Habib University

Dhanani School of Science and Engineering

Digital Logic and Design Fall 2023

EE/CS 172L/130L

T6

Instructor Haseeb Khan

Ifrah Chishti 08351

Izbal Mengal 08044

Antisham Uddin 08429

Hammad Malik 08298

Table of Contents

Introduction

We aim to recreate the classic game “Pong”. This version of the game is played by a single player, who moved the paddle on the screen up/down and has to bounce the ball off of it, without letting it get past and colliding with the wall behind it.

A player would start with 3 balls. Every time the ball hits the left edge of the screen, a ball is lost, and the ball respawns in the center of the screen, and score remains unchanged.

The speed of the ball increases as the score increases to increase the difficulty of the game.

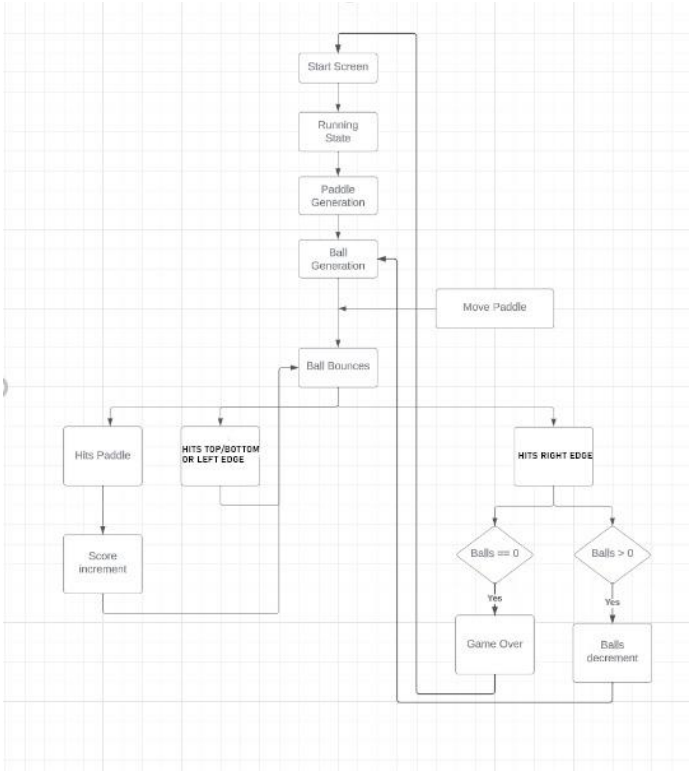
Once the player runs out of balls, the score resets 0 and the text “Game Over” is displayed.

The game is programmed using Verilog HDL and a keyboard is used to move the paddle up/down.

User Flow Diagram

The game starts at the start screen, then as soon as the player moves the paddle, the game starts.

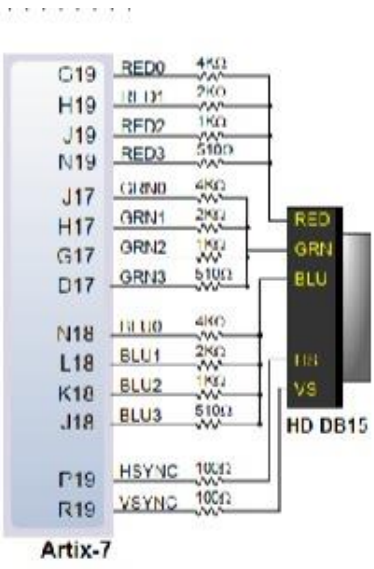
The ball initially starts moving and bounces of the walls/paddle. If it hits the paddle, the score increments by 3 points and the ball will continue moving as normal. If the ball misses the paddle and hits the left edge of the screen, the ball count will decrement by 1 if it is not already 0, and the ball will respawn at the center of the screen. If the ball count is 0 and the ball hits the left edge, the game over screen appears.



Input Block

The input will be taken from the keyboard, with the keys “A” and “S” being used to move the paddle up and down respectively.

a) VGA Pin Configurations



b) Input Ports Integrated in Verilog

ps2c and ps2d refer to the “A” and “S” keys on the keyboard

I/O Ports

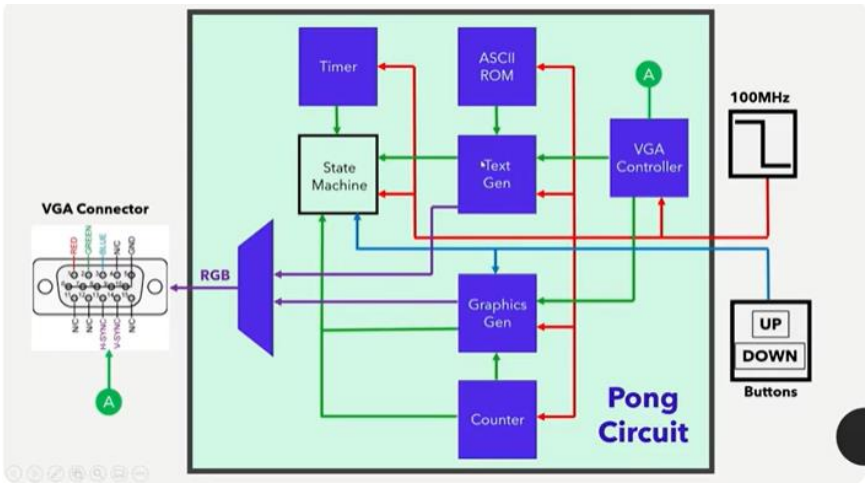
Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
All ports (18)						
rgb (12)	OUT			☑	14	LVCMOS33*
rgb[11]	OUT		G19	☑	14	LVCMOS33*
rgb[10]	OUT		H19	☑	14	LVCMOS33*
rgb[9]	OUT		J19	☑	14	LVCMOS33*
rgb[8]	OUT		N19	☑	14	LVCMOS33*
rgb[7]	OUT		J17	☑	14	LVCMOS33*
rgb[6]	OUT		H17	☑	14	LVCMOS33*
rgb[5]	OUT		G17	☑	14	LVCMOS33*
rgb[4]	OUT		D17	☑	14	LVCMOS33*
rgb[3]	OUT		N18	☑	14	LVCMOS33*
rgb[2]	OUT		L18	☑	14	LVCMOS33*
rgb[1]	OUT		K18	☑	14	LVCMOS33*
rgb[0]	OUT		J18	☑	14	LVCMOS33*
Scalar ports (6)						
clk	IN		W5	☑	34	LVCMOS33*
hsync	OUT		P19	☑	14	LVCMOS33*
ps2c	IN		C17	☑	16	LVCMOS33*
ps2d	IN		B17	☑	16	LVCMOS33*
reset	IN		T17	☑	14	LVCMOS33*
vsync	OUT		R19	☑	14	LVCMOS33*

Output block:

The diagram for the top level module is provided below:

The vga_controller has x and y values that go into the text and graphic generation modules which in turn generate RGB values that are then multiplexed and the selection for the multiplex will be based on the state machine and generated within the top module, and the mux output will drive to the VGA connector. A counter will keep track of the score and the counter module will communicate with the graphic generation circuit and the state machine. A timer will be used to reset the ball for the game over state at the end. The state machine will have four different states according to the game.
ASCII ROM module will provide values for text generation for the main "PONG" display, the scores, the ball count, the game over screen etc.

a) Diagram



b) Modules:

Vga_controller:

This is a standard vga controller module that generates pixels for the vga display using horizontal and vertical sync signals and counters based on a 25 megahertz clock divider value.

Pong_text:

This generates all text for the game including the score, the game over and intro screens, the ball counter etc. It includes the ASCII character converted module.

M100_counter:

This counter is for incrementing the score in the range from 0 to 99 as well as to clear the score when a new game is started. It communicates with the graphics generation module to display score.

Ascii_rom:

This converts ASCII code into characters displayable on screen. This is part of the pong_text unit that generates all text used in the game, including the score, the game over and ball counter texts, and the main 'PONG' text.

Pong_graph:

This generates graphics for the game including the ball, paddle, and walls. It has a graphics still input and a graphics on input to differentiate between the beginning static screen and the dynamic screens when the game starts. It includes hit, miss and rgb outputs to determine the rgb values for the graphics and to determine the incrementation or decrementation of the score counter and ball counter. It has controls for how fast the paddle and ball moves and where they are in the starting position.

Timer:

It is a 2 second timer for the state machine when the game over state is entered. It takes in a timer start and a timer tick value from the state machine and outputs the timer up signal that lets the state machine know to move on to the next state.

Keyboard:

This module handles the input from the keyboard and checks which key was pressed. If "A" is pressed, then the paddle should move up, if "S" is pressed, then the paddle should move down. Pressing any other key would not affect the game.

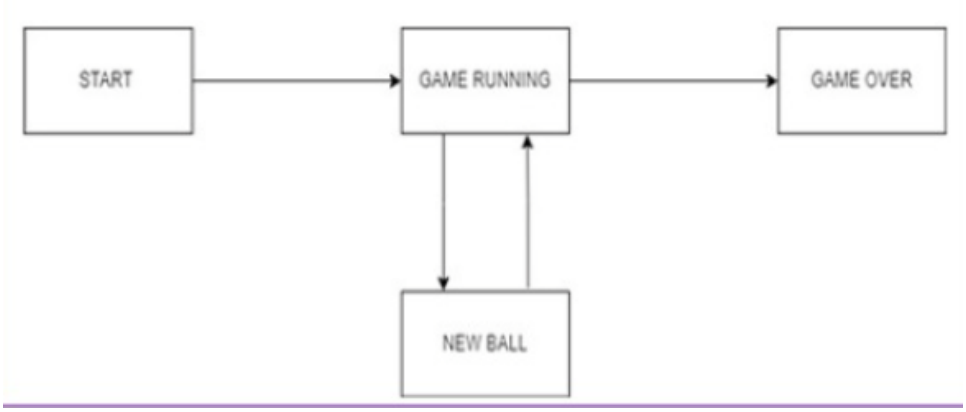
ps2_rx:

The ps2_rx module provided is a PS/2 keyboard interface receiver for the FPGA. The PS/2 protocol is commonly used for connecting keyboards and mice to a computer. This module is designed to interface with a PS/2 device, receive the serial data transmitted by the device, and convert it into an 8-bit parallel format that can be used by other parts of an FPGA design.

Control Block

The states of the game can be seen in the state diagram, with them also being represented on screens in the game.

a) State Diagram



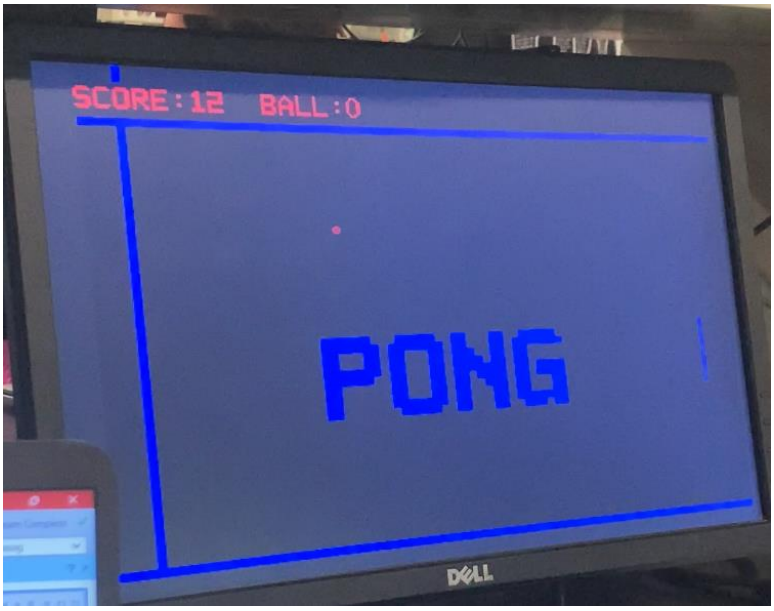
b) Screens

i) Start Screen

The is the initial screen that displays the rules of the game to the user.

ii) Game Running Screen

This is the screen in which the game will be running. There is a paddle on the left side of the screen and a ball that is bouncing around. The player must move the paddle around to avoid letting the ball go past it.



iii) Game Over Screen

When the player runs out of balls, this is the screen that is displayed. The game goes back to the start screen automatically after this, no user input needed.

FSM

Our game relies on keyboard input from the user to change the state of the game at the beginning, when transitioning from the Start state to the Running state. After that, however, the next state change occurs when the ball count is 0, automatically outputting the Game Over screen. This leads us to conclude that the FSM implemented in our game is a Moore Machine, as the output block does not depend on the input block.

Challenges

When modifying the code that we had sourced through GitHub, we faced a few issues.

i) When trying to modify the ball speed, if the x and y values for speed are different, they would cause issues with the score displayed on screen.
ii) Introducing multiple balls to the game to increase the difficulty proved to be quite challenging

References

- https://github.com/FPGADude/Digital-Design/tree/main/FPGA%20Projects/VGA%20Projects/Pong%20pt2

GitHub

- https://github.com/Ahtishamu/PONG-Basys-3-