S.P.U.D.

Single-Core Processor Unit and Display

Meet The Team

Mohamed Moussa
Computer Engineering
Hardware Design

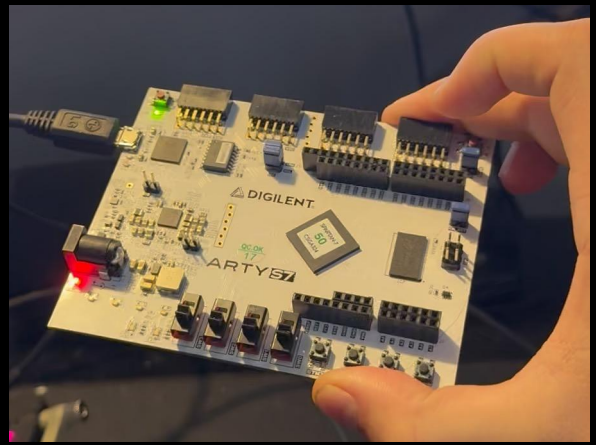
Frank Laterza
Computer Engineering
Project Lead

Gabriel Saborido
Computer Engineering
Electrical Design

Valentina Terry
Computer Engineering
SoC RTL Design

Our Motivation

- Take advantage of the fact that our team is **100% CpE**
- Understand the fundamentals of computer architecture, register transfer level (RTL) design, and programming languages
- Build a project that challenges us in a new and creative space
- Develop specific skills that prepare us for our desired careers and share those skills with future generations
- Build a learning ‘playground’ for aspiring computer engineers
- Earn the title of Computer Engineer and have fun!

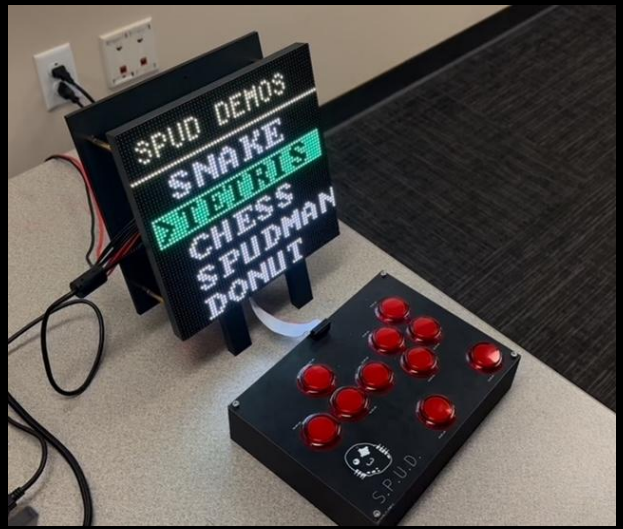


RISC-V®

What Is Project S.P.U.D.?

An educational CPU system that transforms C programs into real-time visual output, allowing users to observe how software instructions directly translate into hardware behavior.

- Modern processors are fully optimized and difficult to inspect
- SPUD proposes a custom, modifiable, demonstrational, and accessible CPU for educational and entertainment purposes
  - can be customized in both the hardware level (Verilog) and software level (C programs)
- C programs can run directly on the FPGA-based CPU using a custom software library (spudkit) and SPUD toolchain
- CPU and System on Chip (SoC) are implemented in Verilog
- A custom matrix core drives the 64x64 RGB Matrix
- The arcade controller will be the main interface to control interactable demos



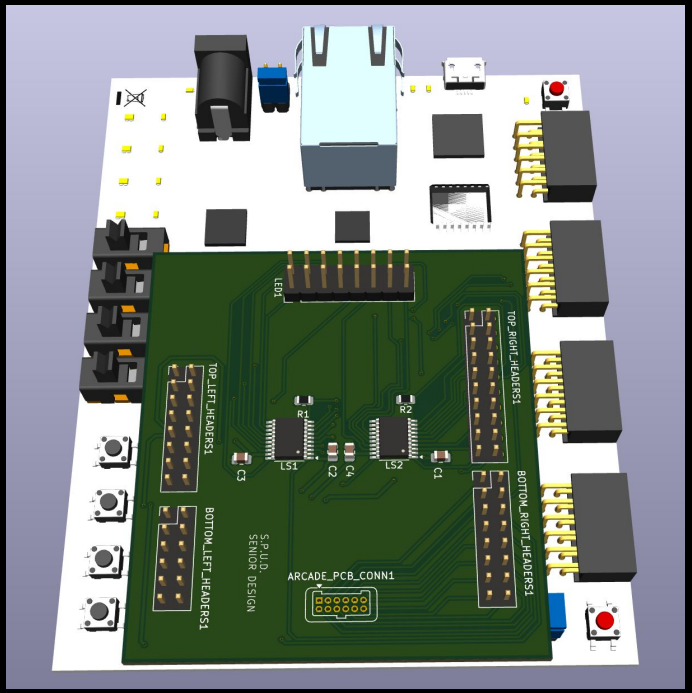
Goals and Objectives

Goals:

- Make a user-friendly, simple pipeline for C program conversion into visual output on a display
- Design our own embedded processor catered to our project

Objectives:

- Build a C library for controlling peripherals like UART, DISPLAY, GPIO, etc
- Implement custom registers in SoC allowing hardware control via instructions
- Configure RV32IM toolchain to generate instructions from C code to load into RAM
- Have each member design a unique interactive demo program!



SPUD PCB connected to the Arty FPGA development board

Engineering Specifications

Specifications	
Processor Performance	> 10 MIPS
Arcade Button Latency	< 30 ns
LED Matrix Refresh Rate	< 200 Hz
Functions	arithmetic, jump, and memory
FPGA Board Power Consumption	< 36 W (12V, 3A supply)
LED Matrix Power Consumption	< 20 W (5V, 4A supply)
Operational Conditions	Room Temperature: 68–77°F Humidity Level: 40-60%
Price Range	$300-$500

House of Quality

House of Quality - S.P.U.D.

Roof

How's	Importance	LED Matrix / Controller IO	SPUDKIT Library	High Frequncy Core	Full System Housing/ Support Brackets	Detachable Power Connectors
Ease of Use/ Modularity	5	5	9	1	1	2
Preformance	5	1	6	9	1	1
Power Consumption	4	2	5	3	1	6
Size	2	4	1	1	9	5
Display Clarity	2	9	3	1	4	1
Importance Weight		21	24	15	16	18
Relative Importance Weight (%)		22.3	25.5	15.9	17	19.1

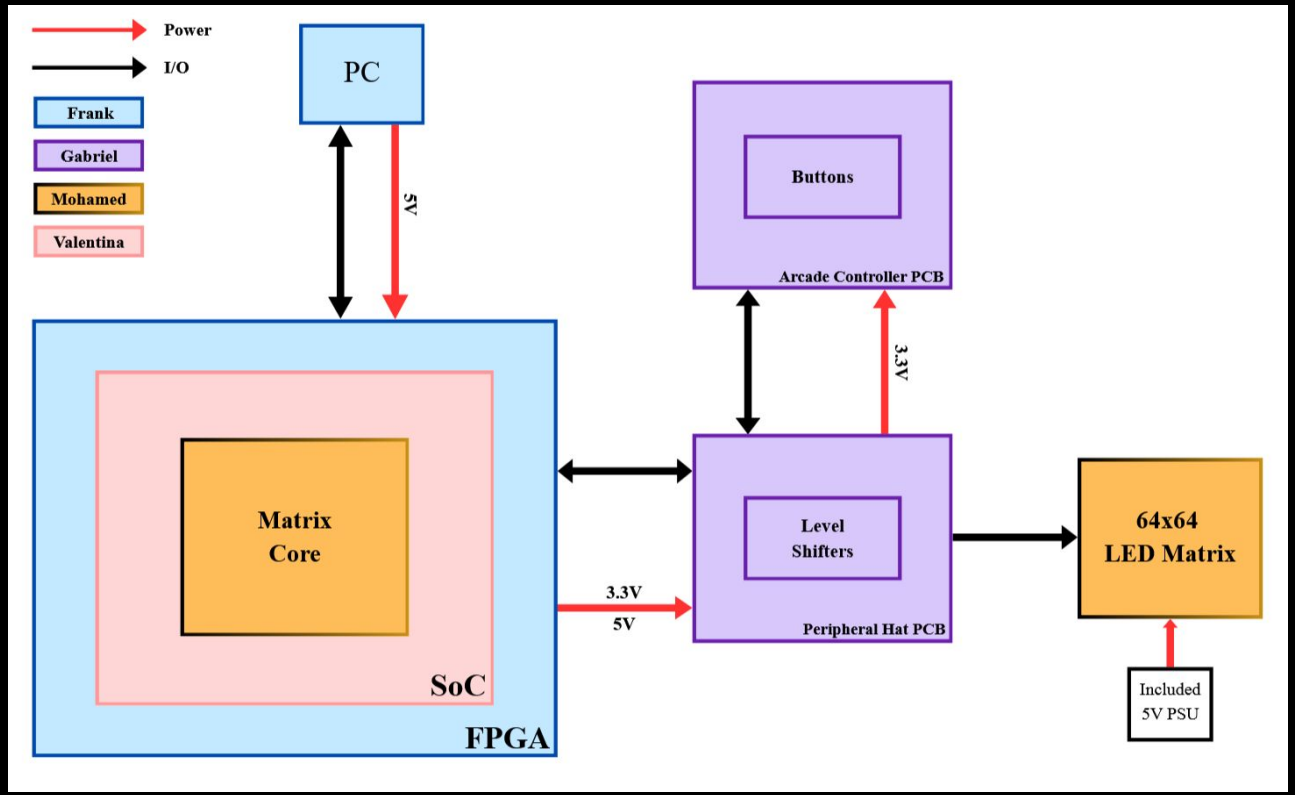
Final Product

64x64 LED Matrix

Arty S7 FPGA

Arcade Controller

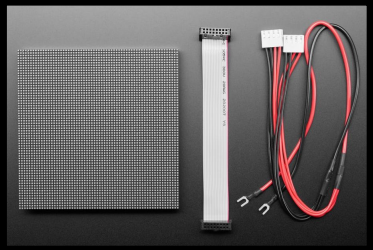
Hardware Block Diagram



Hardware Component Research

Hardware Comparisons - *System Display*

- Utilizes the powerful computing capabilities of FPGA based core.
- Brightness is necessary for demo.
- Simple implementation, does not require external libraries.
- Matches retro feel of our project.



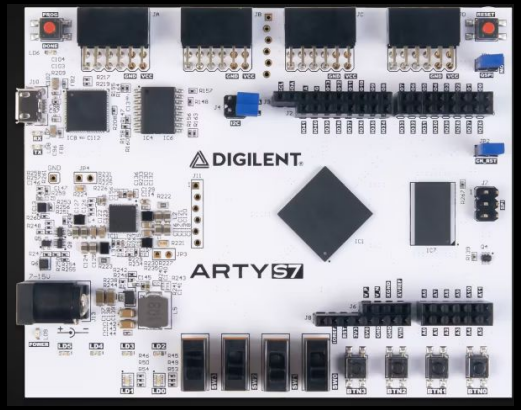
Adafruit 64x64 LED Matrix

Feature	LCD (Liquid Crystal Display)	OLED (Organic Light-Emitting Diode)	Matrix Displays
Resolution Potential	High, dependent on pixel size and TFT technology	High, with fine control over individual pixels	Limited by matrix grid design
Brightness	Moderate to high (200-400 cd/m^2 with LED backlight)	Lower peak brightness compared to LCD; true blacks enhance perceived contrast	Limited brightness (~1/10 of segmented displays)
Contrast Ratio	Moderate (limited by backlight)	Infinite (self-emissive pixels can turn off completely for true blacks)	Moderate
Viewing Angles	Narrower than OLED (varies by LCD type: TN, IPS, VA)	Wide 180 degrees	Moderate
Response Time	Slower (~6-16 ms for LCDs; faster for TN panels)	Extremely fast (<0.01 ms), ideal for motion clarity	Moderate
Power Efficiency	Requires backlight, consumes more power	Highly efficient; only active pixels consume power	Dependent on the driving method

Technology Selected - Matrix Display

Hardware Comparisons - *FPGA Selection*

- Artix-7: most powerful, expensive
- Cyclone V: uses Quartus platform, not Vivado
- Spartan-7: like Artix-7, but smaller and less expensive with sufficient logic cells and BRAM



Xilinx Arty S7-50: Spartan-7

Feature	Xilinx Arty S7-50: Spartan-7	Xilinx Arty A7-100T: Artix-7	Intel Cyclone V GX Starter Kit
Logic Cells	52,160	101,440	77,000
Flip-flops	65,200	126,800	56,832
Block RAM	2,700 Kbits	1,800 Kbits	1,532 Kbits
Clock Frequency	100 MHz	100 MHz	programmable (10 - 810 MHz)
Communication Protocols	- USB-UART - USB-JTAG - Quad-SPI Flash - Pmod ports - Arduino/chipKIT shield connector	- USB-UART - USB-JTAG - Quad-SPI Flash - Pmod ports - Arduino/chipKIT shield connector	- PCIe Gen2 - USB 2.0 - UART - I^2C - SPI - LVDS
Memory Interface Support	- 256 MB DDR3L with a 16-bit bus @325 MHz (650 MT/s) - 128 Mbits Quad-SPI Flash	- 256 MB DDR3L with a 16-bit bus @667 MHz - 128 Mbits Quad-SPI Flash	Supports DDR3, DDR2, LPDDR2
Development Tools	Vivado Design Suite, WebPACK Edition	Vivado Design Suite, WebPACK Edition	Quartus Prime, Lite Edition
Board Price	$150-$200	$299.00	$270.00
Applications	Embedded systems, education, prototyping, hobbies	High-performance systems, communications	Industrial, automotive, telecommunications

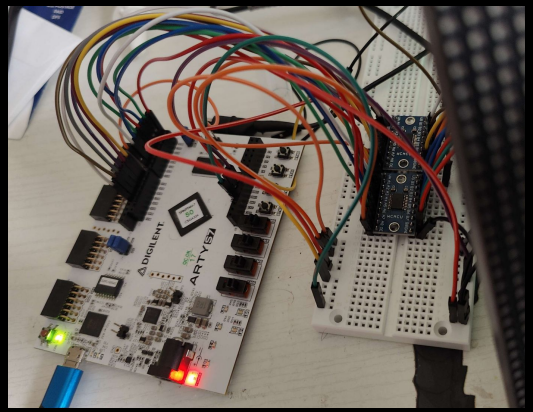
Part Selected - Arty S7-50 Spartan-7

Hardware Comparisons - *Level-Shifter*

- PCA9306 and 74LVC245: not as fast as other options
- TXB0108: fastest, but more expensive
- TXS0108E: similar speeds to the TXB0108, cheaper, and meets required voltage range

Feature	Resistor	Op Amp	Transistor
3.3V to 5V conversion	N/A Only 5V to 3.3V	Yes	Yes
Propagation delay	~50-200ns	~10-100ns	~5-50ns
Directionality	Unidirectional/ Bidirectional	Unidirectional/ Bidirectional	Unidirectional/ Bidirectional
Durability	Low	Medium	High

Technology Selected - Transistor



TXS0108E Level Shifter

Feature	TXS0108E	TXB0108	PCA9306	74LVC245
Propagation delay	1-4 ns	<1 ns	10-20 ns	6-8 ns
Directionality	Bidirectional	Bidirectional	Bidirectional	Unidirectional
Voltage Range	1.2V to 5V	1.2V to 5V	1.2V to 5V	1.65V to 5V
Channels	8	8	2	8

Part Selected - TXS0108E

Hardware Comparisons - *Flash Memory*

- Less pins required than parallel protocol
- Quad-SPI has several improvements over SPI
- No extra circuitry/connections required when using the built in flash memory
- More memory overall compared to other options

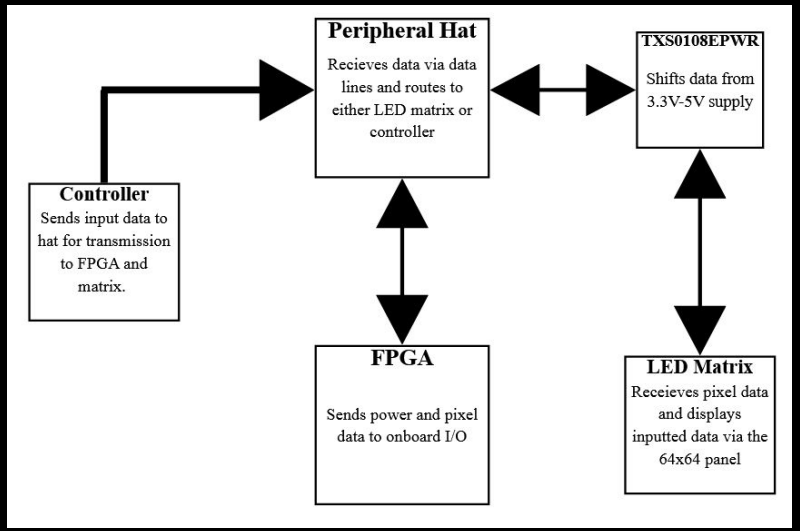
Feature	M3004316	AT25DL081	24LC16B (Arty S7)
Size	4Mb	8Mb	16 MB (128 Mbits)
Mem Technology	MRAM	FLASH	FLASH
Write/Read Protocol	Parallel	SPI	Quad-SPI
Program Technique	Direct write (no erase needed)	Page programming with sector erase	Page programming with sector erase (similar to standard SPI)
Access Speed	35ns read/write	~100MHz clock, sequential access	Up to 50 MHz data rate for Quad-SPI programming
Endurance	Virtually unlimited	100,000 cycles (typical)	Typically 100,000 cycles (similar to standard SPI)
Data Retention	Long-term persistent	>10 years (typical)	>10 years (typical)
Voltage Range	2.7V-3.6V	1.65V-3.6V	1.65V-3.6V (for S25FL127S/S25FL128S)
Temperature Range	-40°C to +105°C	-40°C to +85°C	-40°C to +85°C

Part Selected - 24LC16B

Hardware Design

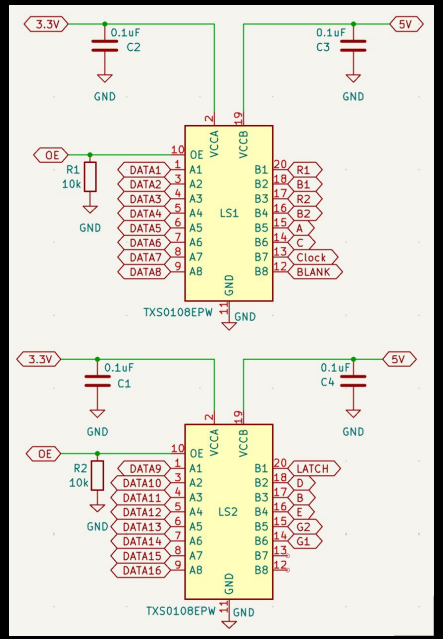
Hardware Design

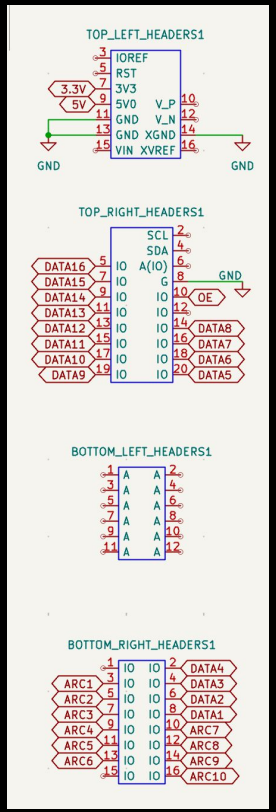
- SPUD uses a custom PCB hat that fits on top of the FPGA.
- The hat allows us to add connectors to make secure connections to the matrix and arcade controller
- Level shifter are also implemented onto the hat to allows signals be control at safe voltage levels
- Button inputs from the arcade controller PCB are sent through the GPIO pins, allowing for real-time interaction with SPUD

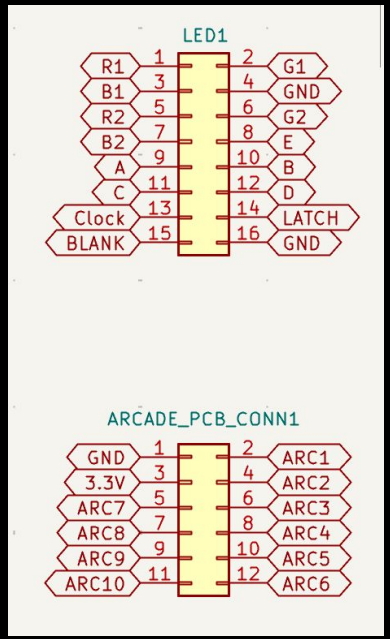


PCB Design

Peripheral Interface PCB - *Schematic*



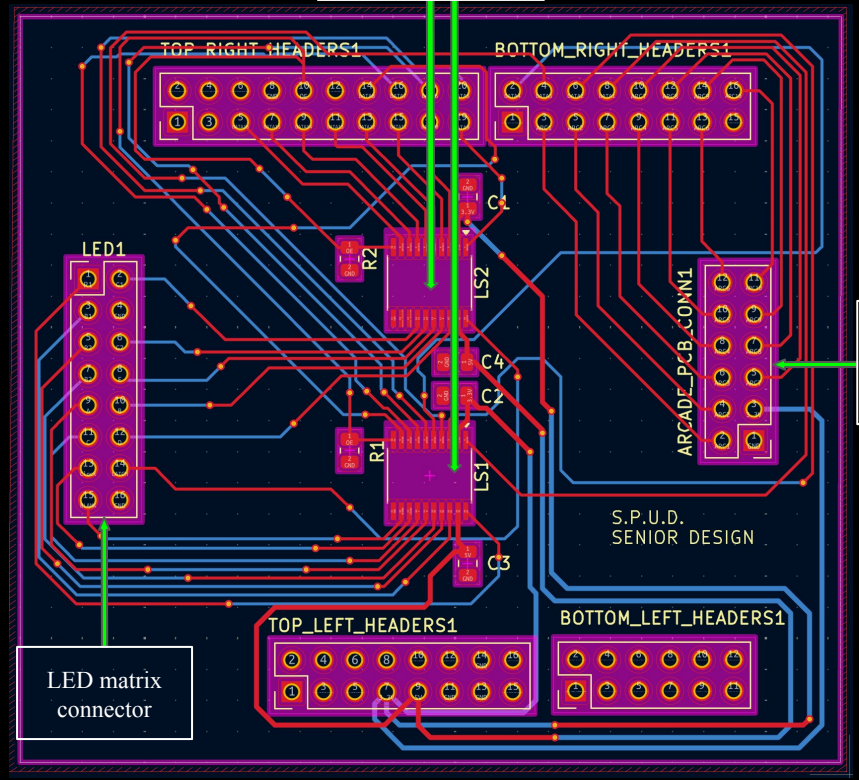


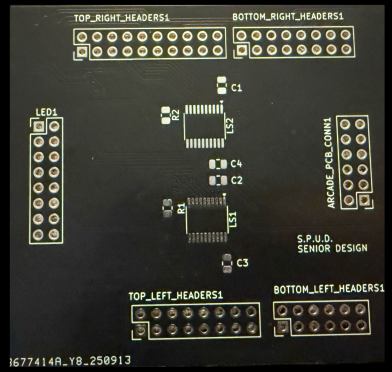


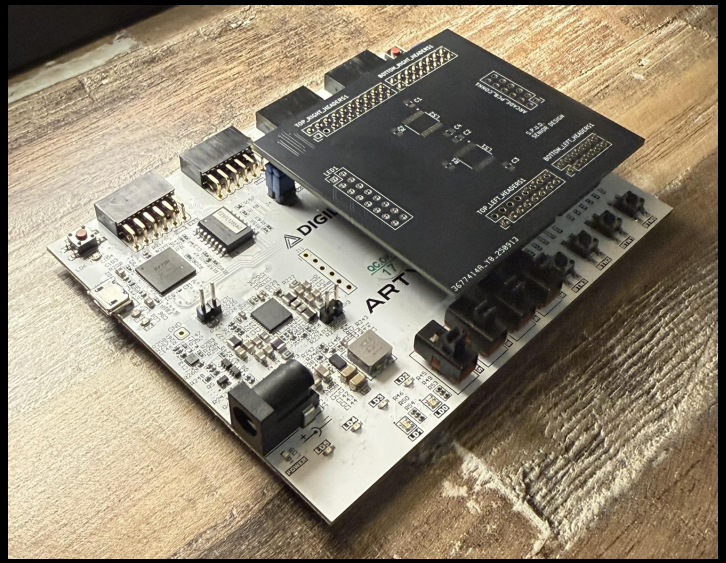
Houses FPGA, two 8-bit bidirectional level shifters, LED matrix & arcade controller connecters.

TXS0108E level shifters are for 3.3V-5V conversion for LED matrix.

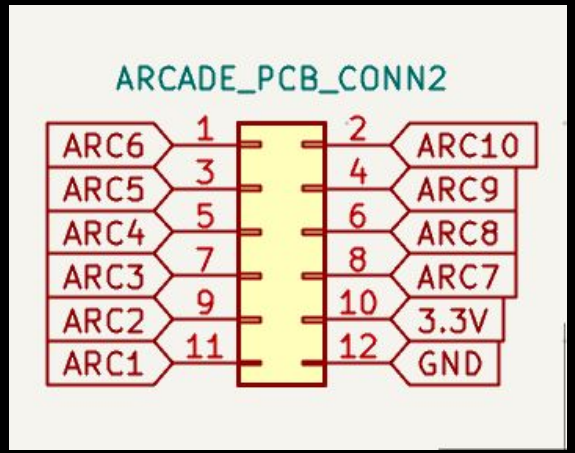
Peripheral Interface PCB - *Layout*

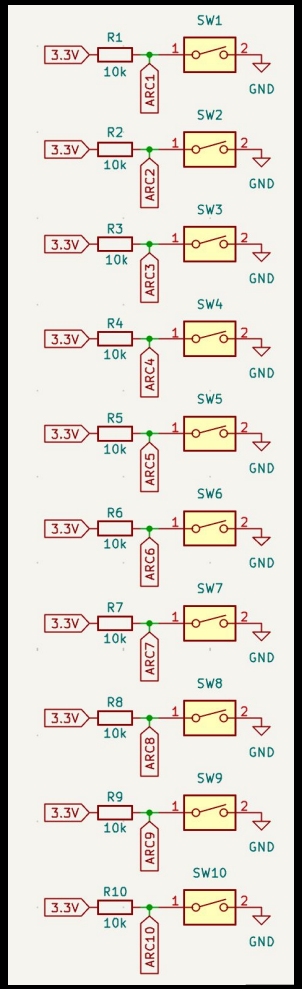






Arcade Controller PCB - *Schematic*



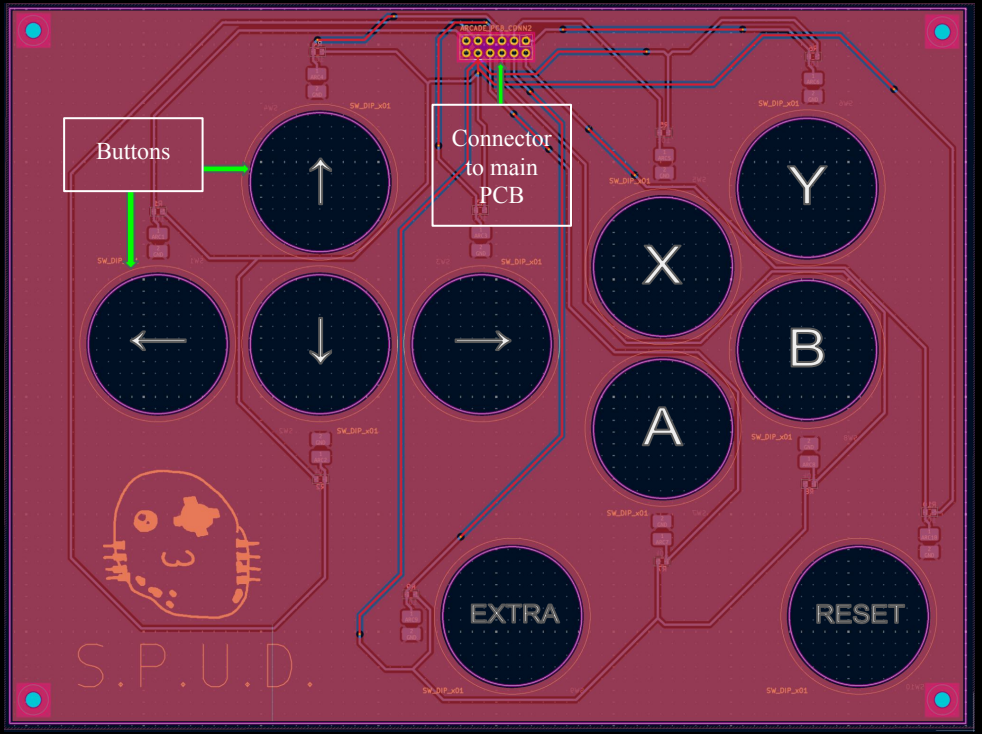


Powered by FPGA at 3.3V.

10 GPIO buttons for controlling S.P.U.D. in demo.

Connects directly to peripheral interface board.

Arcade Controller PCB - *Layout*



Software Component Research

Software Comparisons - *Instruction Set Architecture*

Feature	RISC-V	ARM	X86
Architecture Type	RISC	RISC	CISC
Instruction Length	Fixed 32-bit (base), supports 16-bit compressed	Fixed 32-bit (ARM), 16-bit (Thumb)	Variable length (1-15 bytes)
Extensibility	Highly extensible	Limited extensibility	Limited extensibility
Instruction Encoding	Simple, fixed format	Fixed format	Complex encoding
Address Modes	Simple	Multiple addressing modes	Complex addressing modes
Open Source	Yes	No	No
Market Adoption	Growing	Widespread	Dominant in desktop/server

Technology Selected - RISC-V

Feature	RV32IM	RV64F	RVDFD
Instruction Set Type	Base integer instructions	Single-precision floating-point	Double-precision floating-point
Registers	32 integer registers	32 integer + 32 floating-point	32 integer + 32 floating-point
Advantages	Simple and compact for embedded systems	Enhanced arithmetic precision for scientific applications	High precision and fused operations for advanced computing

ISA Selected - RV32IM

- RISC-V: Open-source ISA, no licensing fees, modular and customizable.
- ARM: Proprietary RISC, power-efficient, dominant in mobile/embedded.
- x86: CISC from Intel/AMD, dominates desktop/server markets

Software Comparisons - *Source Code*

- RV32IM core: has not been tested on FPGAs
- RV32I: tested for FPGAs, but limited usability; no multiplication support
- UltraEmbedded RV32IM core: trusted source, strong documentation, tested on FPGAs, has the M extension

Selection	RV32IM Core	RV32I (Single & Multi-Cycle)	UltraEmbedded RV32IM Core
Architecture	Five-stage pipeline with forwarding and hazard detection	Single-cycle or multi-cycle implementations	Two/three-stage pipeline (configurable) with optional caches, MMU, and AXI bus support
Complexity	More complex than SPUD's needs, but includes useful optimizations	Well-balanced; single-cycle design with an option for multi-cycle	More complex than the other cores, but configurable for FPGA SoCs
FPGA Compatibility	Not tested on hardware, simulation only (ModelSim)	Designed for FPGAs	Verified on similar (but not the same) FPGA
Testing & Verification	Functional simulation performed, but no FPGA testing	Well-organized and has all the necessary testbenches	Includes extensive testbenches and a working SoC reference design (booting Linux!)
Documentation	Limited, originally a school assignment	Clear structure with licensing information (MIT License)	Well-documented repository
Extensions	RV32IM (includes multiplication and division)	RV32I only	RV32IM (M-extension for multiply/divide), optional caches, MMU, and peripheral support

Source Code Selected - UltraEmbedded RV32IM

GitHub

Software Comparisons - *Toolchain Selection*

- Vivado: must-use for Xilinx FPGA
- RISC-V GNU toolchain: big community, trusted source, essential for converting c programs -> machine code
- OpenOCD: additional tool for extra debugging support, not vital since we can make our own debugging programs

Selection	Vivado	RISC-V GNU Toolchain	OpenOCD
Primary Purpose	FPGA design and simulation	Software development for RISC-V	Debugging and flashing
Features	- FPGA synthesis and implementation - Hardware programming - Simulation support	- GCC compiler for RISC-V - Binutils for assembling and linking - GDB for debugging - Cross-platform support	- On-chip debugging - FPGA programming support - Software loading onto the processor
Role in SPUD	Implements and verifies the CPU on the FPGA, manages design synthesis and simulation	Compiles high-level software (including games) into executable machine code for SPUD's CPU	Facilitates debugging and flashing the CPU design onto the FPGA, ensuring software and hardware interaction works correctly
Reason for Selection	Needed for FPGA implementation and simulation	Open-source and widely supported; provides all the necessary tools for RISC-V software development	Enables real-time debugging but at the cost of significant extra time and effort

Toolchain Selected - Vivado & RISC-V

Software Comparisons - *PCB Design Software*

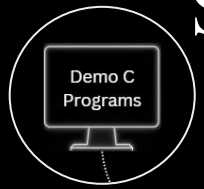
- Fusion360: easy to learn and good component libraries, but only free for students
- KiCad: small learning curve, free, and open-source with a large selection of component libraries
- Altium: has more advanced features in comparison, but not necessary for our build while also being expensive

Feature	Fusion360	KiCad	Altium Designer
Cost	Free for students	Free	Subscription based
Ease of Use	Easy	Moderate	Intuitive
Routing Tools	Basic	Manual	Advanced & constraint-driven
Component Libraries	Good for common parts	Open-source	Large selection of verified parts
3D Visualization	Basic 3D support	Basic 3D support	Advanced 3D visualization
Version Control	Cloud based (Fusion360)	Git	Altium 365 (Built-in)

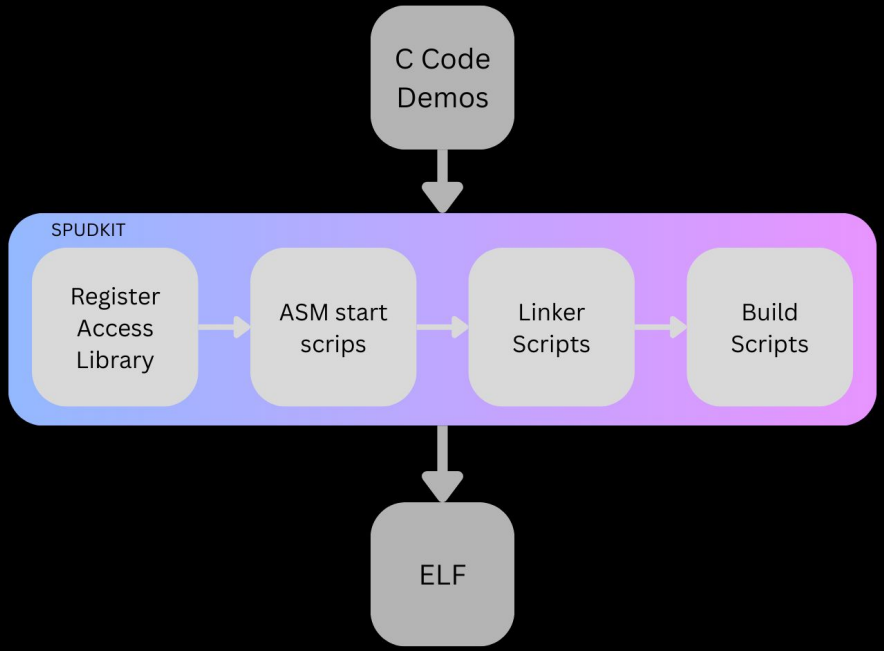
Technology Selected - KiCad

Software Design

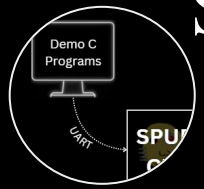
Software Design - *Development*



- Spudkit is a bundle of tools and software that build code designed to be ran on SPUD
- A C library is created to provide easy development to access SPUD's hardware
- Linkers scripts are written for our build scripts memory addressed like the main entry point address stack pointer locations
- RISC-V Cross compiler toolchains generate the ELF file to be loaded onto SPUD

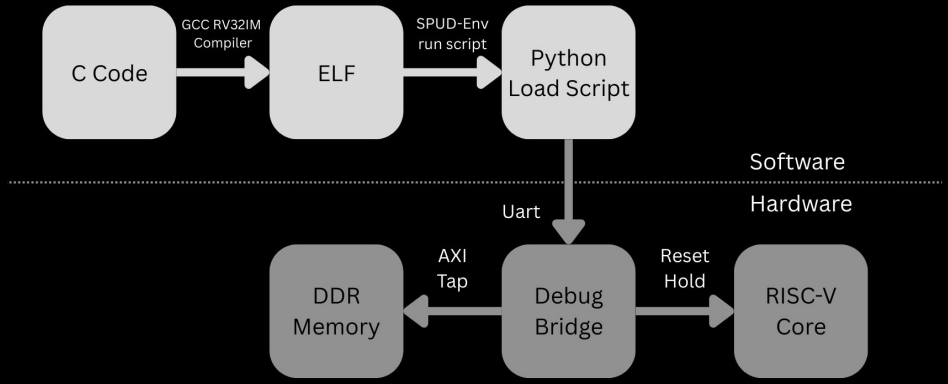


Spudkit flow diagram



Software Design - *Program Loading*

- Each demo is written in C code and uses the GCC RV32IM compiler
- An .elf file is generated using linker scripts that properly configure code for our processor
- A custom python script is used to command the processor into debug mode
- The debug bridge will hold the CPU into reset and hijack the uart port
- Through Serial/UART we load the machine code into DDR3 RAM via the debug bridge



Software upload flow diagram

Communication Protocols

System Level

AXI4 / AXI4-Lite (Within the SoC):

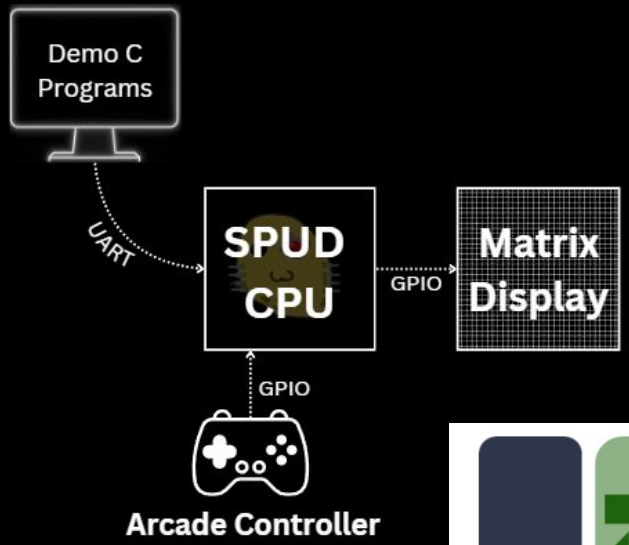
- Used for on-chip communication between the CPU, memory, and peripherals
- Bus interface for read/write transactions with handshake

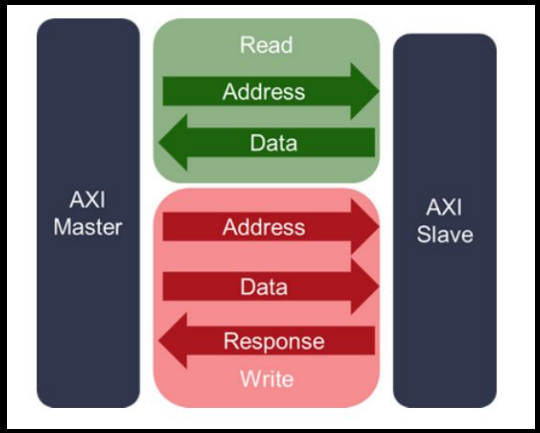
UART:

- Serial communication between FPGA and a host computer
- Debugging, program loading, or console output

GPIO:

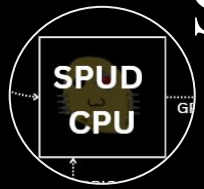
- Handles external input and output signals
- Interfaces with arcade controller and matrix display



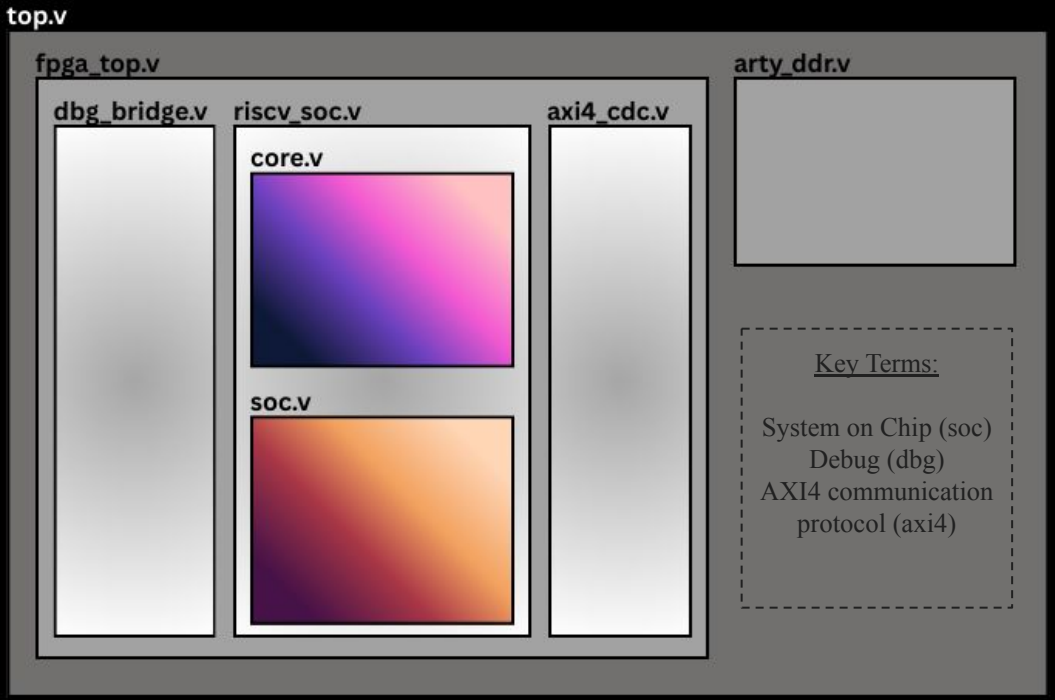


Software Design

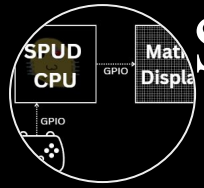
Software Design - *CPU Top Level*



- *arty_ddr.v:* RAM interface for the Arty FPGA
- *dbg_bridge.v:* used for debugging and accessing internal signals
- *riscv_soc.v:* houses the core from UltraEmbedded, and the modified SPUD soc
- *axi4_cdc.v:* handles *clock domain crossing* in the AXI



hierarchical, modular representation of SPUD's source code (Verilog)

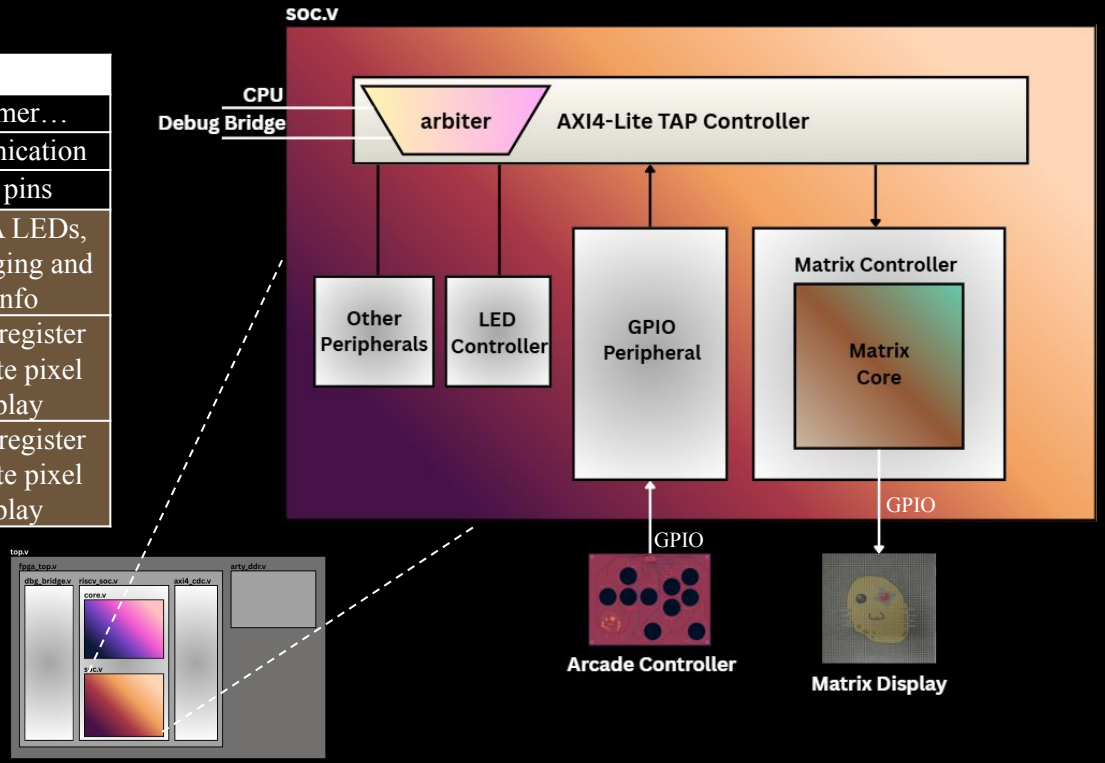


Software Design - *SoC RTL Design*

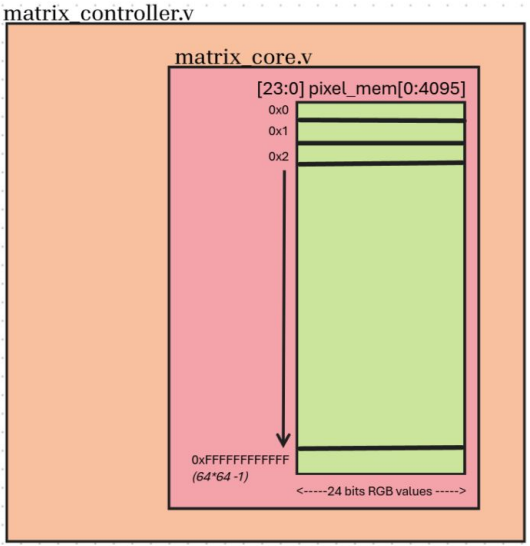
Table of Register-Mapped Peripherals

Peripheral	Base Address	Info
Other Peripherals	0x90000000	Interrupts, Timer…
UART	0x92000000	Serial Communication
GPIO	0x94000000	General I/O pins
LED Controller	0x95000000	Controls FPGA LEDs, used for debugging and processor info
Matrix Controller (PIXEL DATA)	0x96000000	Provides CPU register access to update pixel data on display
Matrix Controller (PIXEL ADDRESS)	0x96000004	Provides CPU register access to update pixel data on display

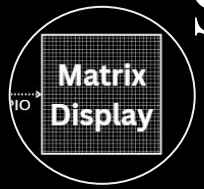
highlighted peripherals are designed specifically for SPUD



SoC Architecture - Matrix

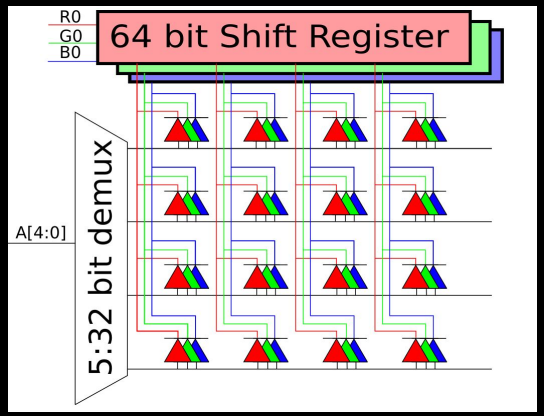


Software Design: *LED Matrix*

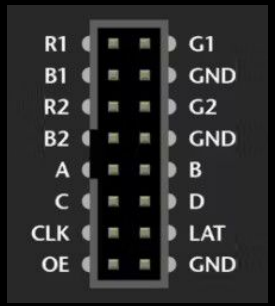


The 64x64 LED matrix utilizes an interface called HUB75

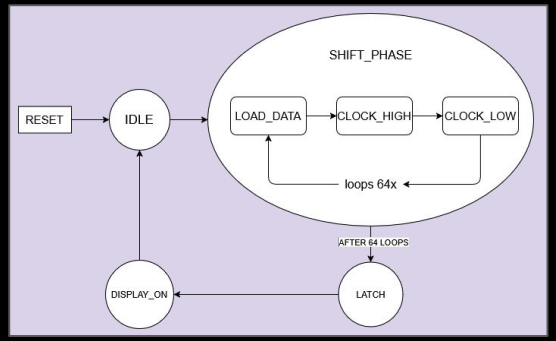
- 6 RGB pins, 2 for each color (R1,R2,G1,G2,B1,B2)
- 5 addressing pins, used to choose which row to select. Addresses one row per panel.
- Clock progresses shift register.
- Latch and OE for display and output control.
- Connects to the SOC video display interface
- State machine-based architecture.



From Ross Schlaikjer- RHYE.org



(HUB75 Pinout)



Software State Machine.

Prototyping and Testing

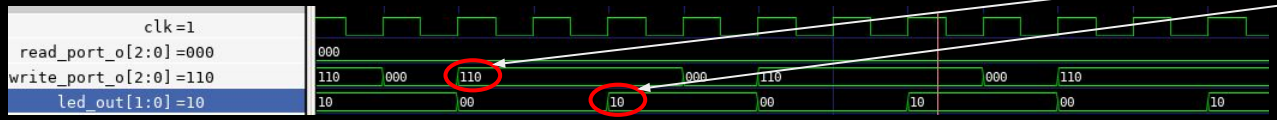
Testing - SoC Design

Tests:

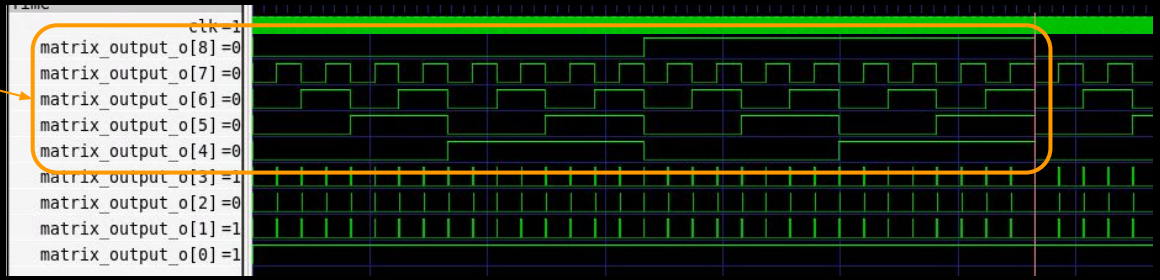
- FPGA Synthesis, Place & Route, Bitstream Generation
- Writing to FPGA LEDs from CPU using custom peripheral in SoC
- Using oscilloscope to measure MIPS via LED toggles
- CPU simulation using Verilator, an open-sourced simulator for Verilog
- Integration testing of software with custom peripherals (Arcade + Matrix Controller)

Results:

- Successful output for LED controller register. AXI bus decodes to correct peripheral number, 3'b110 (or 6), and updates the LEDs accordingly



- Successful iteration of matrix outputs (matrix_output_o[8:4])
  - ➢ 5 bits -> 2^5 = 32, the matrix has two vectors that iterate through half of the screen (64 pixels total)



Testing - SoC Design

Tests:

- FPGA Synthesis, Place & Route, Bitstream Generation
- Writing to FPGA LEDs from CPU using custom peripheral in SoC
- Using oscilloscope to measure MIPS via LED toggles
- CPU simulation using Verilator, an open-sourced simulator for Verilog
- Integration testing of software with custom peripherals (Arcade + Matrix Controller)

Results:

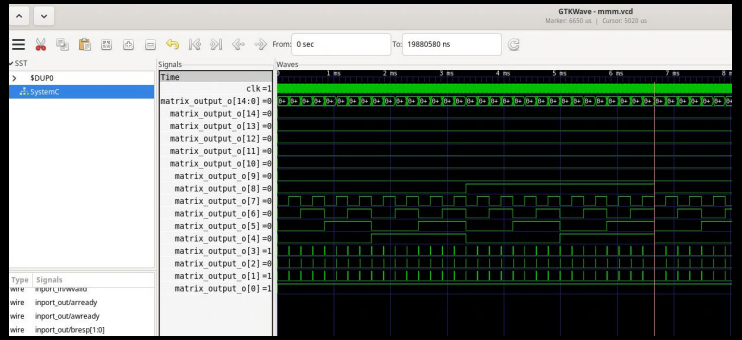
- 18.2 Millions of Instructions Per Second (MIPS) calculated via oscilloscope
  - measured delay between flag toggles
  - flag raised every N instructions (1M, 500k, 2M)
  - cross-checked results with simulation
  - # of Instructions / Time = MIPS

Test #	# of Instructions	Hardware Delay (s)	Hardware MIPS	Simulation Delay (s)	Simulation MIPS	Match
1	1M	0.055	18.2	0.055	18.2	Yes!
2	1M	0.055	18.2	0.055	18.2	Yes!
3	1M	0.055	18.2	0.055	18.2	Yes!
4	1M	0.055	18.2	0.055	18.2	Yes!
5	500k	0.028	18.2	0.028	18.2	Yes!
6	500k	0.028	18.2	0.028	18.2	Yes!
7	500k	0.028	18.2	0.028	18.2	Yes!
8	2M	0.11	18.2	0.11	18.2	Yes!
9	2M	0.11	18.2	0.11	18.2	Yes!
10	2M	0.11	18.2	0.11	18.2	Yes!

Testing - Matrix

Tests:

Refresh rate in simulation and physically. Time between full screen refresh = t, refresh rate formula : 1/t.



Results:

Test #	Full display refresh rate (ms)	Simulation display refresh rate (ms)	Results matching
1	6.57	6.65	Yes
2	6.67	6.65	Yes
3	6.7	6.65	Yes
4	6.66	6.65	Yes
5	6.65	6.65	Yes
6	6.69	6.65	Yes
7	6.69	6.65	Yes
8	6.68	6.65	Yes
9	6.55	6.65	Yes
10	6.71	6.65	Yes

Major Challenges - Initial Stages

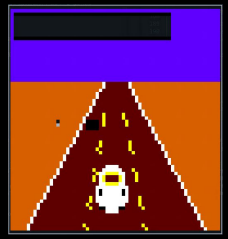
	Challenge	Solution
1	The RISC-V design is large and complicated	Extensive research in RISC-V, digital logic design, CPU architecture, SoC architecture
2	Shifting scope from designing an entire RISC-V core (all code) to something demonstrable	SPUD becomes an SoC customization and physical system with display and arcade controller for interactive demos
3	Verilator (simulator) setup was difficult to configure due to outdated dependencies and links	After a lot of trial and error with different tool versions and dependencies, we finally were able to install Verilator and documented the entire process
4	Vivado setup difficulties (different FPGA)	Discovered the constraints file was completely wrong -> upload proper constraint file for specific FPGA and update accordingly
5	Unstable connections to display caused by cheap bread board and cables	Search and found best bread board possible, could be better

Major Challenges - Middle Stages

	Challenge	Solution
1	Reading/writing to peripherals not working	Lots of debugging and collaboration between both hardware and software members... both ends had bugs!
2	Matrix controller architecture	Consulting our advisor, looking up similar architectures, getting creative
3	Matrix core integration into SoC - initially matrix core drove its pixels from inside itself	Update core to allow pixel data/address *inputs* rather than driving them itself
4	Creating custom build scripts to generate machine code	Leveraged RISC-V rv32i gcc toolchain to create ELF files (which contain machine code)
5	Spudkit library creation	Brainstormed essential library functions for SPUD (i.e. display_pixel_update, SPUD_LED_BASE, led_set, etc) and added more as we went along

Major Challenges - Final Stages

	Challenge	Solution
1	Integrating entire system (C demo -> machine code -> fpga setup -> running machine code -> display output	Double check all connections are intact, research how to connect to FPGA using WSL in Powershell, trial and error
2	Creating functional demos while sharing one FPGA	Created a shared server that everyone can work remotely from (one at a time)
3	Testing the matrix display output remotely	Utilize the UART display in the terminal (not perfect– color mismatch)
4	Testing MIPS: delay measurement in simulation was completely different than real-life	After a lot of debugging, we noticed a pattern in the data that led us to realize the clock configuration in the simulation was incorrect (100 MHz, whereas SPUD runs at 25 MHz)
5	Overwhelming use of resources via complicated c demo programs (“Spudracer”)	Create less resource-intensive games (i.e. “Spudman”)



Test demo, “Spudracer,” displayed over UART in the terminal. Example of resource overuse

Future Improvements

- Making improvements to the RISC-V core like branch predictor and cache levels
- Developing a more robust matrix code that fixes buggy edge cases and supports brightness
- Create a more robust PCB for better signal integrity and decrease IO usage.
- Add expansions to PCB for any extra peripherals like sound or multiple arcade controllers.
- Expanding games and make improvements to Spudkit for drawing shapes, grids, and 3D rendering.
- Better memory management tools and RTOS support.

A 3D spinning donut rendered on SPUD! Built with spudkit and using uart to simulate display output.

Administrative Content

Bill of Materials

Type	Component	Quantity	Total
FPGA	Arty S7-50: Spartan-7 FPGA	1	$199.00
PCB	Peripheral Board	1	$2
PCB	Controller Board	1	$15.80
LED Matrix	64x64 LED Display	1	$37.90
PSU	5V 4A power supply	1	$14.95
Power Adapter	Female DC Power adapter	1	$2.00
Level Shifter	HiLetGo 5pcs TXS0108E Conversion Module	1	$9.75
Button	Arcade Button - 30mm Translucent Red	10	$53.60
Connector	Harwin Gecko 12POS 1.25mm	2	$6.15
Total:			$341.15

Work Distribution

Team Member	Responsibilities
Gabriel Saborido *Computer Engineering*	ARTY S7 Expansion Board PCB and Power Management
	User Control Interface PCB
	Soldering and Assemble
Mohamed Moussa *Computer Engineering*	Matrix display interface
	Bit Shift Display Control Logic
	Display Control C library development
Frank Laterza *Computer Engineering*	Team Lead, overseeing all tasks
	System integration and infrastructure setup
	Demo C library development (Spudkit)
Valentina Terry *Computer Engineering*	SoC design customization
	GCC RISC-V toolchain setup
	CPU simulation and testing

Thank you!

~ The SPUD team 🫠