DLD Final Project Report

Team Members

Syed Jahania Shah - ss06864
Muhammad Qasim Khan - mk05539
Muhmmad Najeeb Jilani - mj06879
Muhammad Ahmed Atif - ma06413

Project Title

Chrome T-Rex Game Using EMG

Team Title

De-MUX-ki-Tears

Evaluator : Dr Owais Talaat
Supervisor: Ahmed Bilal

18th December, 2021

Contents

Acknowledgement

We would like to thank our instructor, Dr. Owais Talaat, for assisting us in collecting data from the EMG module and guiding us through this project. We would also want to thank our wonderful RAs, Hassan Shah and Ahmad Bilal, who were always willing to help us and provide valuable insights into the study, as well as Miss Hafsa Amanullah, who was always willing to help us in the lab and provided necessary materials for this project.

Abstract

The Basys 3 is an entry-level FPGA development board that only works with the Vivado® Design Suite and uses the Xilinx® Artix®-7-FPGA architecture. Basys 3 is the latest addition to the renowned Basys line of FPGA development boards, which are widely used for a wide range of applications. The purpose of this project is to use the Verilog programming language to port Google Chrome's T-Rex mini-game to an FPGA Basys 3 board. The EMG waves are used as the game's input, which is amplified and denoised using the EMG module sensor. The input is then transformed to digital using the XADC code, which allows the dinosaur to be moved using 1-bit input. The goal is to be as similar to the original version's mechanics, aesthetics, and game rules as possible.

1 Introduction

The goal of this project is to use the Verilog programming language to port Google Chrome's T_Rex mini game on an FPGA Basys 3 board. The objective is to make the game mechanics, aesthetics, and game rules as close to the original version as feasible. Only one of the board's input buttons will be used to reset the game; however, the primary input will be collected from the EMG module through XADC and output to a 640x480 60FPS display via the VGA connector.

1.1 Specifications for the Design

1.1.1 Block Modules

• Buttons:
Reset: The score count was reset to zero and the initial state was restored.

• Background Image Processing: Choose from three pre-processed background photos, each of which has three states.
Design by FSM

• Trex Image Processing: Choose from four pre-processed Trex images, each of which has four states.
Default, Left, Right, and Dead are all terms that may be used to describe a situation.

• Image Processing of Obstacles: Small Cactus

• FSM was used to process the image of the score board.

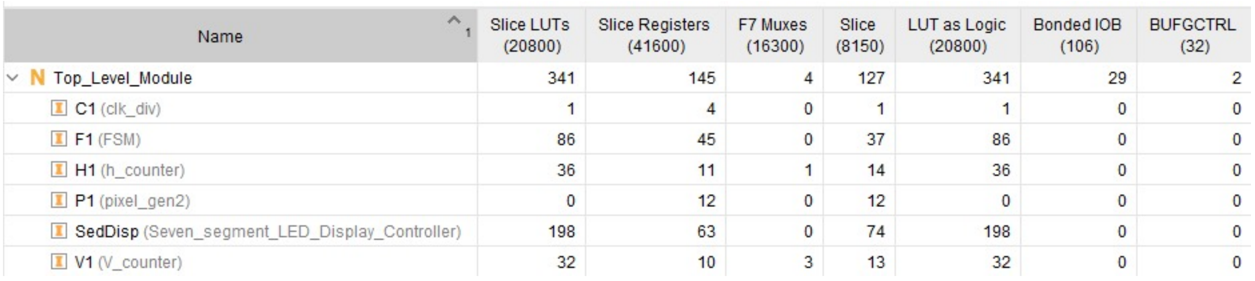
• Display with seven segments

• State Regulations:
FSM for a state transition

• Pixel Drawing:
VGA Module:
With correct synchronisation and porch border processing, settings for the 640x480 VGA display. Modules for Drawing:
Make a Background
Make a Dinosaur Drawing
Make a number
Make an Obstacle

• Module at the top level:
The top module links all of the sub modules and generates the desired display on the screen.

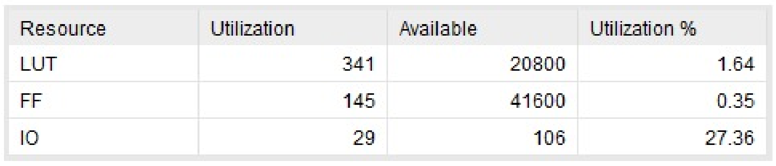
1.1.2 Resources Summary



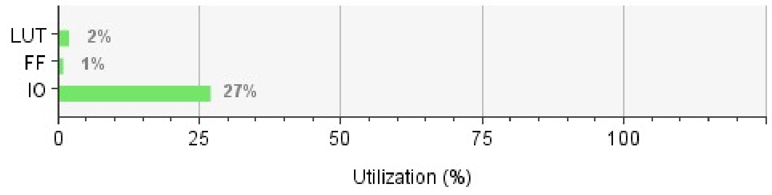
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
Top_Level_Module	341	145	4	127	341	29	2
C1 (clk_div)	1	4	0	1	1	0	0
F1 (FSM)	86	45	0	37	86	0	0
H1 (h_counter)	36	11	1	14	36	0	0
P1 (pixel_gen2)	0	12	0	12	0	0	0
SedDisp (Seven_segment_LED_Display_Controller)	198	63	0	74	198	0	0
V1 (V_counter)	32	10	3	13	32	0	0

Resources

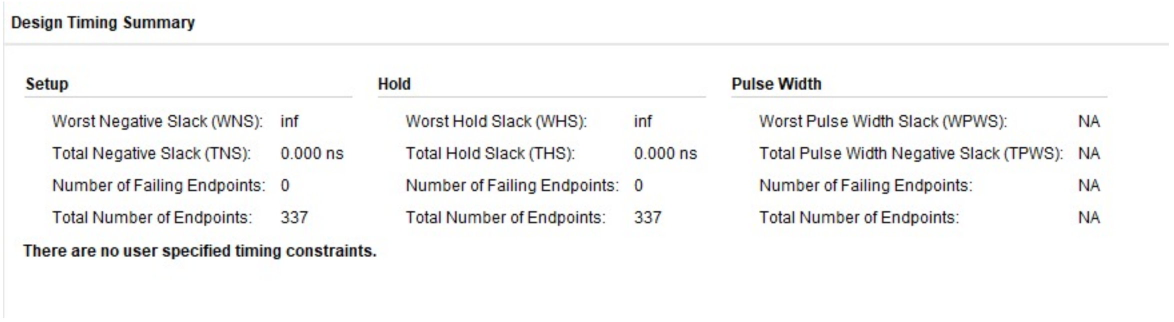
1.1.3 Resources Utilization Graph



Resource	Utilization	Available	Utilization %
LUT	341	20800	1.64
FF	145	41600	0.35
IO	29	106	27.36



1.1.4 Design Timing Summary



Design Timing Summary

Setup

Worst Negative Slack (WNS):	inf
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	337

Hold

Worst Hold Slack (WHS):	inf
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	337

Pulse Width

Worst Pulse Width Slack (WPWS):	NA
Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA

There are no user specified timing constraints.

1.2 EMG Module

EMG or Electromyography is a technique used to measure muscle response or electrical activity in response to nerve stimulation of the muscle. The module being used is EMG Muscle Sensor Module V3.0. This module will measure filtered and rectified electrical activity of a muscle. It will output 0-Vs volts depending on the amount of activity in targeted muscle. Here Vs indicates the voltage of the power source. The minimum power supplied is ±3.5V.

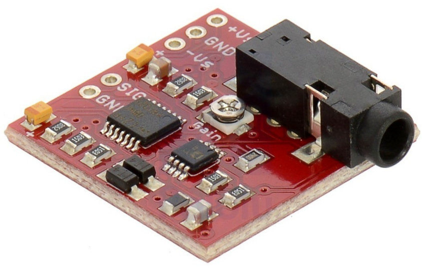


Figure 1: EMG Module

Traditionally EMG is used in medical practice and research but with micro-controllers and integrated circuits getting smaller and smaller and more powerful,we are now able to use the EMG module for various control systems.

This module measures the electrical signal generated from muscle stimulation. Naturally there will be added noise and unwanted data in this signal that needs to be removed. The signal is rectified by the EMG module (converted from AC to DC) and unwanted data/noise is filtered from it. It then produced an analogue signal that can be read by a micro-controller and used to control interfaces in your project.

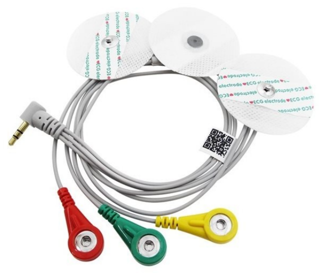


Figure 2: EMG Module Electrodes

The module we are using comes with and EMG cable as well as electrodes. Three electrodes are needed for operation. How to place them and wire them to the module is explained here:

- The reference electrode should be placed on an inactive section of the body, such as the bony portion of the elbow, shin or forearm. This electrode should be connected to the yellow cable.
- The two other electrodes should be placed along the muscle targeted to be measured. The second electrode should be placed along the mid-length of the muscle; this electrode should be connected to the red cable.
- The last electrode should be placed at the end of the muscle and connected to the green cable.
- Finally, connect pin SIG to an analogue input pin of your micro-controller and the GND pin to the ground pin on your micro-controller.

1.3 Basys-3 FPGA Board

1.3.1 What is a FPGA?

A FPGA short for Field Programmable Gate Array are semi-conductor devices based around Configurable Logic Blocks (CLBs) which are connected via interconnects that are programmable. As the name suggests they can be programmed to perform different functions/applications after they are manufactured. FPGAs have evolved continuously in order to meet market demands and as such can be used for a plethora of applications:

- Aerospace and Defense
- ASIC Prototyping
- Video and Image Processing
- Medical
- Automotive
- Consumer Electronics
- High Performance Computing and Data Storage
- Wired/Wireless communications

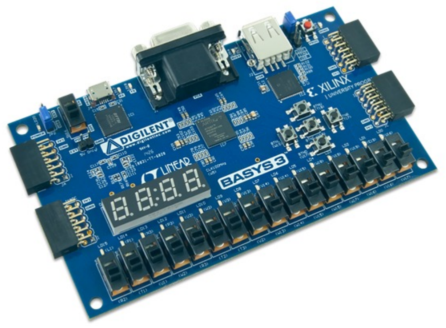


Figure 3: Basys-3 FPGA

1.3.2 Basys-3 FPGA

The Basys-3 is one of the best boards on the market for getting started with FPGA. It is an entry-level development board built around a Xilinx Artix-7 FPGA.

As a complete and ready-to use digital circuit development platform, it includes enough switches, LEDs, and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware. There are also enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits, and all of this at a student-friendly price point.

1.4 Chrome Dino Game

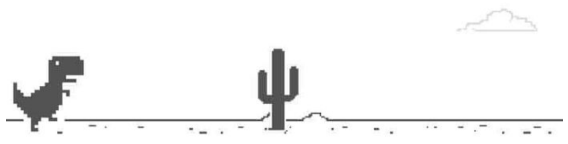


Figure 4: Chrome Dino Game

The game that we are aiming to simulate is the popular Chrome Dino game that appears when Google Chrome is offline. It appears when a user tries to navigate a web page while Chrome is offline. The browser displays a pixelated T-Rex that is ”lonely”. If on a PC/Laptop, the user can get the game running by pressing the space-bar or the up arrow button. The background is a side-scrolling landscape upon which the T-Rex runs. The user (if on keyboard), can get the T-Rex to jump or duck to avoid obstacles. The obstacles are of two types:

- Pterodactyls (the flying ”birds”)
- Cacti (they come in different heights, combinations, and sizes)

The user can get the T-Rex to jump over the cacti by pressing any one of the buttons that are used to launch the game. The T-Rex can either jump above or duck below pterodactyls depending on how they are coming. The ”duck” move comes by pressing the down arrow button. A score counter is also kept and highest score (of the user) is saved. The graphics also change from day to night depending on certain score boundaries crossed. If the T-Rex collides into a cactus or pterodactyl, the game ends with a game over message displaying and then just after it a replay symbol with a message ”REPLAY?”. It’s a single player game and was designed by Sesbastien Gabriel.

1.5 The game on FPGA

We decided that we would simulate this game on FPGA through coding it in Verilog and controlling the T-Rex with our hand by the EMG module. We identified the following conditions in the game that exist:

- Idle - everything is at ”default” position. The T-Rex is standing, the score counter hasn’t begun yet, obstacles aren’t arriving yet, the background is not moving
- Run - the user presses a space-bar or up arrow button and the game begins. The T-Rex starts moving, the obstacles start appearing, the background is moving, and the score increments is recorded (if it is the new high score).
- Jump - the user presses the space-bar or up arrow button to get the T-Rex to dodge the obstacles. The T-Rex jumps and after reaching a certain height, it returns back to the ground. This i vertical movement only.
- Dead - this is the state when the T-Rex collides with any obstacle. The game then stops completely with the T-Rex also stopping. The score doesn’t get incremented anymore and if it’s passed the previous high score, it is now the new high score. A replay button pops up on the screen.

1.6 Adjustments to the original Game

Considering that we are wanting to implement this game on Basys-3 FPGA and it will be programmed in Verilog, we decided to reduce some features from the original game:

- We will be counting the score, but we will not be recording for higher records.
- We will not include pterodactyls as an obstacle
- The cactus will be present as an obstacle, but a single cactus and not cacti (i.e. two or three)
- The cactus will be of the same height and will not change as it happens in the original
- The T-Rex jumps at a certain height and for a certain length, depending on how hard the button is pressed. We won’t be incorporating this in our game
- The speed of the game increases as you play even further. What it means is that the obstacles come at a faster speeds. This will not be a feature in our game
- The game doesn’t usually have a ”Start” dialogue screen and ”You Lost” screen. We will be incorporating these features in our game

2 Methodology

2.1 Flow Chart

2.1.1 Input Block

The EMG module is capable enough to generate muscle waves with great denoising power, the electrodes are placed on the transition of hand, where it can easily measure the difference between two waves however, the ground is connected to the movement of hand, therefore at every movement, the waves will move towards the peak position, the analog waves are then converted into digital waves by using XADC module generator in FPGA, the code is easily available on the Diligent Website, however the configuration was bit difficult part but it has been configured successfully. The digital output will be between 0 and 1, the rest condition occurs between 0 to 0.5 hence we can easily determine the position of hand by using the seven-segment voltage, for the voltage equal to or greater than 0.5 we will assume the state the movement, hence we can make two or three states very easily. There is one advantage or it may be disadvantaging that due to the ground if anyone touches on the hand of the player, the voltages get to its peak 1V, it is due to the ground but it can be beneficial in a sense when we want to help the player to ease the game once it gets faster.

Input Block will manage the input stream from the keyboard and sensor and translate it to store the currently pressed keys along with detecting unpressed state which is ‘000’. For the first case when we tested the project we also integrated the keyboard with the game to avoid any inconvenience.

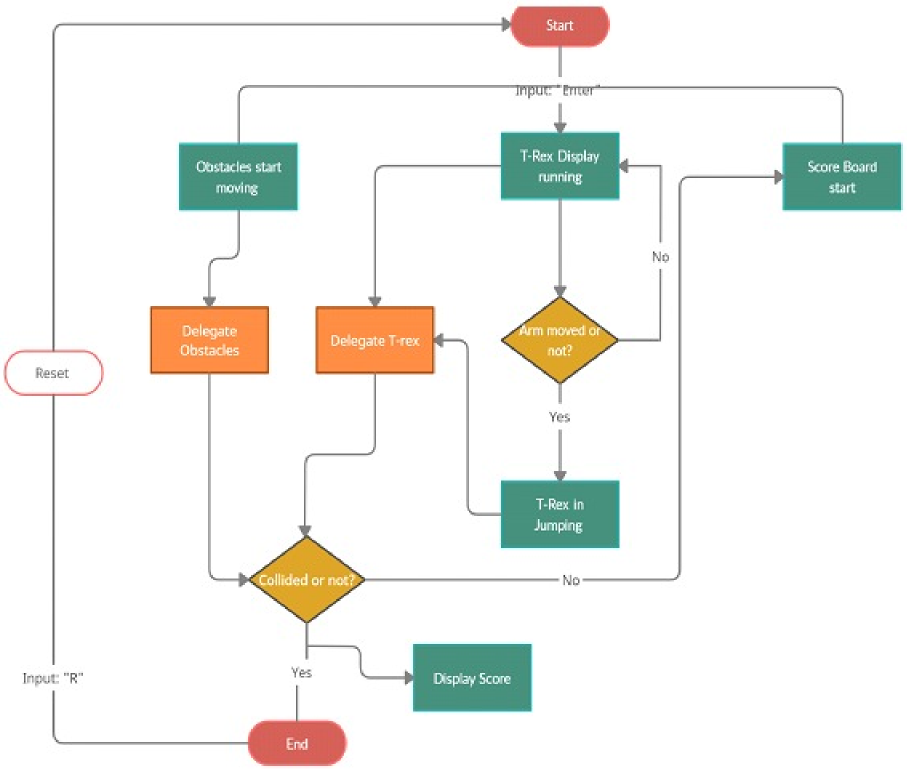


Figure 5: Flow Chart for Game

2.2 FSMs in our Game

Our game has many different things running in order to create the best playing experience. These things can be modeled by FSM (Finite State Machine). They will tell what states the game is in and what inputs are causing it to move to the next state as well as the output. The first FSM we'll be modelling here is for the score counter.

Gate Level FSM

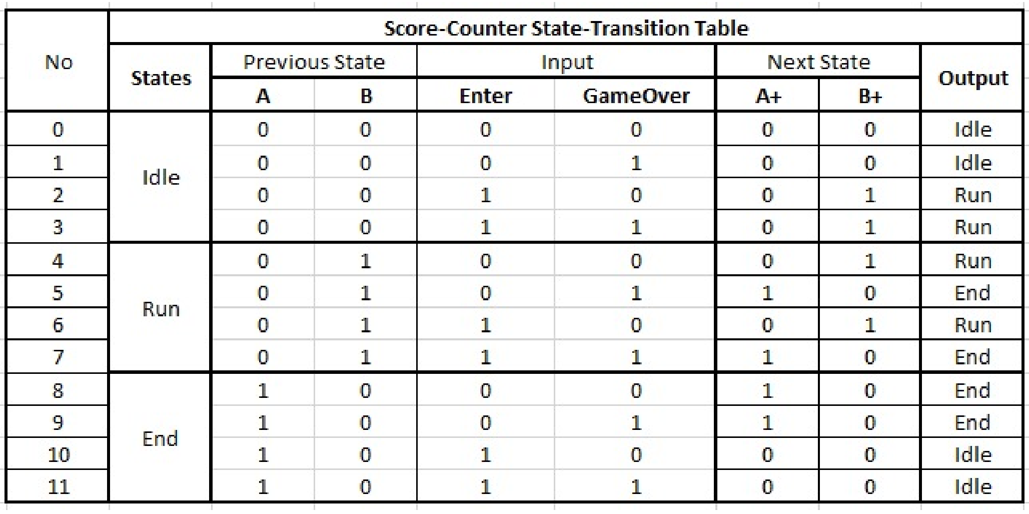
2.2.1 Score Counter Module

The FSM being used here is Gate Level Implementation of the FSM diagram.
The states are:

- Idle - T-Rex is not running, score is not counting
- Run - The T-Rex is running, the score is also being counted
- End - The T-Rex has collided and is stationary and score has stopped counting

Since there are two bits being used for each state, we have $2^2 = 4$ combinations of bits.
Accordingly 2 flip-flops are adequate to construct this.

State Transition Table



No	Score-Counter State-Transition Table								
	States	Previous State		Input		Next State		Output
		A	B	Enter	GameOver	A+	B+	
0	Idle	0	0	0	0	0	0	Idle
1		0	0	0	1	0	0	Idle
2		0	0	1	0	0	1	Run
3		0	0	1	1	0	1	Run
4	Run	0	1	0	0	0	1	Run
5		0	1	0	1	1	0	End
6		0	1	1	0	0	1	Run
7		0	1	1	1	1	0	End
8	End	1	0	0	0	1	0	End
9		1	0	0	1	1	0	End
10		1	0	1	0	0	0	Idle
11		1	0	1	1	0	0	Idle

Figure 6: Score Counter State Table

State Transition Diagram

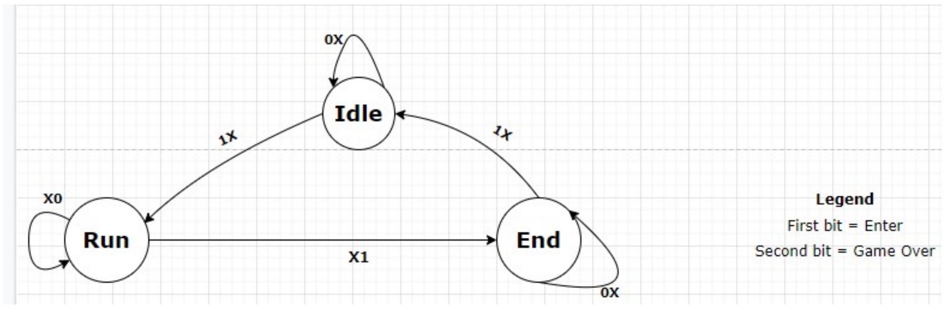
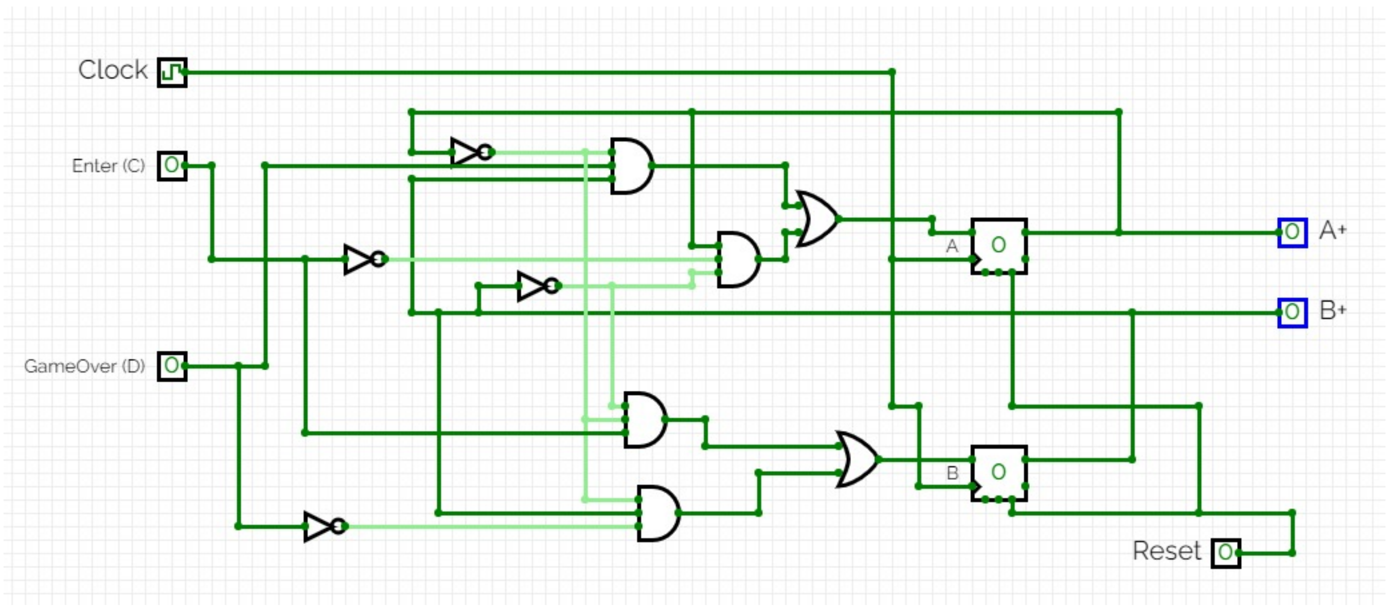


Figure 7: Score Counter State Diagram

State Equations

$A(t+1) = A(t)'B(t)D + A(t)B(t)'C'$
$B(t+1) = A(t)'B(t)'C + A(t)'B(t)D'$

Gate Level Implementation



Gate Level Implementation

Verilog Implementation

```
module DflipFlop(q, q_inv, clk, d, a_rst, pre, en);
    parameter WIDTH = 1;
    output reg [WIDTH-1:0] q, q_inv;
    input clk, a_rst, pre, en;
    input [WIDTH-1:0] d;

    always @ (posedge clk or posedge a_rst)
    if (a_rst) begin
        q <= 'b0;
        q_inv <= 'b1;
    end else if (en == 0) ;
    else begin
        q <= d;
        q_inv <= ~d;
    end
endmodule
module ScoreCounter(out_A, out_B, clk_i, enter_C, gameover_D, reset);
  output out_A,  out_B;
  input enter_C, gameover_D, reset, clk_i;
  wire B_Q, and_3_out, or_1_out, and_0_out, or_0_out, A_Q, and_1_out, not_0_out, and_2_out, not_2_out, n
  DflipFlop B(B_Q, , clk_i, or_1_out, reset, , );
  assign out_B = B_Q;
  assign and_3_out = not_0_out & B_Q & not_3_out;
  assign or_1_out = and_2_out | and_3_out;
  assign and_0_out = not_0_out & gameover_D & B_Q;
  assign or_0_out = and_0_out | and_1_out;
  DflipFlop A(A_Q, , clk_i, or_0_out, reset, , );
  assign out_A = A_Q;
  assign and_1_out = A_Q & not_1_out & not_2_out;
  assign not_0_out = ~A_Q;
  assign and_2_out = not_2_out & not_0_out & enter_C;
  assign not_2_out = ~B_Q;
  assign not_3_out = ~gameover_D;
  assign not_1_out = ~enter_C;
endmodule
```

2.2.2 Main Module

2.3 Behavior Finite State Machine

There are total 5 states in this module:

- Idle
- Run
- Move Up
- Move Down
- Dead

As we are using three bits to represent each state. So $2^3 = 8$ bit combinations. As there are 5 states so, the number of bits is enough to accomodate them.

State Transtition Diagram

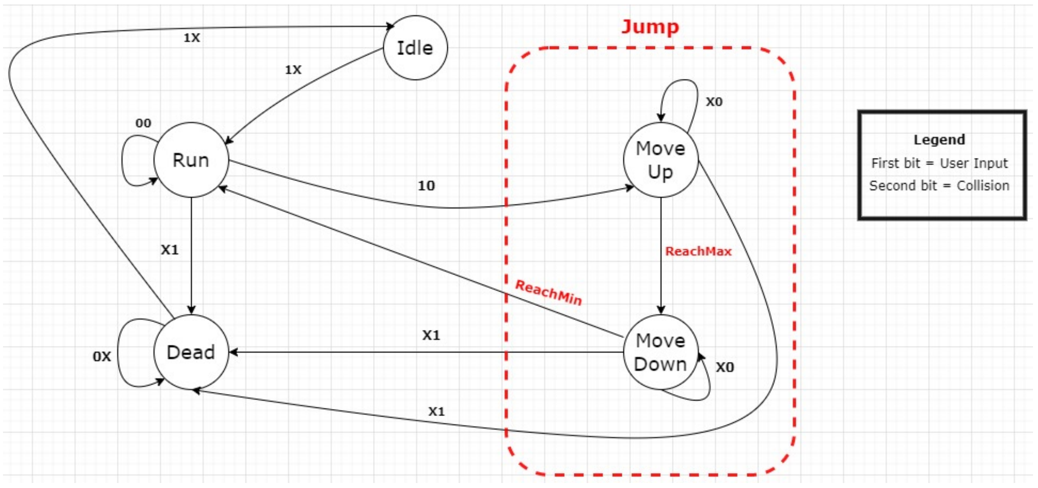
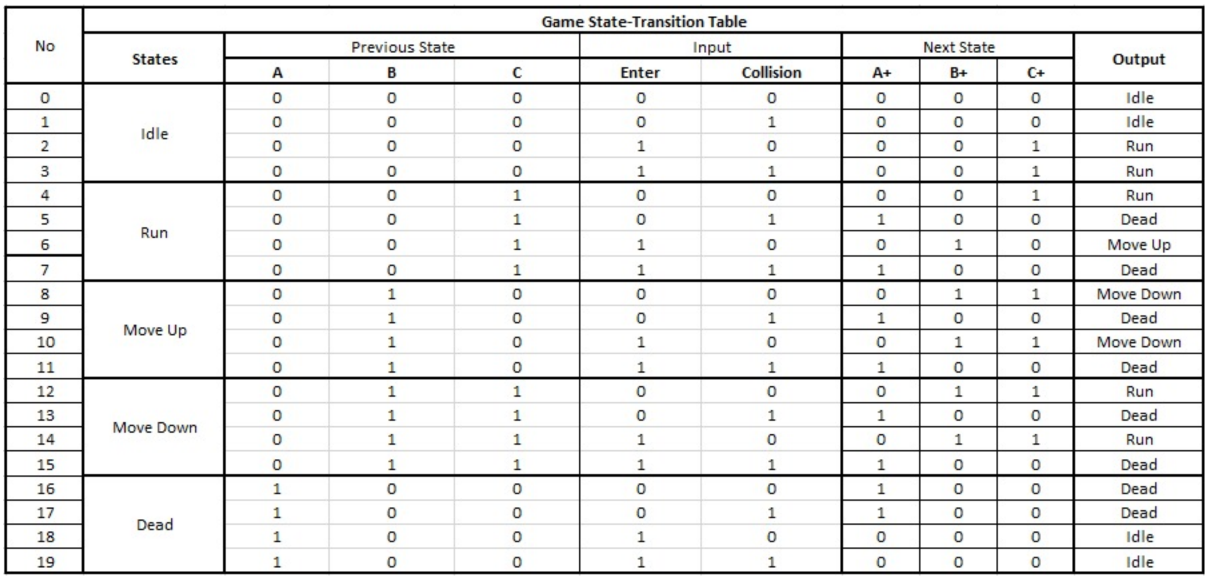


Figure 8: Main FSM Diagram

State Transtition Table



No	Game State-Transition Table									
	States	Previous State			Input		Next State			Output
		A	B	C	Enter	Collision	A+	B+	C+	
0	Idle	0	0	0	0	0	0	0	0	Idle
1		0	0	0	0	1	0	0	0	Idle
2		0	0	0	1	0	0	0	1	Run
3		0	0	0	1	1	0	0	1	Run
4	Run	0	0	1	0	0	0	0	1	Run
5		0	0	1	0	1	1	0	0	Dead
6		0	0	1	1	0	0	1	0	Move Up
7		0	0	1	1	1	1	0	0	Dead
8	Move Up	0	1	0	0	0	0	1	1	Move Down
9		0	1	0	0	1	1	0	0	Dead
10		0	1	0	1	0	0	1	1	Move Down
11		0	1	0	1	1	1	0	0	Dead
12	Move Down	0	1	1	0	0	0	1	1	Run
13		0	1	1	0	1	1	0	0	Dead
14		0	1	1	1	0	0	1	1	Run
15		0	1	1	1	1	1	0	0	Dead
16	Dead	1	0	0	0	0	1	0	0	Dead
17		1	0	0	0	1	1	0	0	Dead
18		1	0	0	1	0	0	0	0	Idle
19		1	0	0	1	1	0	0	0	Idle

Figure 9: Main FSM Table

Verilog Implementation

```
module FSM(clk_div1,fsm_reset,fsm_input_user,state,mover_dino,mover);
input fsm_input_user;
input clk_div1;
input fsm_reset;
//////////////////////////////////
output reg[2:0]state = 3'b000;
output reg [9:0]mover = 639;
output reg[9:0]mover_dino = 380;

//////////////////////////////////
reg flag = 1;


reg fsm_coll;
reg[2:0] next;
 reg [17:0]counter_fsm = 0;
/////////////////////////////////
  parameter Idle = 3'b000;
  parameter Run = 3'b001;
  parameter MoveUp = 3'b010;
  parameter MoveDown = 3'b011;
  parameter Dead = 3'b100;

  assign ReachMax = (mover_dino<=150)? 1'b1:1'b0;
  assign ReachMin = (mover_dino>=380)? 1'b1:1'b0;



always @(posedge clk_div1 or posedge  fsm_reset or posedge fsm_input_user)//or posedge video_on)
begin
if (counter_fsm < 195619)
```

```
            counter_fsm <= counter_fsm + 1;
       else
       begin
                counter_fsm=0;

   if(fsm_reset==1)
   begin
      state=Idle;

    end
   else
   begin


       if(((mover_dino+50)<438 && (mover_dino+50)>348)&& (mover<100 && (mover+25)>48))
            fsm_coll <= 1;
        else
            fsm_coll <= 0;
        state = next;
      case(state)
        Idle:
        begin
        mover_dino<=380;
        mover <= 600;

          if(fsm_input_user==1'b1)
          begin
            next <= Run;
            flag<=1;
            end
          else if(fsm_input_user == 1'b0)
            next <= Idle;
            end
        ////////////////////////
      Run:
      begin
      mover_dino <= 380;

          if(mover<=0)
                    mover <= 639;
                 else
                   mover <= mover - 2;
          if(fsm_coll== 1'b1)
           next <= Dead;
          else if (fsm_input_user== 1'b0)
                 begin
                 flag<=0;
                next <= Run;
                end
          else if(fsm_input_user == 1'b1)
             next <= MoveUp;
        end
        /////////////////////////
        MoveUp:
          begin
          mover_dino <= mover_dino-3;
          if(mover<=0)
                    mover <= 639;
                 else
                   mover <= mover - 2;

          if(fsm_coll== 1'b1)
                next <= Dead;
          else if(ReachMax==1'b0)
                next <= MoveUp;
          else if(ReachMax==1'b1)
                next <= MoveDown;
          end
        ///////////////////////////
        MoveDown:
        begin
        mover_dino <= mover_dino+3;


        if(mover<=0)
                    mover <= 639;
                 else
                   mover <= mover - 2;
          if(fsm_coll== 1'b1)
                next <= Dead;
          else if(ReachMin== 1'b0)
                next <= MoveDown;
          else if(ReachMin==1'b1)
                next <= Run;
          end
          /////////////////////////
        Dead:
        begin
          if(fsm_input_user == 1'b0)
                    next<=Dead;
          else if(fsm_input_user==1'b1)
                    next <= Idle;
          fsm_coll<=0;
        end
        default:
          next = Idle;
        endcase
          end
        end
      end
```

endmodule

3 Schematic

3.0.1 Clock Divider

Inputs: clk: The main FPGA hardware clock (which operates at 100MHz).
Ouput: clk-d; A slower clock (which operates at 25MHz).
Module Description: This module divides the clock time period into half until the clock slows down to 25MHz.

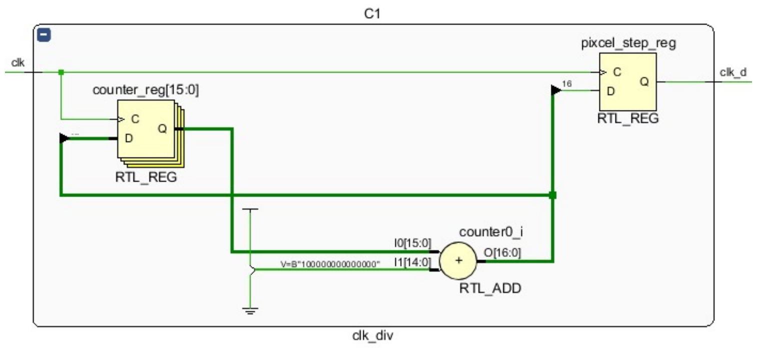


Figure 10: Clock Divider Schematic

3.0.2 H-Counter

Input; clock Uses 25MHz from module clk-div as main clock.
Output; h-count It is a counter which counts till 799 digits (used for indexing columns in the used VGA based Screen).
trig-V; It is a flag which flag to 1 when h-count resets to 0.
Module Description; This module act as a counter for counting columns (horizontally) of an VGA based display. When the ‘h-count’ reaches 799 (which may be used to denotes the left-most end of the display horizontally), the ‘trig-V” flags and ‘h-count’ is reset (which may be used to denote the right-most end of the display).

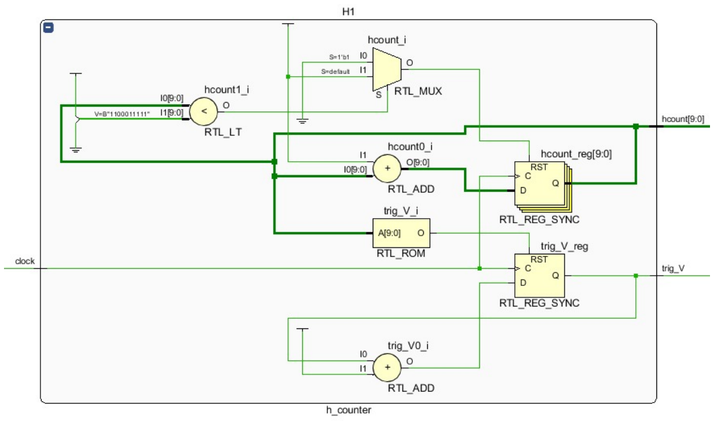


Figure 11: h-counter Schematic

3.0.3 V-counter

Input : clock Uses 25MHz from module ‘clk-div’ as main clock.
V_signal: Uses ‘trig-v’ from ‘h-counter’ module as reset for ‘v-count’ counter.

Output : V_count: It is a counter which counts till 524 digits (used for indexing rows in the used VGA based Screen).

Module Description: This module act as a counter for counting rows (vertically) of an VGA based display. When the ‘V-count’ increments (which may be used to denotes moving onto next row), when ‘V-signal” flags. Simply the ‘V-counts’ waits for its increment until module ‘h-counter’ has counted through all the columns of a particular row on an VGA based display.

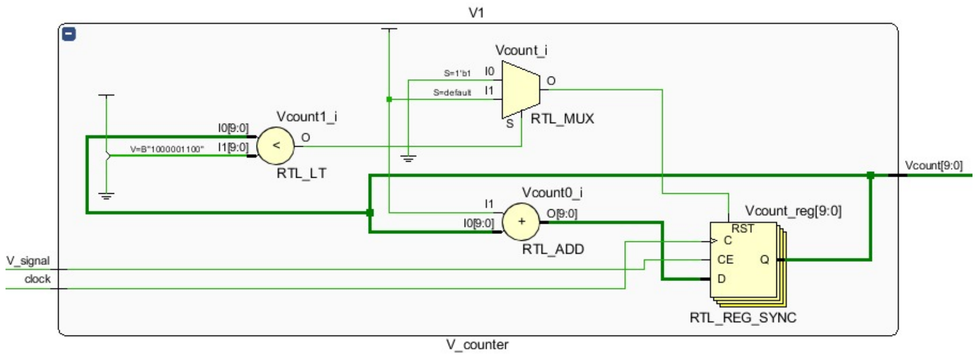


Figure 12: V-counter Schematic

3.0.4 VGA Sync

Input : hcount : The counter for columns in the VGA based screen.
vcount : The counter for rows in the VGA based screen.

Output : h_sync : During the horizontal scan, it scans from left to right and then return to the right .A period of the h_sync signal contains 800 pixels and can be divided into two regions:

Display: Region where the pixels are actually displayed on the screen. The length of this region is 640 pixels.

Blanking: region has 3 segments:

Retrace: Region in which the electron beams return to the left edge. The video signal should be disabled (i.e., black), and the length of this region is 96 pixels.

Right border: Region that forms the right border of the display region. It is also known as the front porch (i.e., porch before retrace). The video signal should be disabled, and the length of this region is 16 pixels.

Left border: Region that forms the left border of the display region. It is also known as the back porch (i.e., porch after retrace). The video signal should be disabled, and the length of this region is 48 pixels.

V-sync : During the vertical scan, it scans from top to bottom and then return to the top. This corresponds to the time required to refresh the entire screen. A period of the v_sync signal is 525 lines and can be divided into two regions:

Display: Region where the horizontal lines are actually displayed on the screen. The length of this region is 480 lines.

Blanking: region has 3 segments:

Retrace: Region that the electron beams return to the top of the screen. The video signal should be disabled, and the length of this region is 2 lines.

Bottom border: Region that forms the bottom border of the display region. It is also known as the front porch (i.e., porch before retrace). The video signal should be disabled, and the length of this region is 10 lines.

Top border: region that forms the top border of the display region. It is also known as the back porch (i.e., porch after retrace). The video signal should be disabled, and the length of this region is 33 lines.

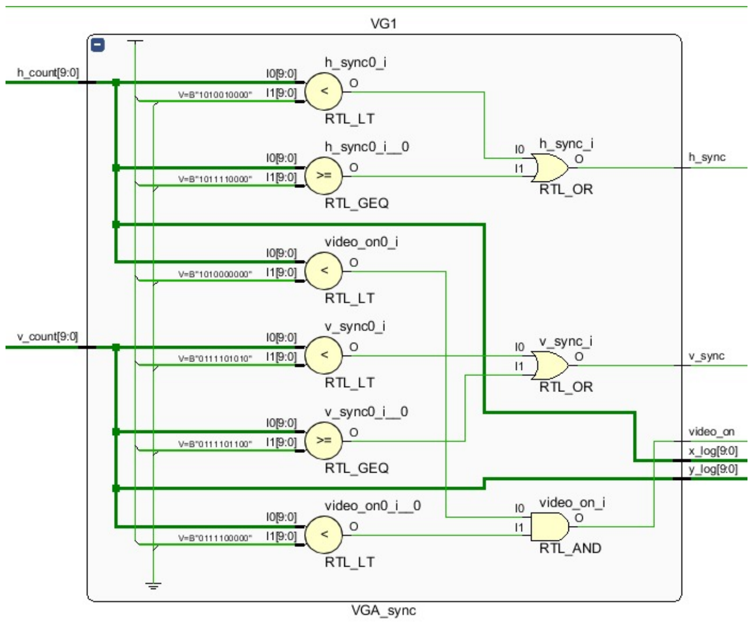


Figure 13: VGA Sync Schematic

Video_on : The video_on signal indicate whether the current targeted pixel is in the displayable region. It is asserted only when the h_count smaller than 640 and v_count is smaller than 480.

X_log : It represent the location of pixel x. The location of the current pixel and can be obtained from h_count.

Y_log : It represent the location of pixel y. The location of the current pixel and can be obtained from v_count.

Module Description:
This module essentially is used to synchronise the horizontal scanning with vertical scanning. It defines parameters such as display pixel width, height, porch areas, blanking areas and retrace area on the VGA based display.

3.0.5 Pixel Generator

Input :

pixel_x ; The location of pixels horizontally.

pixel_y ; The location of pixels vertically.

clk_div1 ; The main Clock used in this module.

video_on : The video_on signal indicate whether the current targeted pixel is in the displayable region. It is asserted only when the h_count smaller than 640 and v_count is smaller than 480

state : The current State of Game FSM, which is used to generate different set of output signals.

mover_dino : It is used to move the Dinosaur object on the screen (changes only vertical location of Dinosaur).

mover : It is used to move the obstacle object on the screen (changes only horizontal location of obstacle).

Output :

Red : The Red Data channel for VGA based display.

Blue : The Red Data channel for VGA based display.

green : The Red Data channel for VGA based display.

Module Description:
This module primary function is to output Red, Green, Blue (RBG) signals that will be used by VGA input based display screen . The position where RGB signals will be displayed is controlled by 'state' input to display elements base on current Game FSM state.

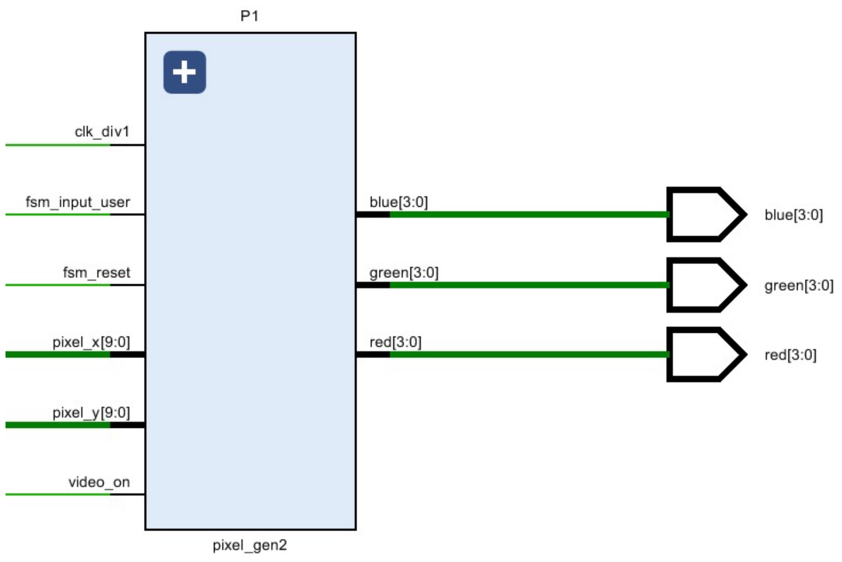


Figure 14: Pixel Generator Schematic

3.1 Top Level Schematic

Here is the final schematic diagram, which contains some other modules as well, which have not been discussed above because of less priority.

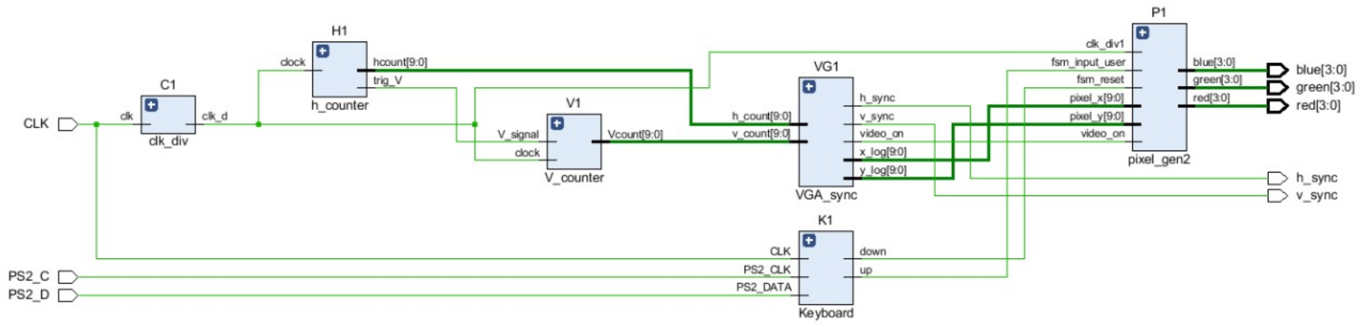


Figure 15: Top Level Schematic

4 Troubleshooting

- Inferred Latch: Missing Default Case
  -Remedy: By adding the default case, VGA display the game properly

- Overflow: Many time the over flow messed up the result
  -Remedy: setting proper size with global variable by doing math and check it constantly

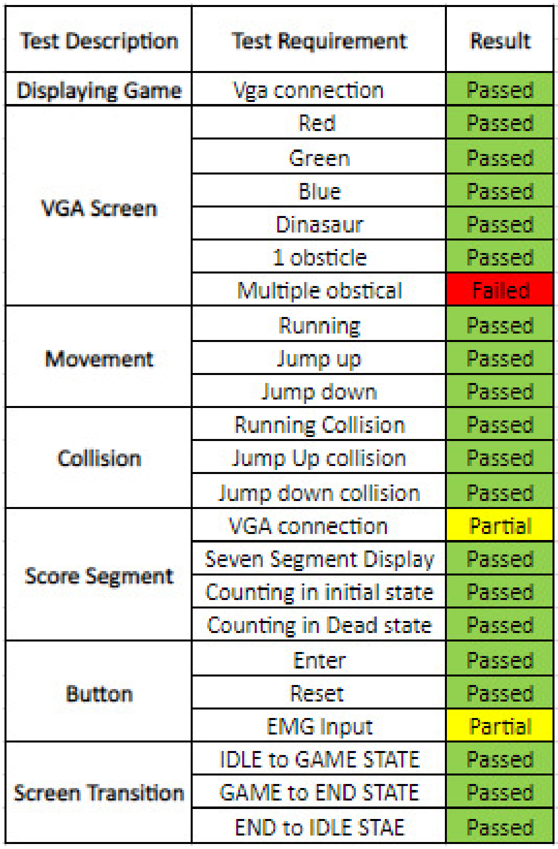
- Negative Sign: Our jump control and background got into great issues when negative number was applied in the equation.
  -Remedy: Use Signed wire and reg

- Blocking/non-block assignment: one of the bug that we can't catch instantly is misused of Blocking assignment where it should actually be non-Blocking. This bug cause invisible cactus and corruption in background display
  -Remedy: change to non-Blocking assignment.

- On one of Main Game FSM state, when 'enter' is pressed it skips over one state due to reading input as level trigger.
  -Remedy: Taking 'enter' input as edge trigger.

5 Testing And Evaluation

Following cases have been measured to test our final evaluation.



Test Description	Test Requirement	Result
Displaying Game	Vga connection	Passed
VGA Screen	Red	Passed
	Green	Passed
	Blue	Passed
	Dinasaur	Passed
	1 obsticle	Passed
	Multiple obstical	Failed
Movement	Running	Passed
	Jump up	Passed
	Jump down	Passed
Collision	Running Collision	Passed
	Jump Up collision	Passed
	Jump down collision	Passed
Score Segment	VGA connection	Partial
	Seven Segment Display	Passed
	Counting in initial state	Passed
	Counting in Dead state	Passed
Button	Enter	Passed
	Reset	Passed
	EMG Input	Partial
Screen Transition	IDLE to GAME STATE	Passed
	GAME to END STATE	Passed
	END to IDLE STAE	Passed

Figure 16: Test Cases

6 Team Work Distribution

There were total four main important tasks which had been divided among the four members of the group; Following were the tasks;

FSM DESIGNING

The documentation for this task was majorly done by **Muhammad Najeeb Jilani**, but the modification of the implemented design in verilog has been done collectively.

GRAPHICS

Graphics was the main part of the game after FSM, it includes all the designing of objects; Dinasaur, Obsticles, and Background, this part was majorly done by **Muhammad Ahmed Atif**, however the effort has been made collectively.

SCORE SEGMENT

Score segment is another important part of this project, it measures the success of the player's performance, this part has been done majorly by **Muhammad Qasim Khan**, however the effort has been made collectively.

EMG Input and Delegates

This part was also important and interesting, however delegates is the main important function of this project, this task was majorly done by **Jahania Shah**, however **Ahmed Atif** also played an important role on it.

7 Discussion And Conclusion

The Verilog VGA project with FPGA board was inspired by Chrome's existing T_rex game. The intended purpose of this project was to create a reproduction of the original game, however owing to "physical capabilities limitations and time constraints," this was not possible. Our modest wish seemed to be too much to ask. The project has suffered a setback as a result of making a heartbreaking decision has decreased the number of types of obstacles from three to one, and we must also get rid of the clouds and the fog birds that have the potential to make the game more intriguing and challenging.

The project can be categorized in four main parts;

Delegates Each Object has its unique position and handles its position according to the FSM designed.

Graphic Displaying the module with colorful objects.

FSM there are two FSM s which control the game; first is state transition FSM, which is designed on behavior model and second is score count FSM, which is designed on gate level implementation.

Clock Multi-functional clock that can handle request for a desire pulses per second.

EMG Input This part is very interesting as it takes input from the muscles waves when we bend our hand or apply some force on it, the output is then converted from analog waves to digital waves through XADC algorithm, which is then used as user input of 1 bit, which controls the movement of T_rex. However for uncertainty of our model, we have also designed keyboard module in which we can alternatively use the keyboard key to get user input.

8 Future Work

This project necessitates some time commitment; nonetheless, it has the potential to be highly artistic and engaging, and we want to work on it throughout our winter vacation in order to display it in an exhibition.

Following are the goals to be implemented.

Designing Clouds

Designing Fog birds

Designing Multiple Obstacles

EMG modulation

The motive for the EMG module cannot be overlooked, as we began this research with the goal of controlling items through the deduction of hand movement. We discovered out after a lot of effort that this could be done using EMG waves, and it offered us a means to work on T rex where we could control the dinosaur's movement with the movement of our hands.

We aim to work on an EMG module in the future to control the movement of one's hand with the movement of another person's hand, as has been done at many colleges, and this is our goal.

"The Future belongs to those who believe in the beauty of their dreams" Eleanor Roosevelt