Kathmandu University
Department of Computer Science and Engineering
Dhulikhel,Kavre

A Project Report

on

"FPGA Console"

COMP 342

for partial fullfullment of III/II in Computer Science/Engineering

Submitted By
Ashish Thapa(56)
Aditya Jha(20)

Submitted To
Dr. Rajani Chulyadyo
Department of Computer Science and Engineering

Submission Date:

May 15, 2023

Bonafide Certificate

This project Work on
"FPGA Console"
is the bonafide work of
"Ashish Thapa, Aditya Jha"
who carried out the project work under my supervision

Project Supervisor

Name: Dr. Sudan Jha
Academic Designation: Professor

Abstract

FPGA technology has been widely used in digital systems, especially in embedded systems. Tang Nano 9k is a low-cost FPGA board that can be used to implement digital systems. The aim of this project is to create a custom CPU focused to act as a brain for a game console using Tang Nano 9k FPGA board. This project aims to provide hands-on experience with FPGA development and custom ISA implementation.The project uses open source tools for digital design like Yosys,synth_gowin,next pnr-gowin,iverilog, GTKWave etc. The project includes the development of a CPU with a custom ISA, buttons as input device, SSD1306 OLED as a output device, and ESP8266 as a display controller. Various protocols like UART, SPI, and I2C are used to communicate with various modules within the fpga console. The outcome of this project is a working game console where assembly code for games like pong can be loaded into the flash and played. The project has provided a better understanding of communication interface, fpga toolchain and implementation of custom ISA.This project can be extended to include more advanced features such as networking capabilities, sound output, and additional input devices.

Keywords: *FPGA technology, game console,, Yosys, nextpnr-gowin, GTKWave, custom CPU, ,UART, SPI, assembly*

Contents

List of Figures

List of Tables

Abbreviation

Acronym	Description
FPGA	Field-Programmable Gate Array
ISA	Instruction Set Architecture
OLED	Organic Light Emitting Diode
ESP	Electronic Stability Program
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
CPU	Central Processing Unit
VGA	Video Graphics Array
HDL	Hardware Description Language
DSA	Domain Specific Architecture
RISC	Reduced Instruction Set Architecture
CISC	Complex Instruction Set Architecture
ASIC	Application Specific Integrated Circuit

1 Introduction

1.1 Background

Field Programmable Gate Arrays (FPGA) are integrated circuits that can be programmed to perform any digital function by configuring the interconnects and logic gates inside the chip. They are different from processors or microcontrollers as FPGA contains logic blocks and look up tables that are placed and routed , which means there is no instruction set and fetching, decoding or executing. It can be optimized for specific tasks and it allows parallel processing .Since its inception in Xilinx has seen rapid growth and adoption mainly because of its high configurability and flexibility.

Since FPGA performance is comparable to real digital circuit, and the toolchain it comes bundled with allows easy testing and prototyping. Building the CPU with custom Instruction set becomes relatively easy. A CPU has a predefined set of instructions one can use to program it. Those instructions or Instruction set architecture are of two types i.e Complex Instruction Set Architecture(CISC) and Reduced Instruction Set Architecture(RISC). RISC has limited number of instruction set and simpler microarchitecture. RISC-V contains 32-bit integer instruction set called **RV32I**, which includes 47 instructions only [arvindpdmn, 2018]. Other Popular Instruction set architectures are x86, ARM , MIPS, Xtensia etc.

Gaming Consoles have been around 1970s and has evolved significantly over the years. The first game console, Magnavox Odyssey, was released in 1972 was a simple analog device. Today consoles like PS5, Xbox, Nintendo Switch are sophisticated devices with advanced features. The project aim is to create a simple enough game console using an FPGA which offers flexibility and customization. This approach allows us total control over how the console is designed. The Instruction set architecture can be specifically designed to cater console devices.

1.2 Objectives

- To create a simple game console using the FPGA, with simple input and output devices.
- To Design and implement our own CPU with custom Instruction Set Architecture(ISA) which can be used to write the games for the console
- To understand and use various hardware communication interfaces like UART, SPI , I^2C
- To develop proficiency in programming with Hardware Description language(HDL) like Verilog

1.3 Motivation and Significance

FPGAs are highly versatile programmable devices which can perform any digital function. It also generally supports large array of input/output pins which can interface with other deivces. The motivation for this project is to create a custom game console from scratch using FPGA technology, with a focus on developing a CPU with a custom instruction set architecture (ISA) and implementing it in hardware.

Learning to create a game console or any hardware system from scratch is arduous task. Newer ones are very complex and the old ones or the simple ones either have a closed system architecture or requires different Integrated Circuits(ICs) which are already discontinued to be produced. For those who wants to understand how simple consoles work, there is no proper resources available. This project aims to be as simple as possible and include bare minimum number of input devices, some form of video output while documenting each and every significant part of the process. By creating a custom game console, this project provides a platform for hardware enthusiast to experiment and create new games with a custom instruction set architecture while utilizing parallelization of FPGAs.

This project is different from existing works in that it focuses on creating a custom game console from scratch, rather than modifying an existing system. Additionally, the development of a custom ISA allows for greater flexibility and optimization of game code. The use of FPGA technology also provides the ability to modify and upgrade the hardware, enabling the console to evolve over time.

Overall, the significance of this project lies in its potential to inspire further innovation in the field of hardware world, as well as its educational value in providing hands-on experience with FPGA technology, computer architecture, and digital circuit design.

2 Related Works

2.1 Architectures

Starting From the 1960s, when IBM had 4 computer lines, all of them had 4 different ISAs. Custom instruction set for each computer line which had similar task was not optimal so companies moved towards a standarizing a single instruction set architecture. Intel has x86 architecture, Softbank Group owns ARM architecture, University of California Berkely manages RISC-V architecture. These exisiting architecures represents culmination of years of development and is suitable for various kinds of tasks. Custom Instruction set Design also has gained considerable attention in recent years. As the increasing demands for performance arises. Custom design gives total control. One example of Domain specific Architecture (DSA) is Google *Tensor Processing Unit* first deployed in 2015.

RISC-V It is one of the opensource General purpose specification that can run operating systems, compilers and so on. It has a modular architecture.It supports little endian byte ordering system. Its primary features are

- completely open ISA which is free to academia and industry.
- 32-bit and 64-bit address space variants
- support for highly-parallel multicore or manycore implementation

ARM architecture is owned by Softbank group. First arm processor called ARM1 was developed by Acord computer in Britain for BBC. The architecture state for the ARM processor consists of 16 *32-bit* registers and the status register. In 1995 it introduced the *16-bit* thumb instruction set which makes the code size small but are less powerful.Later enhancement has added digital signal processing (DSP), optional floating point instructions, power saving and Single instruction Multiple data (SIMD) instructions. ARMv8 introduced a completely new 64-bit architecture.

Google TPU can run *Deep Neural Network* interface 15-to-30 times faster than contemporary CPU and GPU. It can achieve such performance because

- it has large 2-dimensional multiply units compared to contemporary CPU which has 13 to 18 times smaller multiply units.
- it uses domain specific architecture (DSA) specifically created to decrease the cost and increase the performance.

2.2 Consoles

2.2.1 Mister FPGA Project

Mister is an open source project that aims to recreate classic computers, video game consoles, and arcade machines using modern hardware. It uses an FPGA board called DE10-Nano which connects to TV or monitor via HDMI video out. It can also be expanded with various add-on boards such as 7-port USB hub.

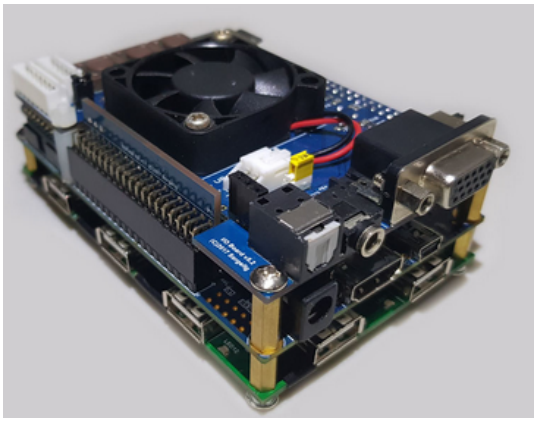

Figure 1: Mister FPGA DE10-Nano Board

2.2.2 FPGA-CHIP8

CHIP-8 is a virtual machine with an interpreted environment widely used in the 1970s. FPGA-CHIP8 implements CHIP-8 specification for TinyFPGA BX board. Other hardware components are a 16-button keyboard and 128x64px monochrome OLED screen

Figure 2: FPGA-CHIP8 running Space invaders

2.2.3 FPGAboy

FPGABoy is an implementation of Nintendo's classic handheld Game Boy system. It runs on Digilent Atlys board. Xilinx ISE webPACK is used as IDE and for simulations

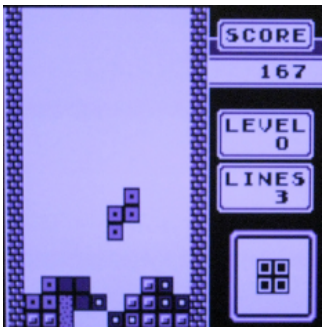

Figure 3: Tetris run on FPGAboy

3 Design and Implementation

3.1 Procedure

3.1.1 Learning oss-cad-suite

OSS CAD Suite is distribution for open source software used in digital logic design. There are tools for RTL synthesis, hardware verification, place and route and support for Hardware Definition Language(HDL) like Verilog. HDL-Bits is collection of verilog exercises suited for learning Verilog. `https://hdlbits.01xz.net/wiki/Special:VlgStats/216F4A6FBFA9331` contains the list of solved problems. *Yosys* is the synthesis tool which converts the high level description of design into optimized gate-level netlist. Netlist is list of electric component in a circuit and a list of nodes they are connected to.

Nextpnr-gowin is used for place and route. Place and route is the stage of FPGA which takes a *constraint* file and the synthesized file and then decides where to place and route them according to FPGA type. Constraint files are special files which defines the relation between the pins and module input and output variables and then decides where to place and how to route them. *gowin_pack* is used for Bitstream generation. *OpenFPGALoader* is used to load burn the generated bitstream into the FPGA .

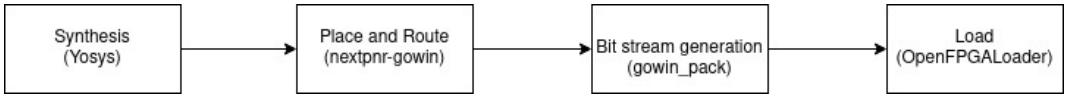

Figure 4: Process of loading HDL Code to FPGA using OSS tools

3.1.2 Learning Communication Interfaces and Protocols

Communication interfaces defines the set of protocols, and communication channels that allows devices to exchange data or signals with each other.

3.1.2.1 Interfacing with flash memory with SPI interface The flash chip used by the FPGA is *P25Q32U* which uses SPI protocol. Serial Peripheral Interface(SPI) is synchronous serial communication interface commonly used for short-distance communication. There are four lines *SCK* ,*Master Out Slave In(MOSI),Master In Slave Out(MISO)* and *ChipSelect(CS).*

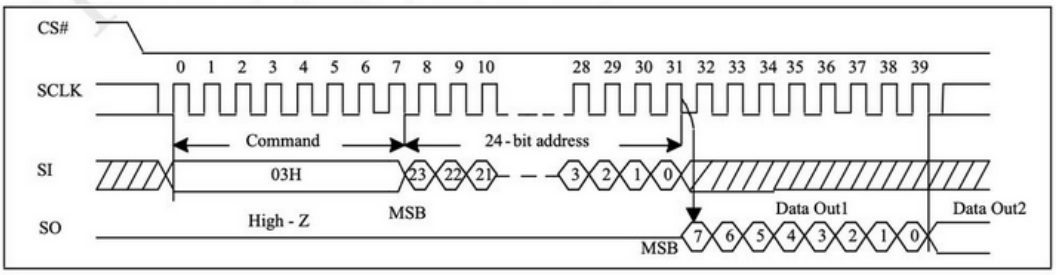

Figure 5: SPI Read Sequence Timing Diagram for Tang Nano 9k

Using verilog, SPI state machine was created. There needs to be certain delay time to accomodate for the speed difference between FPGA and the chip itself. The The states are shown by *figure 6*. Each node has *input/output* representation. Input section defines the path to take, i.e if past output was 0 then current node will take the path that has $0/X$ as the input.

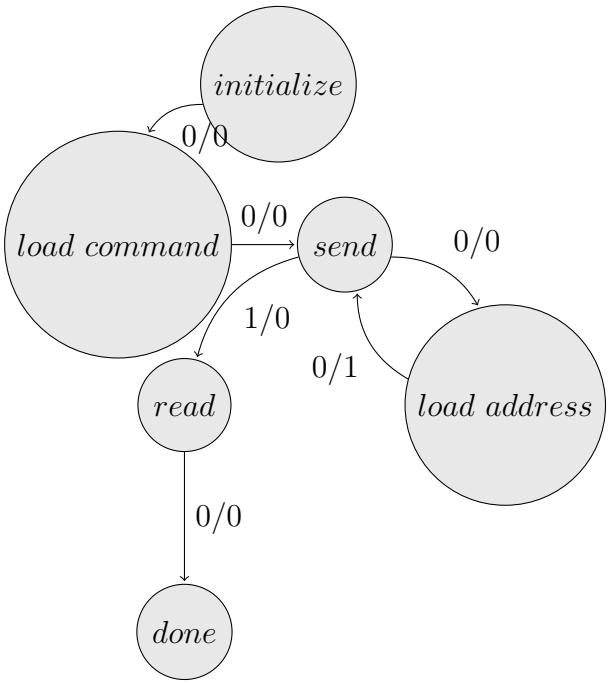

Figure 6: State diagram for SPI Flash Protocol

3.1.2.2 Interfacing with Screen using UART interface

Universal Asynchronous Receiver/Transmitter allows two devices to communicate using serial Data. It is simple protocol and works by sending one bit of data at a time. Baud rate must be same on both transmitting and receiving devices. The first bit or start bit tells the receiver that the data is coming and then the last bit is stop bit which tells the data has stopped.

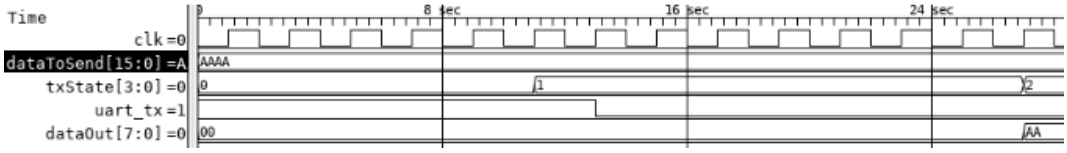

Figure 7: UART Transmit Timing Diagram

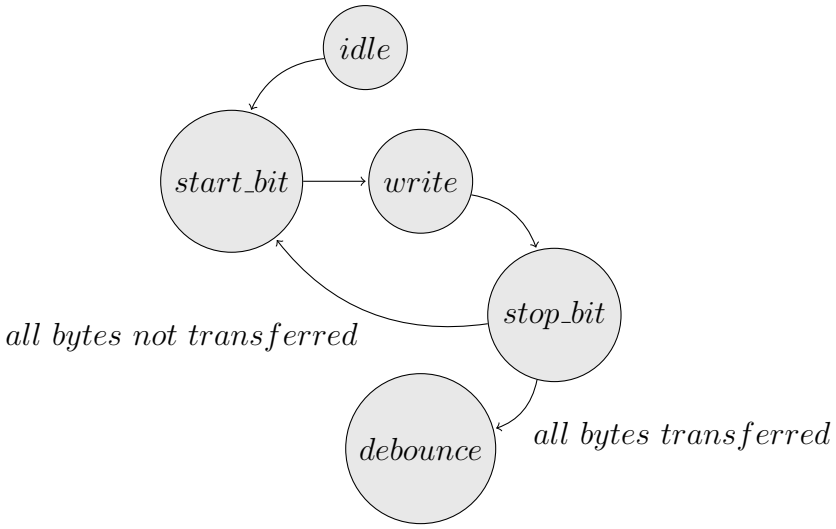

Figure 8: State diagram for UART Transfer Protocol

UART was used for communication between Screen interface and FPGA. The screen interface contained ESP8266 Microcontroller and a 128x32 OLED screen. 115200 was set as the baud rate between both microcontroller and FPGA.The message passed by the FPGA to screen interface is present on *Table 1*

UART Message	**Description**
RECT X Y W H	build the rectangle of width W and height H on X and Y position.
CIRC X Y R	build the circle of Radius R on X and Y position.
PIX X Y	build the pixel on position X and Y
TEXT value	Show the pixel on screen
CLEAR	Clear everything on screen

Table 1: UART Message sent from FPGA to Screen Interface

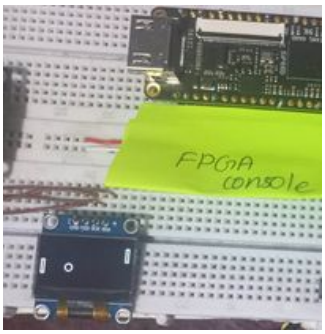

Figure 9: Screen rendering rectangle and circle element with basic screen commands

3.1.3 Building the CPU

Building the CPU with custom instruction set architecture requires several key steps. First the requirements of the intended CPU was laid out. The required commands were finalized. According to commands, instruction set architecture was built. Discussed Instruction Set Architecture(ISA) defines

- **Instruction Set**

  16 Bits of word size was chosen because around 12 instructions was planned. 5 Bits out of total word size can represent all the commands as $2^5 = 32 > 12$. Other 20 bits are reserved to add other cruicial features that will support procedures, label jumping etc. The first bit is *Command Modifier*(CM) bit which signifies either the addressing mode is intermediate or not. Rest of the bits are *variation* bits which either has the constant data attached to it, or signifies the variation . *Table* shows how variation, commands and CM are used for Add command.

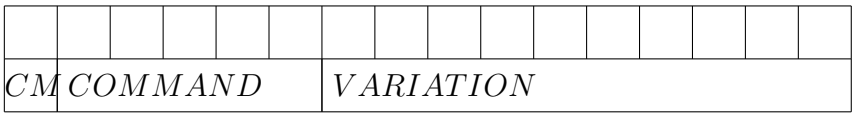

Figure 10: Layout for the 16 bit Computer Word

Instruction	**CM**	**COMMAND**	**VARIATION**	**Description**
ADD A	0	00101	1000000000	Add A to accumulator
ADD B	0	00101	0100000000	Add B to accumulator
ADD C	0	00101	0100000000	Add B to accumulator
ADD 30	1	00101	000000001	Immediate Add

Table 2: Instruction bits for Add Commands

- **Register set**

  There are 5 main registers A, B ,C , Accumulator(AC) and Program Counter(PC). Initially Registers hold 0.

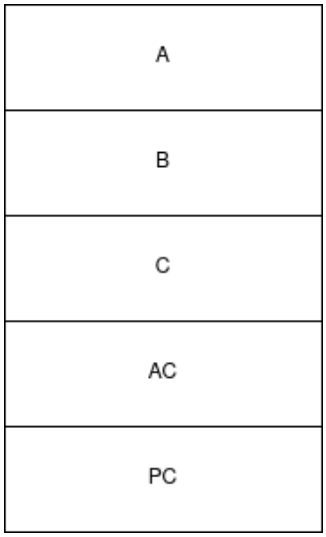

Figure 11: Registers

Other essential features of the CPU are Control Unit, Arithmetic Logical Unit and other I/O devices.

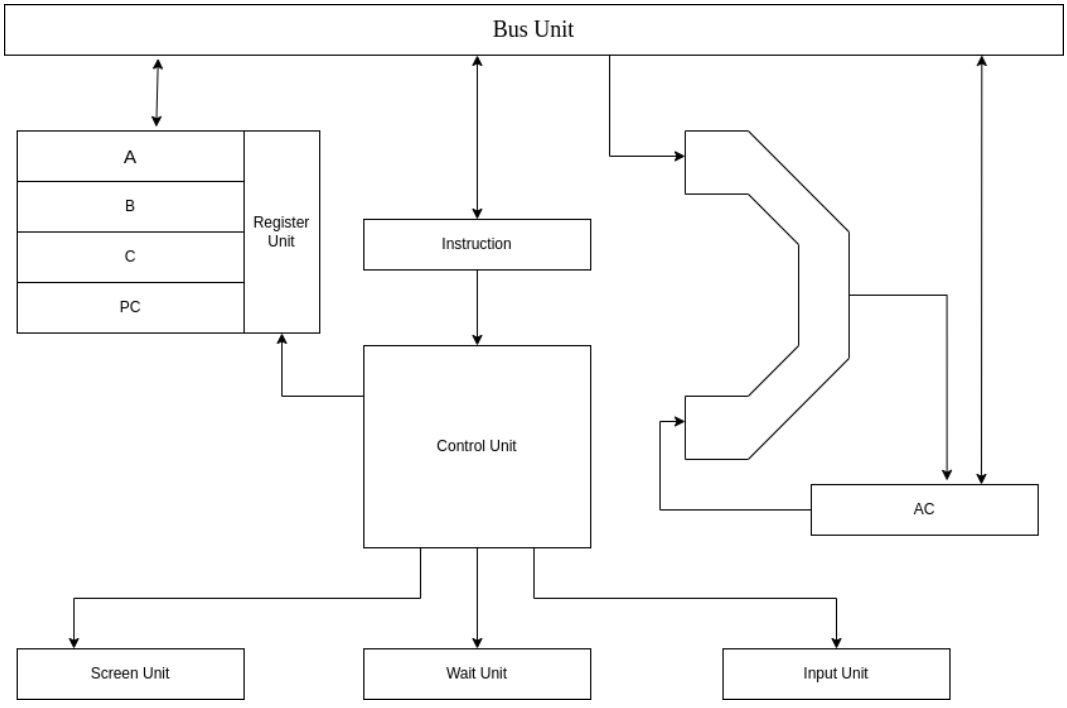

Figure 12: Architecture of the CPU

Input Unit is constantly monitored on each clock cycle. Interrupts are not implemented to CPU polls for each input device. Screen Unit and CPU are connected with each other through the UART protocol. The functionality of control Unit state machine is shown in the *Figure 13*.

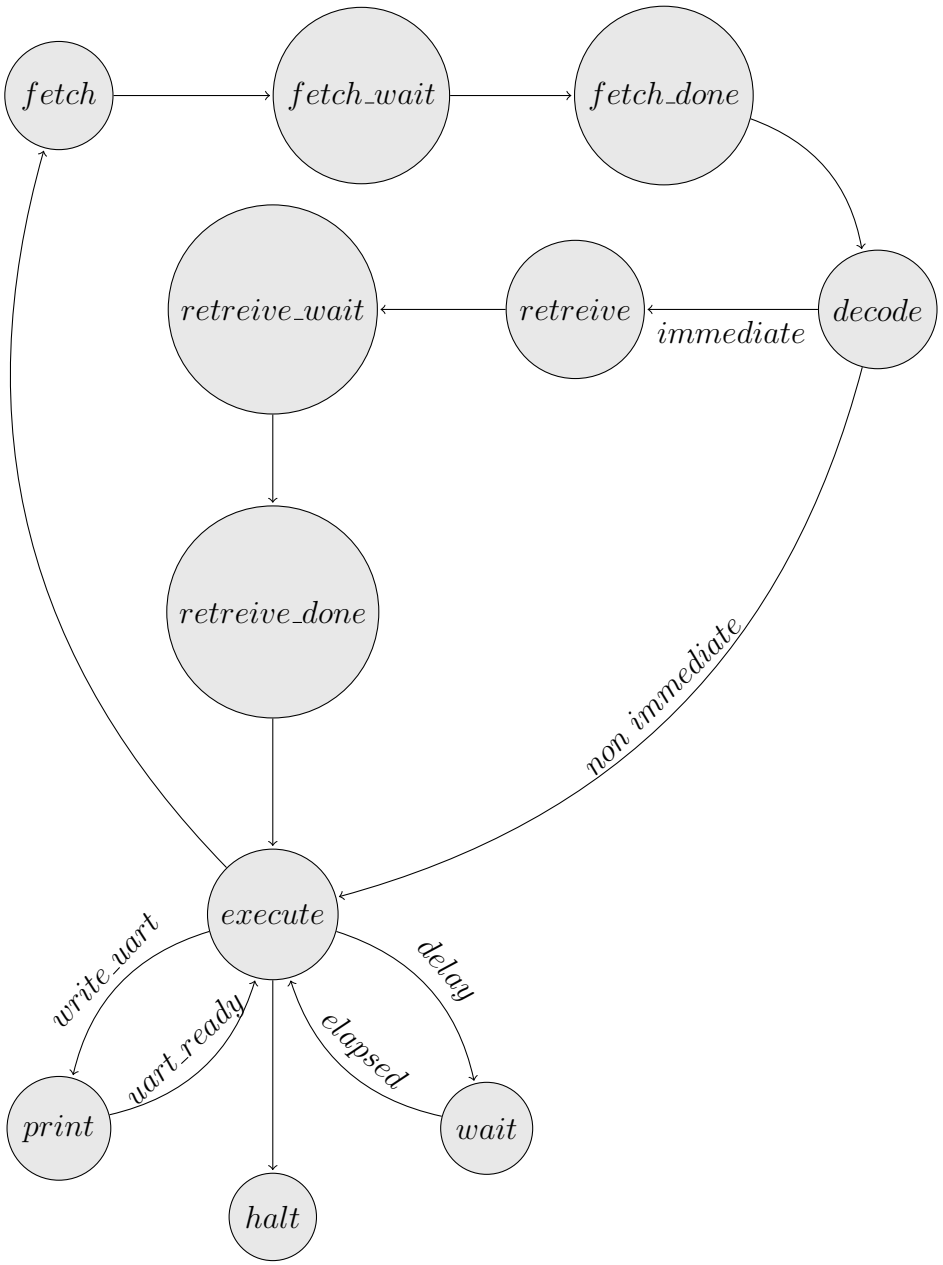

Figure 13: State diagram for CPU

CPU has total of 11 states in total. First state starts with a fetch. Other two fetch states fetch_wait and fetch_done are required because, reading from flash is not instantenous. The time where flash is being read represents wait time for flash and after data is read, flash has to send *dataReady* signal which represents fetch_done. Decode divides the command into *CM*, *COMMAND*

and *VARIATION* bits and checks if the instruction is immediate. If it is immediate then another 16 bit value has to be retreived which follows the same process as fetch, then it gets to execution stage. according to the instruction, it can either print, or wait, then it again goes to fetch stage.

3.1.4 Writing Assembler and Assembly code

An assembler is a program that translates assembly language code into machine code. *Table 3* contains all the possible instruction and its equivalent binary equivalent with description, which the assembler has to conver. Assembler was written in Rust which expects the filename as an argument. Each character is pushed to a dynamic array. It is then fed to the mattern matching function of rust which returns the binary executable for the CPU.

```
ADD 0 ; position of first bat 0 30
STA A
ADD 30
STA B

ADD 5 ; size of bat is 5 20
STA C
ADD 20

RECT ; RECT Areg Breg Creg AC

ADD 123  ; position of second bat 256 30
STA A
ADD 30
STA B

ADD 5 ; size of bat is 5 20
STA C
ADD 20

RECT; RECT Areg Breg Creg AC
```

Figure 14: Pong Game Assembly Code Snippet

3.2 System Requirement Specification

3.2.1 Software specification

3.2.1.1 Verilog FPGA was programmed using Verilog HDL. It compiles the code and provides a executable which can be simulated. During simulation we can configure it so that Value Change Dump(VCD) files can also be produced.

3.2.1.2 GtkWave GTKWave reads the VCD files, then allows us to inspect the timing variables, which is useful for testing the usability of verilog modules.

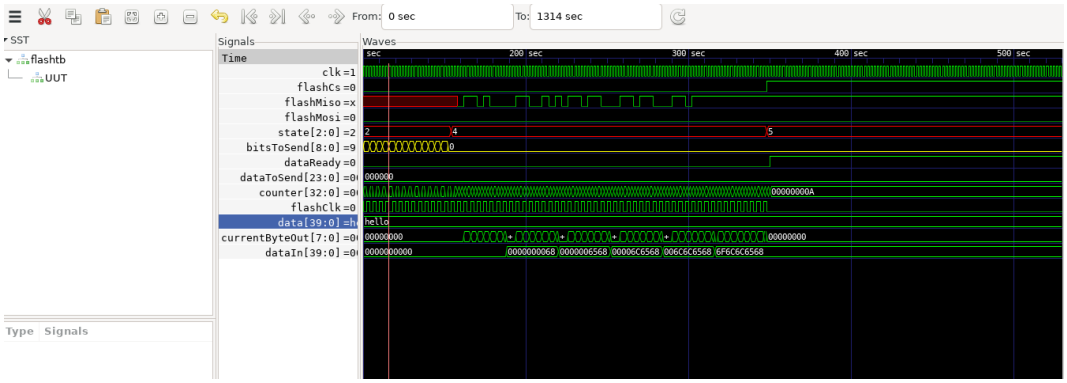

Figure 15: Gtkwave Timing Diagram for SPI Flash

3.2.1.3 Yosys Yosys is framework which synthesizes the verilog code and produces a netlist. A netlist is a file that contains the interconnections between various electric components of a circuit. So verilog code is described in digital logic form.

3.2.1.4 gowitn-pnr A constraint file or CST file maps the verilog variables with real pins present in FPGA. Gowin-pnr takes the constraint file then performs place and route which is suitable for Gowin FPGA.

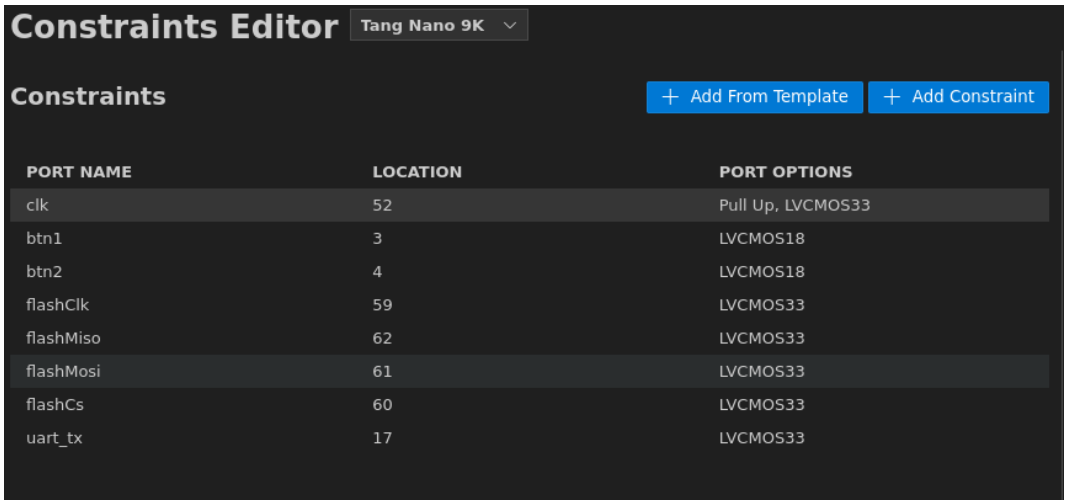

Figure 16: Constraint mapping between verilog variables and the corresponding pin location

3.2.1.5 gowin_pack gowin-pack reads the output of gowin-pnr and then produces the bitstream file that is suitable to be uploaded into fpga.

3.2.1.6 OpenFPGALoader OpenFPGALoader is a tool to write the bitstream file into FPGA.

3.2.1.7 Rust Toolchain Rust is a memory safe, blazing fast language which was used to create the assembler.

3.2.1.8 Espflash Espflash is a tool used to flash ESP microcontroller. It was used to flash screen interface code into esp8266 microcontroller.

3.2.2 Hardware specification

3.2.2.1 Tang Nano 9k It is open source FPGA board based on Gowin GW1NR-9 FPGA chip. It has 9k logic units, HDMI connector, SPI screen connector, SPI flash and 6 LEDs.

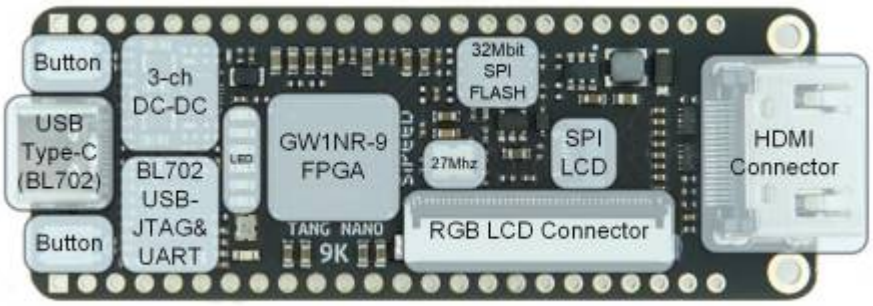

Figure 17: Tang Nano 9k

3.2.2.2 NodeMCU It is also opensource micrcontroller board which is based on ESP8266 chip. For this project the features that were important are USB-to-UART converter and GPIO UART.

Figure 18: Esp8266 Micocontroller

3.2.2.3 SSD1306 OLED Screen It is low power consuming 128x32 pixel OLED display controller.

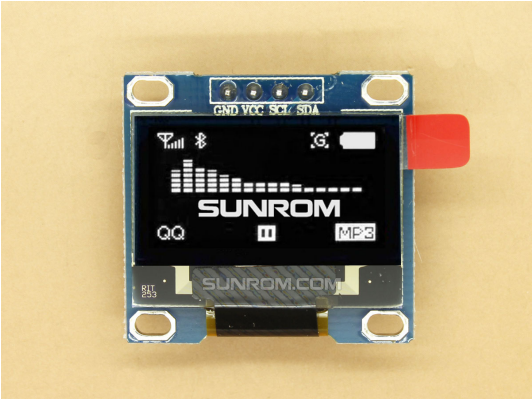

Figure 19: OLED Screen

4 Discussion on Achievements

Custom Instruction set Architecture was implemented and the functional game console was created using FPGA technology. The project also gave hands-on-experience with embedded systems. Core communication protocols like UART, SPI and I2C were also explored. Rigorous testing was done in order to find the errors on simulation phase. This project demonstrates the ability to program FPGA using open source digial design tools like Verilog, yosys, gowin-pnr etc. Other achievements include Reading Datasheets, writing simulation tests and interfacing with the device .

4.1 Features

Some Significant features of the project are listed below

- **Custom CPU**
  - RISC style CPU
  - 16 Bit word length
  - Supports up to 32 instructions
  - Two addressing modes supported
  - 3 usable Registers: A, B, C
  - 2 internal Registers : PC and AC
- **External Memory** P25Q32U is SPI supporting memory chip which is in built into FPGA itself. To load the program we can just burn the program into the flash.
- **State Machine** The paradigm with which FPGA is programmed is different as FPGA supports parallelization and can run multiple blocks all at once. To take advantage of this, core modules like Flash, UART are implemented as state machine.
- **Uses Open source toolchains only** Sipeed also provides its own proprietary IDE, But opensource fpga tooling and documentation is very good for TANG NANO 9k.
- **Communication Protocols**
  - Microcontroller and OLED screen communicate with I2C protocol
  - FPGA and Flash communicate with SPI protocol
  - FPGA and Microcontroller communicate with UART

5 Conclusion and Recommendatation

The project was a great way to gain hands-on expereince with FPGA developement and custom ISA implementation. The project can be improved further by removing the limitations discussed below

5.1 Limitation

- **Limited user interface/Input Device** The input device for the console consists of only 4 pushbuttons, which may not provide a rich user interface for playing games. This limits the types of games that can be played on the screen
- **Non Standarized custom ISA** There are reliant and practical architectures like ARM and RISC-V which would have allowed the CPU to run already available games and the CPU would be more reliant and deterministic.
- **Limited scalability** The project focuses on specific FPGA and specific OLED which means complex additions like HDMI, VGA would require significant changes to code.
- **Lack of Sound** Console doesn't have any audio output, which limits the gaming experience.
- **Limited Memory** The size of inbuilt flash is 4 megabytes. But due to architecture limitations. only 2^{16} bytes can be accessed.
- **Display Controller** Screen Unit is managed by microcontroller, but microcontroller can be completely eliminated. Due to time constraints I_2C communication couldn't be implemented in CPU.
- **Cost** FPGA is versatile, and it can act as any digital circuit. The tradeoff is it's price. FPGA is good for prototype phase or for projects where ASIC fabrication price is higher than that of FPGA integration.
- **Limited Instruction Set** Number of instructions is very minimal which code size long and some features is not supported.

5.2 Future Enhancement

- **Adding Support to more Input/Output Devices** USB support can be added which can allow range of controllers for input device. As for Output Devices we can integrate HDMI output and instead of hardcoding the screen resolution we can use clip space coordinates which will accomodate for all screen sizes.
- **Adding More Games** More games can be added, and selection menu can also be added where users can select and play from range of games.

- **Adding Sound Support** Interactivity is very important and for that we can add sound support which makes the game more immersive
- **Switch to RISC-V or ARM Architecture** Switching to such architecture allows the games to be portable. Games that are already compiled for such architecture will be able to run on the CPU itself.
- **Improve Performance** Currently on fetch cycle, only 16 bits is read from flash memory. Instead we can read more bytes from it.
- **Introduce Pipelining** Pipelining creates two different units inside the CPU. Execution unit will keep on executing and fetch unit will keep on fetching parallely, which increases the performance of the CPU.
- **Networking Capabilities** We can leverage chips like ESP8266 which already has implemented TCP/IP stack to create multiplayer games.
- **Create Operating System** Core features of operating system like memory management, network capabilites, exceptions, interrupts and paging can be implemented.

Bibliography

[arvindpdmn, 2018] arvindpdmn, h. (2018). RISC-V Instruction Sets.

[MAXIMO H. SALINAS et al., 1993] MAXIMO H. SALINAS, BARRY W. JOHNSON, and JAMES H. AYLER (1993). Implementation Independedt Model of an instruction set architecture in VHDL. IEEE.

[Patterson and Berkeley, 2018] Patterson, D. and Berkeley, UC, G. (2018). 50 Years of computer architecture: From the mainframe CPU to the domain-specific tpu and the open RISC-V instruction set. IEEE.

[Waterman et al.,] Waterman, A., Patterson, D., and Asanovic, K. Specifications – RISC-V International.

References

[1] List of CPU Architectures.

[2] The Magnavox Odyssey, First Video Game Console, January 1972.

[3] 50 Years of Computer Architecture: From Mainframe CPUs to DNN TPUs and Open RISC-V | IEEE Computer Society of Silicon Valley.

[4] Tang Nano 9K: Reading the External Flash, September 2022.

[5] Tang Nano 9K - Sipeed Wiki.

[6] Sipeed.

[7] Documentation – Arm Developer.

6 Appendix

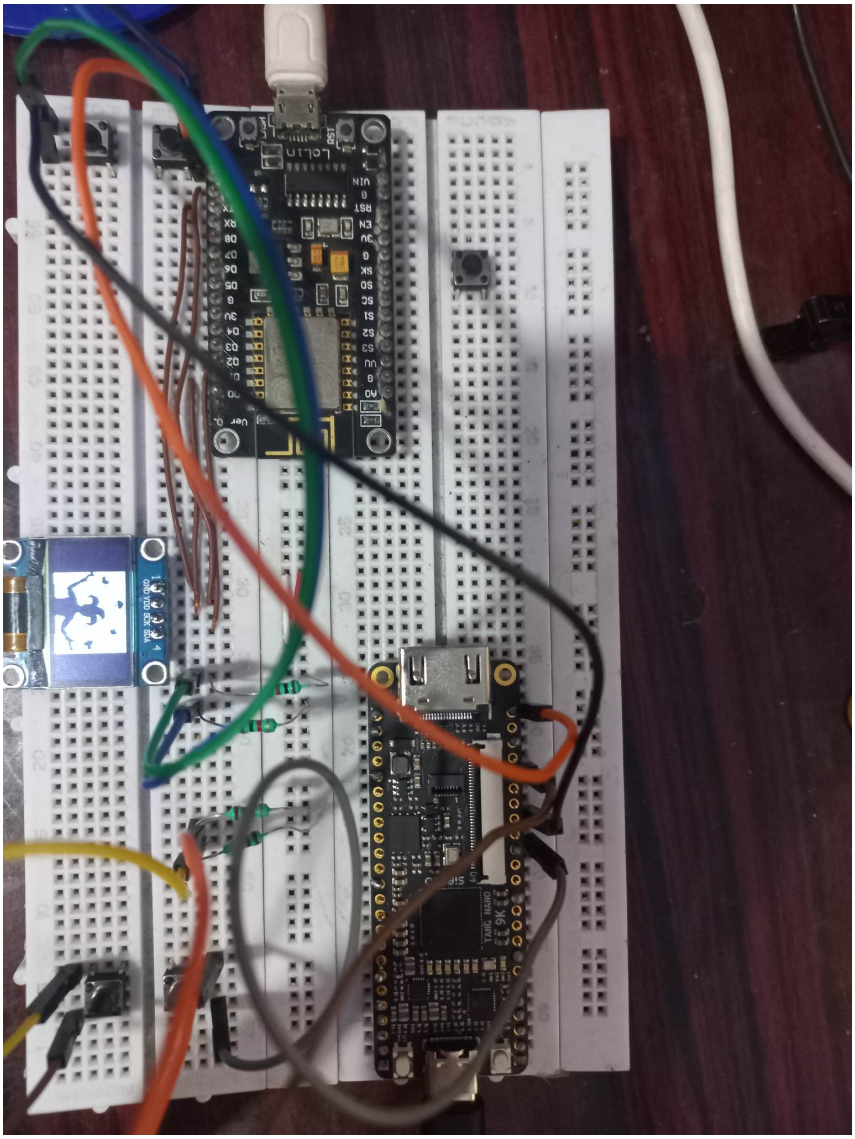

Figure 20: Game Console on Breadboard

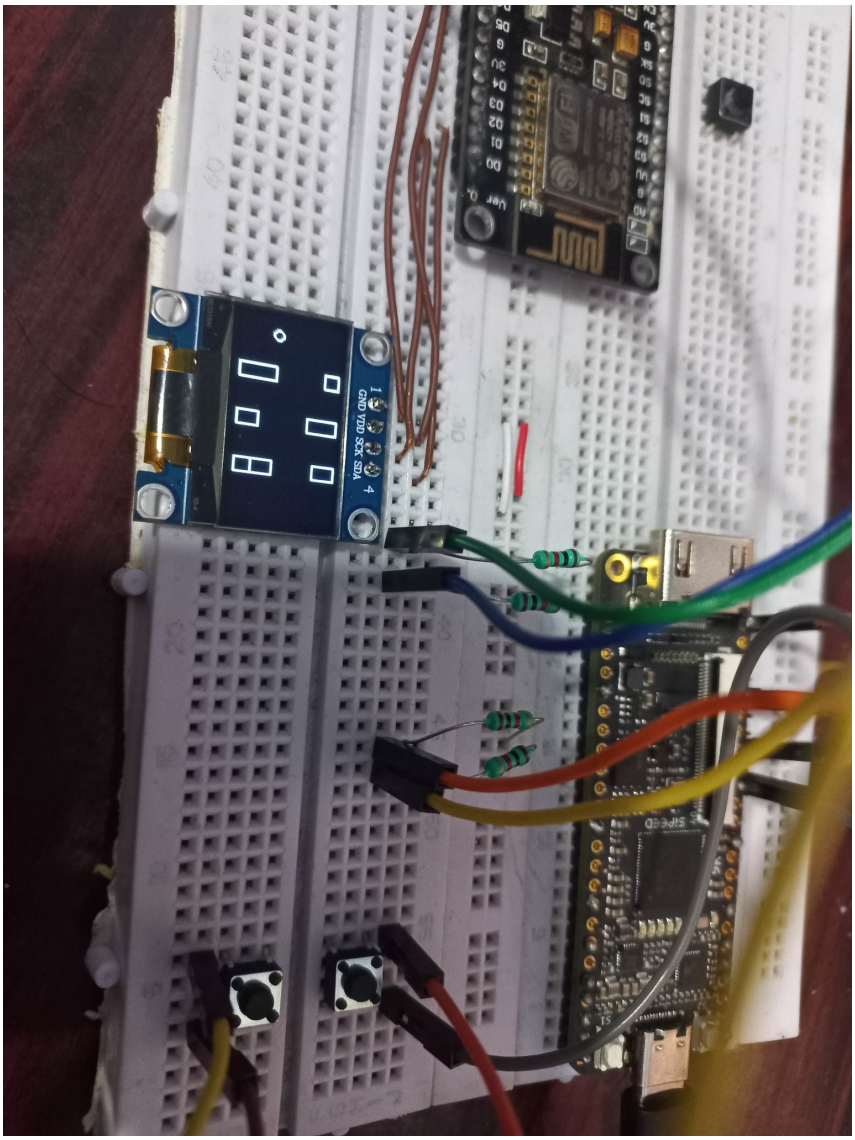

Figure 21: Flappy Bird style game on console

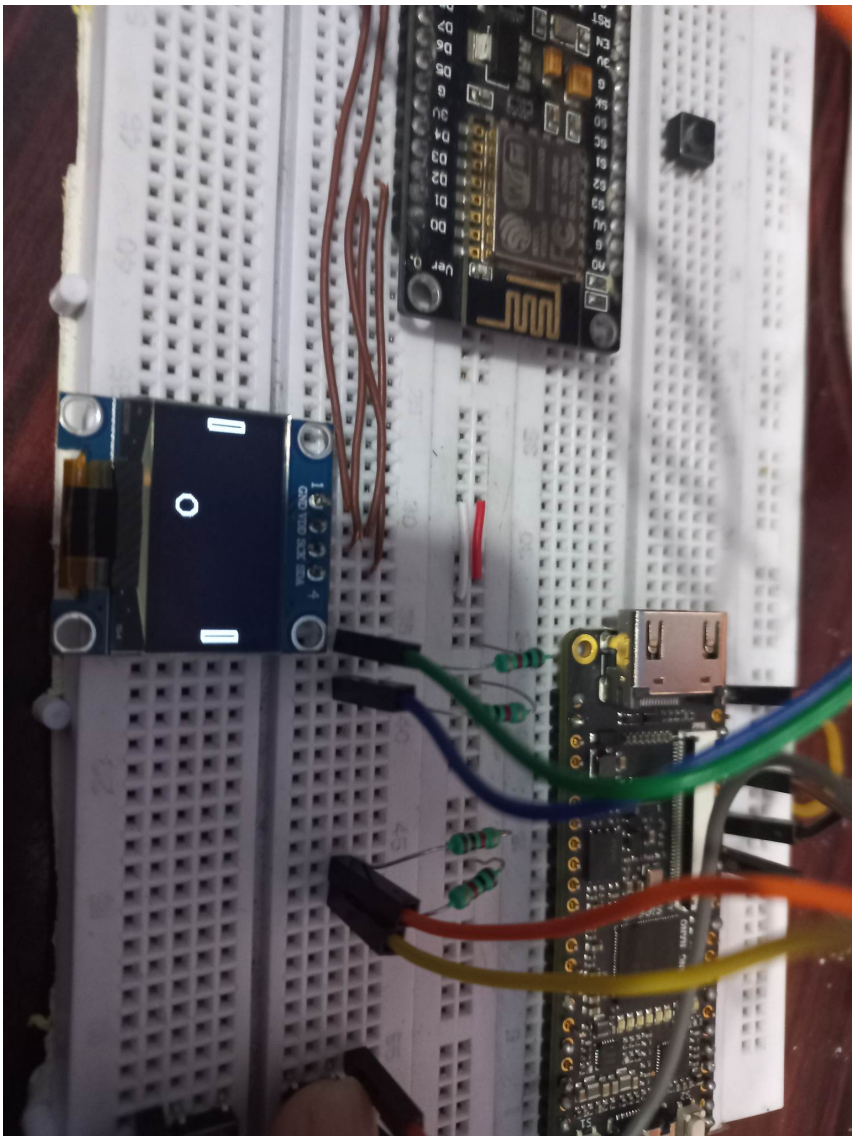

Figure 22: Pong style game on console

Command	constant	opcode	variation	description
CLEAR A	0	00000	1000000000	clear A Register
CLEAR B	0	00000	0100000000	clear B Register
CLEAR AC	0	00000	0010000000	clear AC Register
CLEAR BTN1	0	00001	1000000000	clear AC if btn1 pressed
CLEAR BTN2	0	00001	0100000000	clear AC if btn2 pressed
CLEAR BTN3	0	00001	0010000000	clear AC if btn3 pressed
CLEAR BTN4	0	00001	0001000000	clear AC if btn4 pressed
STA A	0	00010	1000000000	store AC in A register
STA B	0	00010	0100000000	store AC in B register
STA C	0	00010	0010000000	store AC in C register
STA LED	0	00010	0001000000	set first 6 bits of AC register to LED output
INV A	0	00011	1000000000	invert A register
INV B	0	00011	0100000000	invert B register
INV C	0	00011	0010000000	invert C register
INV AC	0	00011	0001000000	invert AC register
HLT	0	00100	0000000000	halt
ADD A	0	00101	1000000000	Add A to AC register
ADD B	0	00101	0100000000	Add B to AC register
ADD C	0	00101	0010000000	Add C to AC register
ADD 20	1	00101	0000000001 value	Immediate Add
SUB A	0	00110	1000000000	Sub A and AC register
SUB B	0	00110	0100000000	Sub B and AC register
SUB C	0	00110	0010000000	Sub C and AC register
SUB 20	1	00110	0000001010 value	Immediate Subtract
PRNT A	0	00111	1000000000	Print A reg to OLED
PRNT B	0	00111	0100000000	Print B reg to OLED
PRNT C	0	00111	0010000000	Print C reg to OLED
PRNT 110	1	00111	0110110110	Immediate Print value to OLED
RECT X Y W H	0 0	01000 00000	XXXXXYYYYY WWWWWHHHHH	Draw rect of in pos x and y of size w and h
CIRC X Y R	0 0	01001 00000	XXXXXYYYYY 00000RRRRR	Draw Circle of radius R in position X and Y
PIX X Y	0	01010	XXXXXYYYYY	Draw pixel in X and Y position
JMPZ A	0	01100	1000000000	Jump to A Register position if AC = 0
JMPZ B	0	01100	0100000000	Jump to B Register position if AC = 0
JMPZ C	0	01100	0010000000	Jump to C Register position if AC = 0
JMPZ 20	1	01100	0000000001 value	Immedate jump
WAIT A	0	01101	1000000000	Wait for amount of seconds as specified by A register
WAIT B	0	01101	0100000000	Wait for amount of seconds as specified by B register
WAIT C	0	01101	0010000000	Wait for amount of seconds as specified by C register
WAIT 20	0	01101	0000000001 value	Immediate Wait for amount of seconds

Table 3: Instruction set Architecture