SRLI
0100000		shamt	rs1	101	rd	0010011	SRAI
0000000		rs2	rs1	000	rd	0110011	ADD
0100000		rs2	rs1	000	rd	0110011	SUB
0000000		rs2	rs1	001	rd	0110011	SLL
0000000		rs2	rs1	010	rd	0110011	SLT
0000000		rs2	rs1	011	rd	0110011	SLTU
0000000		rs2	rs1	100	rd	0110011	XOR
0000000		rs2	rs1	101	rd	0110011	SRL
0100000		rs2	rs1	101	rd	0110011	SRA
0000000		rs2	rs1	110	rd	0110011	OR
0000000		rs2	rs1	111	rd	0110011	AND
fm	pred	succ	rs1	000	rd	0001111	FENCE
1000	0011	0011	00000	000	00000	0001111	FENCE.TSO
0000	0001	0000	00000	000	00000	0001111	PAUSE
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK

Above is the **RV32I** 32 bit RISC-V instruction set that our CPU will support.

Table 2.2.2: RISC-V Instruction Cycle

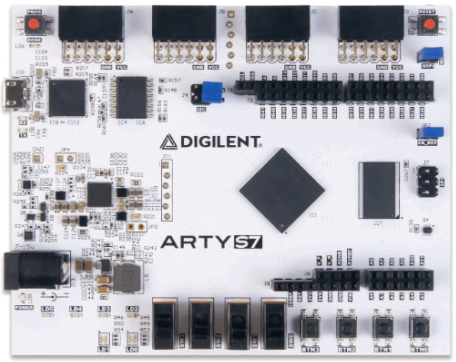


This is the 5 cycle process that will run each instruction.

2.3 Prototype Blueprint

Our prototype will include the following items

Image 2.3.1: Arty S7-50: Spartan-7 FPGA



This is our FPGA of choice. A Xilinx development board with plenty of logic cells and IO. As well as analog read pins.

Image 2.3.3: 32x32 PCB Silkscreen Layout

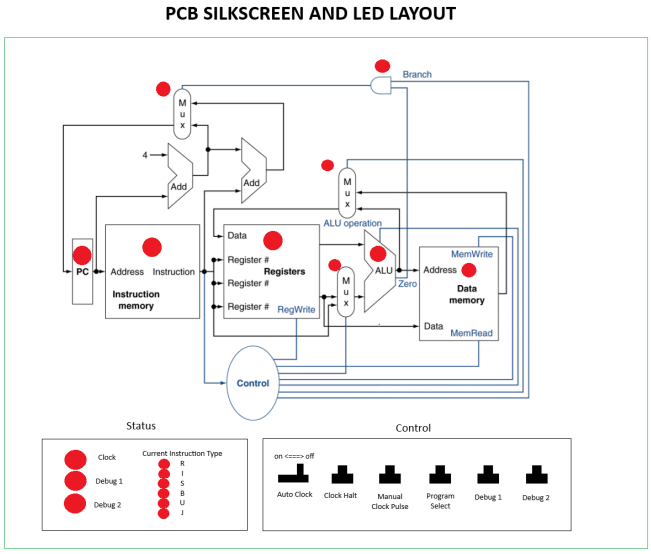
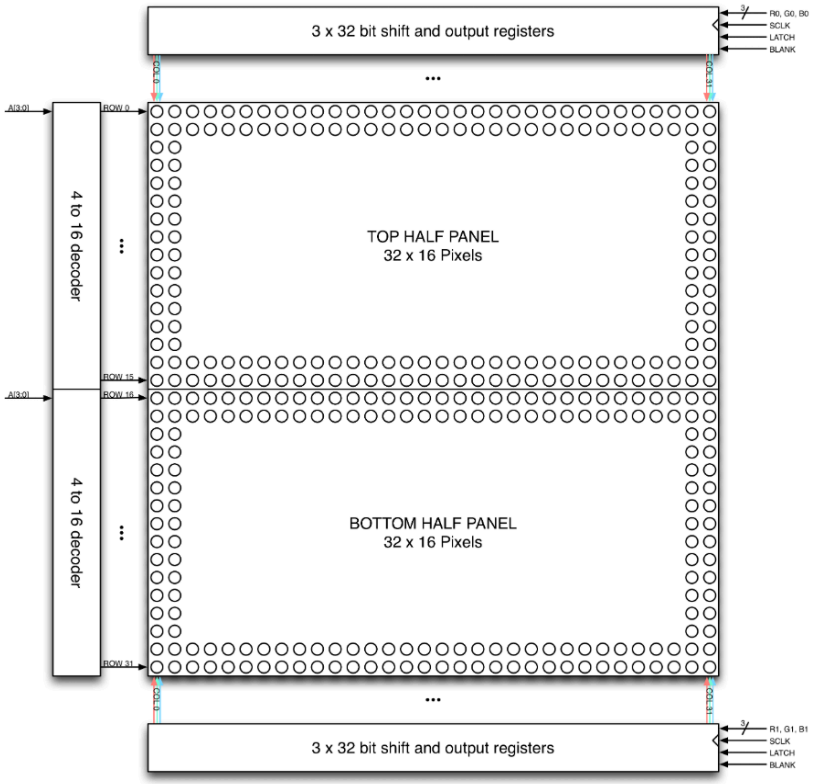
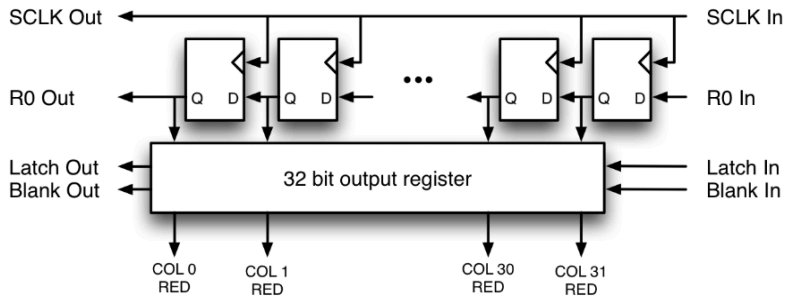


Image 2.3.3: 32x32 LED Matrix



Alternatively, and <u>probably a better idea</u>, is to buy an existing display. This one from adafruit is similar but with an extra 3x shift register and latch technique. This is perfect for an FPGA!

Image 2.3.4: RGB shift buffers



Here are the shift registers used to slide RGB data down the buffers.

Chapter 3: Budget and Financing

Table 3: Budget - First Draft

Component	Quantity	Total	Details
Arty S7-50: Spartan-7 FPGA	1	$199.00	Programmed CPU
Printed Circuit Board	1	TBD	Integrates interface and CPU
32x32 LED Display	1	$37.90	Interface
Controller/Buttons/ LEDS	MISC	TBD	Simple Momentary Push Buttons and through hole LEDs
Total:			$500 max

Chapter 4: Initial Project Milestones

Table 4: Initial Project Milestones

Senior Design I		
Week	**Description**	**Due Date**
1-2	Brainstorm project ideas	-
3	Research and D&C Document	01/17/25
4	Finish D&C Document	01/24/25
5	Meet with SD Coordinators to Discuss Project	01/27/25
6	Begin 60-page Draft	03/24/25
7	Study Possible Architecture and Design Patterns	-
8-10	Brush Up With SystemVerilog and Start Simulations	-
11	Finish 60-page Draft	03/24/25

12-13	Midterm Report Feedback Meeting and Work on Final Report	-
14	Order FPGA and LED screens	-
15	Continue working on Final Report	04/22/25
16	Submit 120-page Final Report	04/22/25

Senior Design II		
Week	**Description**	**Due Date**
1	Scheme and plan work breakdown	08/22/25
2	Implement testing software with SystemVerilog	08/29/25
3	Start Design of Display Controller	09/05/25
4-7	Design Verilog For ALU, Registers, Data Mem, and Instruction Mem. In parallel.	09/12/25
8	Test design with Verilog and Integration Testing Software With New Modules.	10/10/25
9	Finish Controller Logic	10/17/25
10	Test CPU on machine code	10/24/25
11	Start Making Instructions with RISC-V GCC Toolchain	10/31/25
12	Start making simple programs and confirm Instructions run correctly after the compiler makes machine code.	11/07/25
13	Make CPU machine code loader for instruction memory	11/14/25
14	Make Efficiencies On CPU	11/21/25
15	Design Programs for Demo Day	11/28/25
16	Complete last TODOs and Finish Strong!	12/05/25

Appendices

Appendix A – References

XSoC Series. (n.d.). The XSOC Series. [Online resource]. Retrieved January 23, 2025, from http://www.fpgacpu.org/papers/xsoc-series

YouTube. (n.d.). RISC-V playlist [Video playlist]. YouTube. Retrieved January 23, 2025, from https://youtube.com/playlist?list=PL5LiOvrbVo8nPTtdXAdSmDWzu85zzdgRT&feature=shared

PassLab. (n.d.). Lecture 08: RISC-V Single-Cycle Implementation. [Online lecture slides]. Retrieved January 23, 2025, from https://passlab.github.io/CSE564/notes/lecture08_RISCV_Impl.pdf

KianRiscV. (n.d.). KianRiscV: RISC-V Linux SoC. [Source code repository]. Retrieved January 23, 2025, from https://github.com/splinedrive/kianRiscV

InkboxSoftware. (n.d.). excelCPU: 16-Bit CPU for Excel, and related files. [Source code repository]. Retrieved January 23, 2025, from https://github.com/InkboxSoftware/excelCPU

Levy, B. (n.d.). learn-fpga: Learning FPGA, yosys, nextpnr, and RISC-V. [Source code repository]. Retrieved January 23, 2025, from https://github.com/BrunoLevy/learn-fpga

Appendix B – Documentation

Five-EmbedDev. (n.d.). RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA (Priv-v1.12). Retrieved January 23, 2025, from https://five-embeddev.com/riscv-user-isa-manual/Priv-v1.12/instr-table.html

Adafruit. (n.d.). 32x32 RGB LED Matrix Panel – 4mm Pitch. [Product documentation]. Retrieved January 23, 2025, from https://www.adafruit.com/product/607

RISC-V Collaboration. (n.d.). RISC-V GNU Toolchain. [Source code repository]. Retrieved January 23, 2025, from https://github.com/riscv-collab/riscv-gnu-toolchain

Appendix C – copyright permission

Everything is Open Source!