HABIB UNIVERSITY

Dhanani School of Science and Engineering

DIGITAL LOGIC AND DESIGN

(EE-172L/ CS-130L)

PROJECT REPORT
INFINITY RUN

DESIGNED BY:

ABBAS HAIDER TAQVI

DANIA SALMAN

HANIYA KHAN

SAFI HAIDER

GAME FEATURES

Figure 1.1

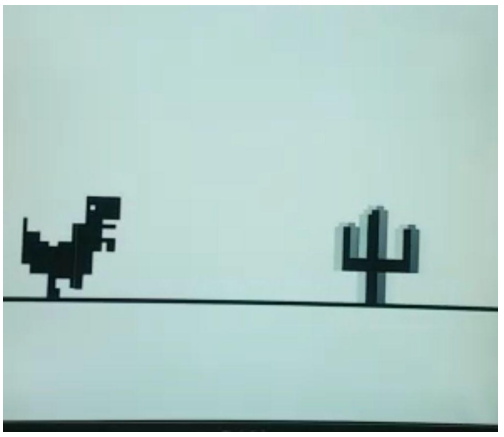


Figure 1.3

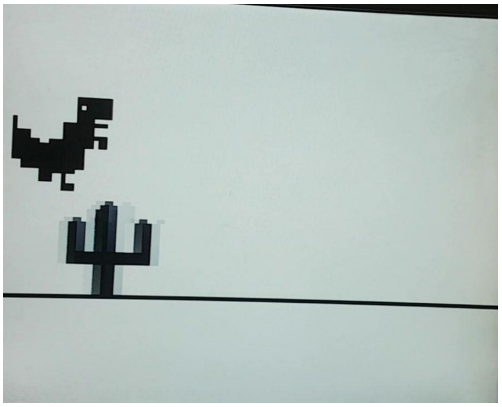


Figure 1.2

Figure 1.4

ABOUT THE GAME:

The game presented is a single player implementation of the Chrome extension T- rex Runner. This game is implemented by taking input from the two sensors, one for jump and one for duck and the distance of the hand movement of the user decides the high or low input, where high represents jump for jump sensor and low represents duck for duck sensor which are both placed opposite to each other. When the distance between sensor and hand is negligible the output is high.

Thus, the running dinosaur can either jump or duck to avoid bumping into obstacles and to stay alive. A game over screen will be displayed at the end of the game if you fail to avoid the obstacle. The game goes back to running if you press the reset button and to the main screen if you turn off the start switch.

BLOCK DIAGRAM OF DINO GAME

The game can be implemented using the 3 main blocks (input, control unit and output). The input block contains the input received from the IR sensor which goes to the control block that gets executed using the 3 FSM's (game FSM, Dino FSM, obstacle FSM) resulting in the Dino to jump, duck and collide. Finally the game output is able to show the implementation of this through VGA signals to monitor.

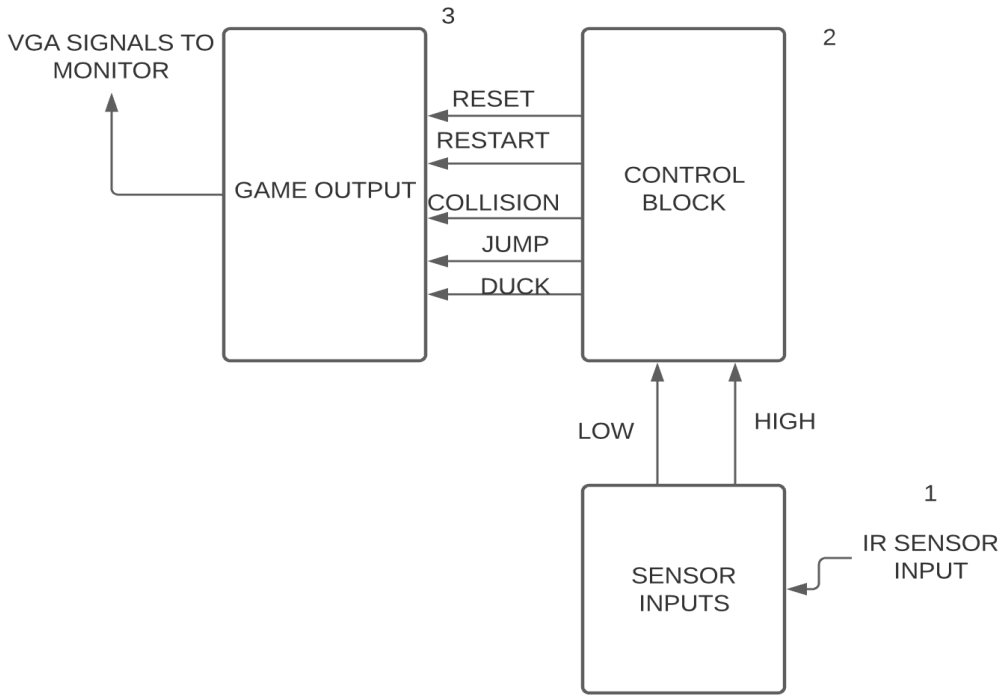


Figure 1.5

The control block will include 3 different FSM's.

1) GAME FSM
2) DINO FSM
3) OBSTACLE FSM

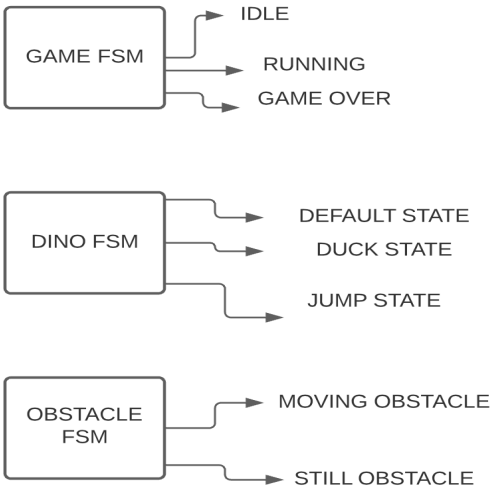


Figure 1.6

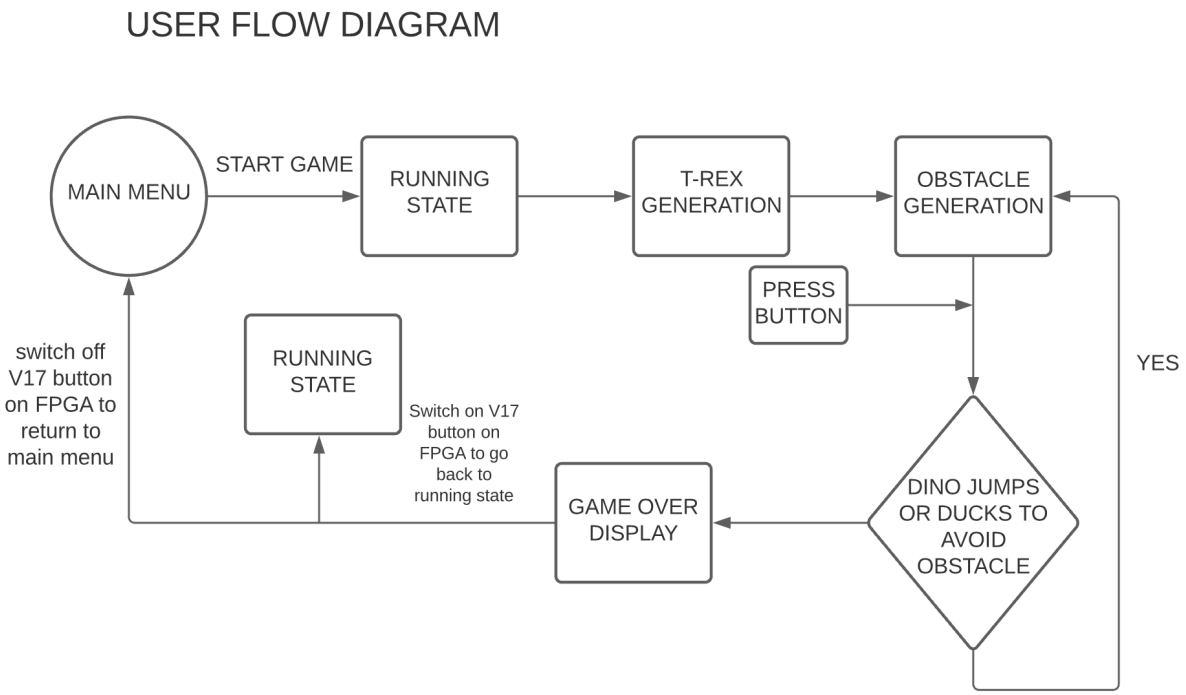


Figure 1.7

The user flow diagram above shows the user experience from the perspective of the user. In the diagram above, the user starts with the main menu which displays the start game. Instantly, the user will be able to see a dinosaur on the screen and as it is moving forward an obstacle will be generated. Using the sensor the Dino will jump or duck to overcome obstacles in its way. If it collides with the obstacle then the game will be over as the player has lost.

There are three main blocks of our game;

Input Block:

Input block will manage the input from the sensor and translate them into game.

Control Block:

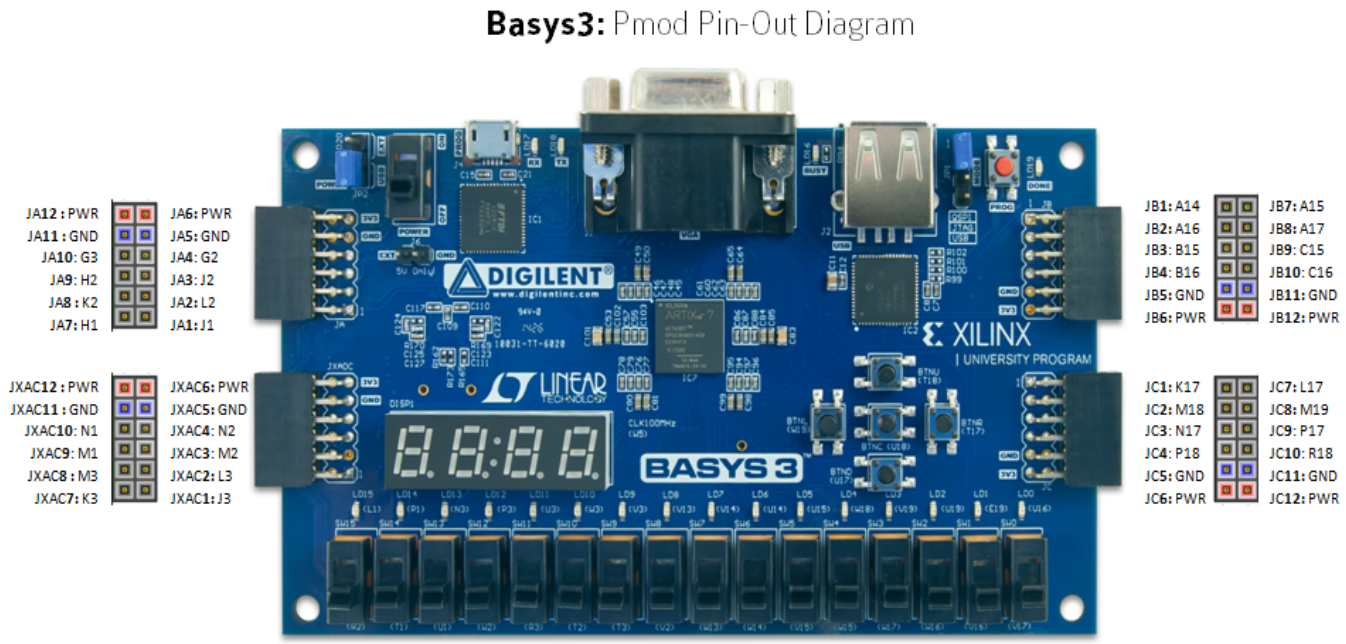
Control block will manage all the game's finite state machines, controlling start, game, movement screens and all the collision and counters.

Output Block:

Output block will manage all the displays, and display the images, pixel generation and animations etc.

INPUT BLOCK:

The Buttons in the BASYS – 3 FPGA boards are currently being used as input mechanisms of our game, later to be replaced by an IR Sensor. The IR sensor would be connected through PMOD input pins present on FPGA BASYS – 3 Board. We would use the JXADC port to interface our sensor. The interfacing will give us the data in the form of 0's and 1's.



PIN Details:

JB6: Power

JB5: GND

JB4: Input

I/O Details:

The module takes one input and one output; input is taken through the sensor which is then utilized in our game as the trigger for jump and duck.

The Infra-red sensor gives high signal when the receiver LED receives no input ie when nothing is in the range of the sensor, it provides a low output whenever the sensor detects something so we have assigned our port to the complemented value of what IR sensor is giving us as to make the dino jump and duck accordingly. The range of the sensor can also be adjusted as per requirements.

Code for sensor

```
module ir_input(sensor, out);
  input sensor;
  output out;

always@(sensor)
        out = ~sensor;

endmodule
```

Pin configuration for input block

Figure 1.8

CONTROL BLOCK

FINITE STATE MACHINES (FSMs) IN OUR PROJECT

1) Game FSM with design procedure and gate level implementation

State Diagram:

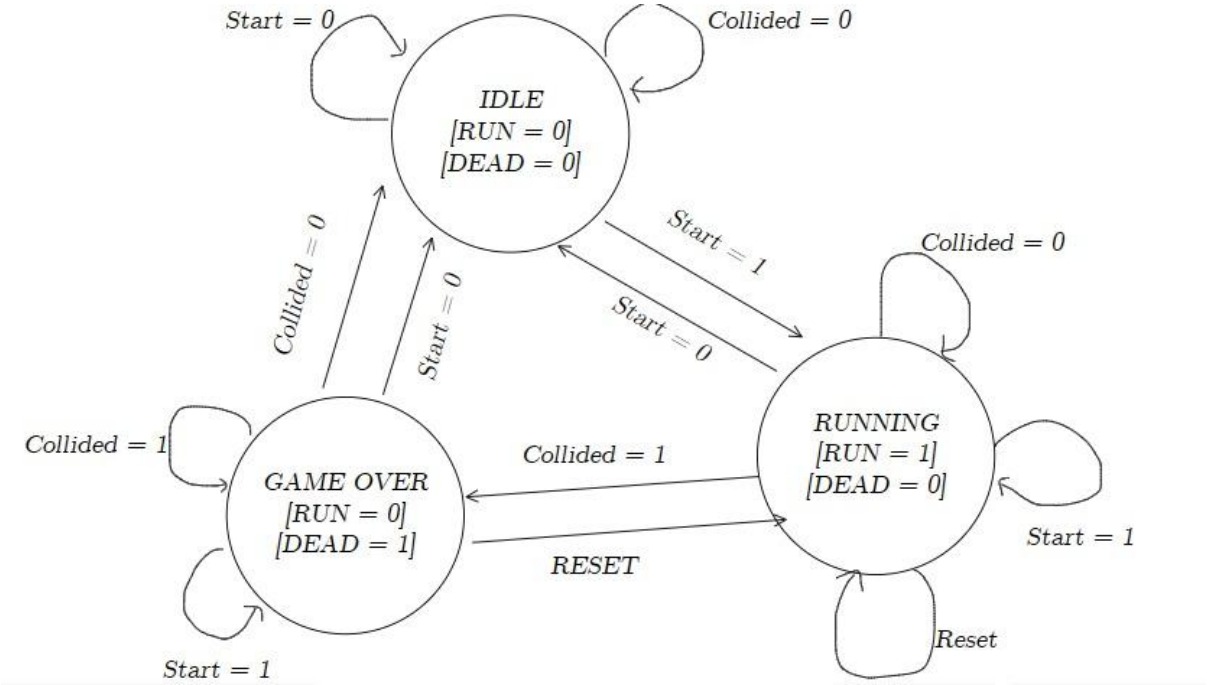


Figure 1.9

The FSM has three states: Idle, running and collided. In idle state the game is at main menu, in running state, the game is running and the player is still playing until it's collided, this means when collided, the player goes to the game over screen.

Assumptions:

- The game will go back to the main menu if you switch off the start button at any stage so this feature can also be used to pause the game at any point and continue from the same point by switching it on.
- The game starts over when the reset button is pressed

State Table:

Present State		Inputs			Next State		Outputs	
Q1	Q0	Start (S)	Reset (R)	Collided(C)	D1	D0	Run	Dead
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	1	0	0	1	0	0	0
0	0	1	0	1	1	1	0	0
0	0	1	1	0	1	0	0	0
0	0	1	1	1	1	0	0	0
1	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	1	0
1	0	0	1	0	0	0	1	0
1	0	0	1	1	0	0	1	0
1	0	1	0	0	1	0	1	0
1	0	1	0	1	1	1	1	0
1	0	1	1	0	1	0	1	0
1	0	1	1	1	1	0	1	0
1	1	0	0	0	0	0	0	1
1	1	0	0	1	0	0	0	1
1	1	0	1	0	0	0	0	1
1	1	0	1	1	0	0	0	1
1	1	1	0	0	0	0	0	1
1	1	1	0	1	1	1	0	1
1	1	1	1	0	1	0	0	1
1	1	1	1	1	1	0	0	1

Table 2.0

K-maps:

Map

	$\overline{R}.\overline{C}$	$\overline{R}.C$	$R.C$	$R.\overline{C}$
$\overline{Q1}.\overline{Q0}.S$	0	0	0	0
$\overline{Q1}.\overline{Q0}.S$	1	1	1	1
$\overline{Q1}.Q0.S$	x	x	x	x
$\overline{Q1}.Q0.\overline{S}$	x	x	x	x
$Q1.\overline{Q0}.S$	0	0	0	0
$Q1.\overline{Q0}.S$	1	1	1	1
$Q1.Q0.S$	0	1	1	1
$Q1.Q0.\overline{S}$	0	0	0	0

Figure 2.1 (K-map for d1)

	$\overline{R}.\overline{C}$	$\overline{R}.C$	$R.C$	$R.\overline{C}$
$\overline{Q1}.\overline{Q0}.S$	0	0	0	0
$\overline{Q1}.\overline{Q0}.S$	0	1	0	0
$\overline{Q1}.Q0.S$	x	x	x	x
$\overline{Q1}.Q0.\overline{S}$	x	x	x	x
$Q1.\overline{Q0}.S$	0	0	0	0
$Q1.\overline{Q0}.S$	0	1	0	0
$Q1.Q0.S$	0	1	0	0
$Q1.Q0.\overline{S}$	0	0	0	0

Figure 2.2 (K-map for d0)

	$\overline{R}.\overline{C}$	$\overline{R}.C$	$R.C$	$R.\overline{C}$
$\overline{Q1}.\overline{Q0}.S$	0	0	0	0
$\overline{Q1}.\overline{Q0}.S$	0	0	0	0
$\overline{Q1}.Q0.S$	x	x	x	x
$\overline{Q1}.Q0.\overline{S}$	x	x	x	x
$Q1.\overline{Q0}.S$	1	1	1	1
$Q1.\overline{Q0}.S$	1	1	1	1
$Q1.Q0.S$	0	0	0	0
$Q1.Q0.\overline{S}$	0	0	0	0

Figure 2.3 (K-map for run led)

	$\overline{R}.\overline{C}$	$\overline{R}.C$	$R.C$	$R.\overline{C}$
$\overline{Q1}.\overline{Q0}.S$	0	0	0	0
$\overline{Q1}.\overline{Q0}.S$	0	0	0	0
$\overline{Q1}.Q0.S$	x	x	x	x
$\overline{Q1}.Q0.\overline{S}$	x	x	x	x
$Q1.\overline{Q0}.S$	0	0	0	0
$Q1.\overline{Q0}.S$	0	0	0	0
$Q1.Q0.S$	1	1	1	1
$Q1.Q0.\overline{S}$	1	1	1	1

Figure 2.4 (K-map for dead led)

State equations:

	State Equations
D1:	Qo' . Start + Start . Collided + Collided . Reset
Do:	Start . Reset' . Collided
Run:	Q1 . Qo'
Dead	Qo

Gate-level implementation:

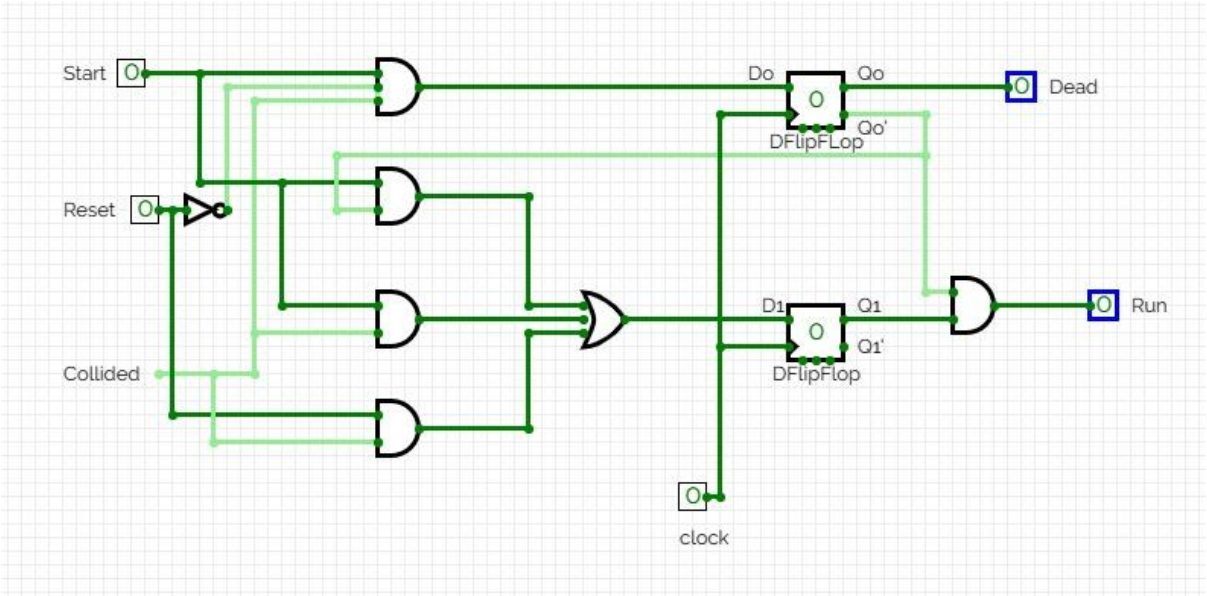


Figure 2.5 (logic gate diagram using d Flip-Flop)

Block Diagram for verilog module of Game FSM:

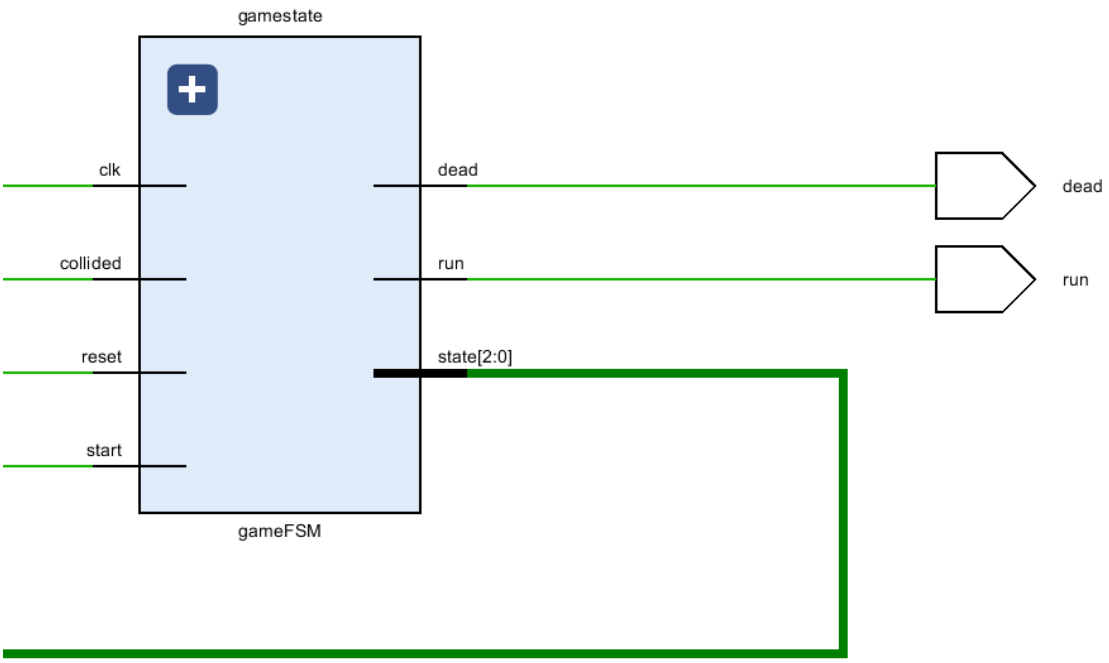


Figure 2.6 (block diagram)

Module Description:

This module takes start collided and reset inputs. The game is initially at idle state i.e. 00. If the dino collides with the obstacle, collided is true and the game state changes to the dead state i.e 11 or game over screen. this section lights up the collided and dead led and shuts of the run led. Otherwise the run led is lit up, dead and collided leds are off.

2) Dino FSM State Diagram

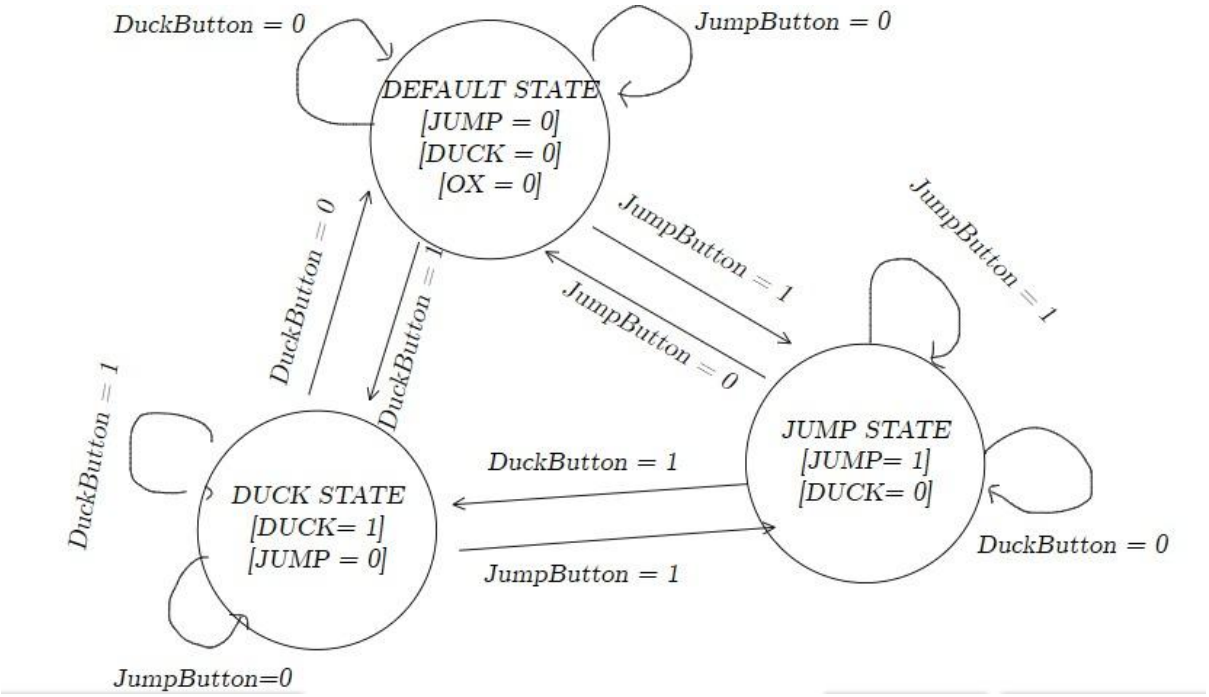


Figure 2.7 (State diagram for Trex/ Dino)

Description and Assumptions:

- The ox integer here represents the number of units the object is supposed to move up by when sensor gives a high input for jump. This ox is a dont-care condition in duck state because the ducked dino is always supposed to be on ground
- The FSM will only reach default state when the game state from the game FSM is 10 i.e. the game is running.

- If an input of duck is sent when the dino is jumping, the dino will jump and vice versa for jumping state too.

Block diagram for verilog module of Dino Fsm:

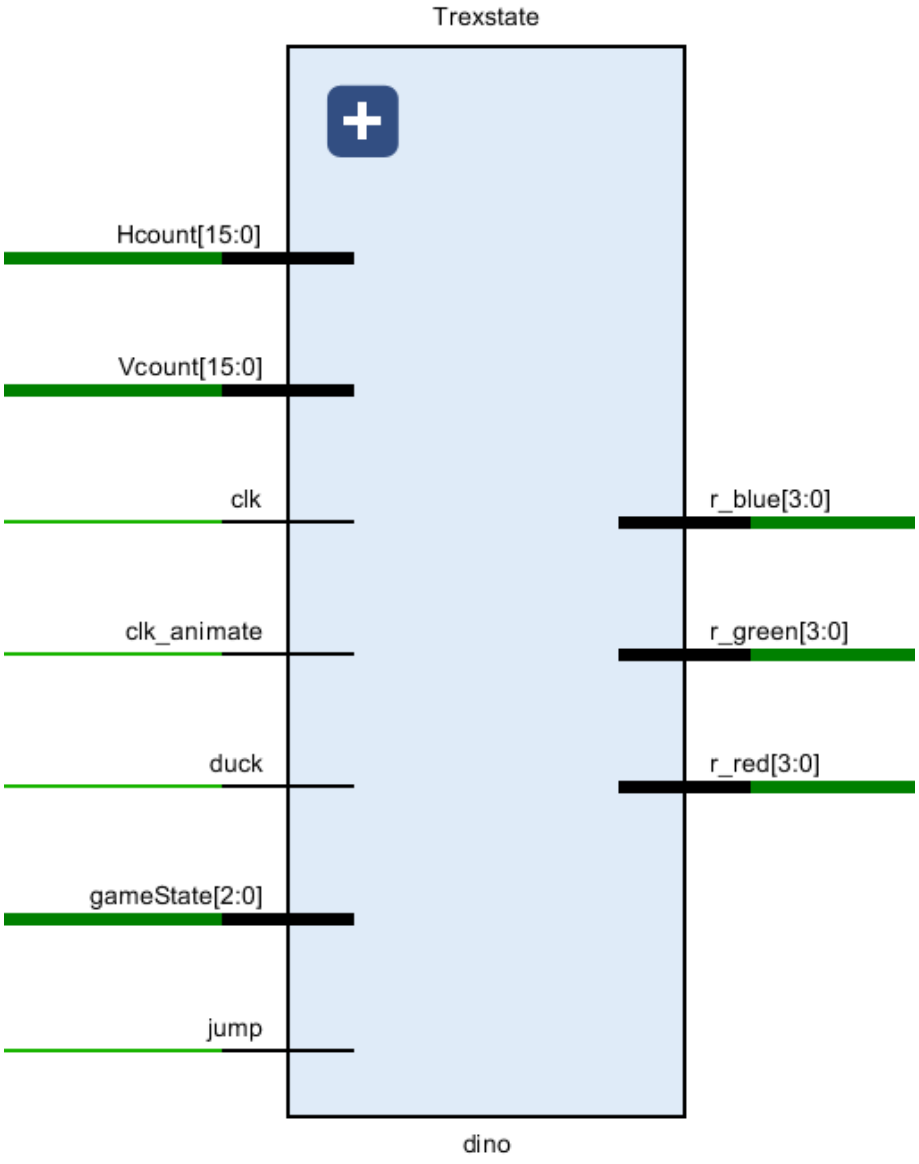


Figure 2.8 (Block Diagram for Dino Fsm)

Module description of Dino FSM:

The module generates and animates the Dino on the screen, it uses the animation clock to move the Dino legs up and down at a frequency of 4 Hz. This animation movement is independent of the jump state. The dino is only generated when the game is running i.e. gamestate is 10. The module uses conditional if-else and nested if-else statements to see whether the Hcount and Vcount values lie in the particular range and then assign colors. We detected this range by drawing our image on a graph paper. Furthermore it uses an integer ox to move the dino up by 120 denary units when we give a jump input and draw a ducked image when we give the duck input.

Image on graph paper:

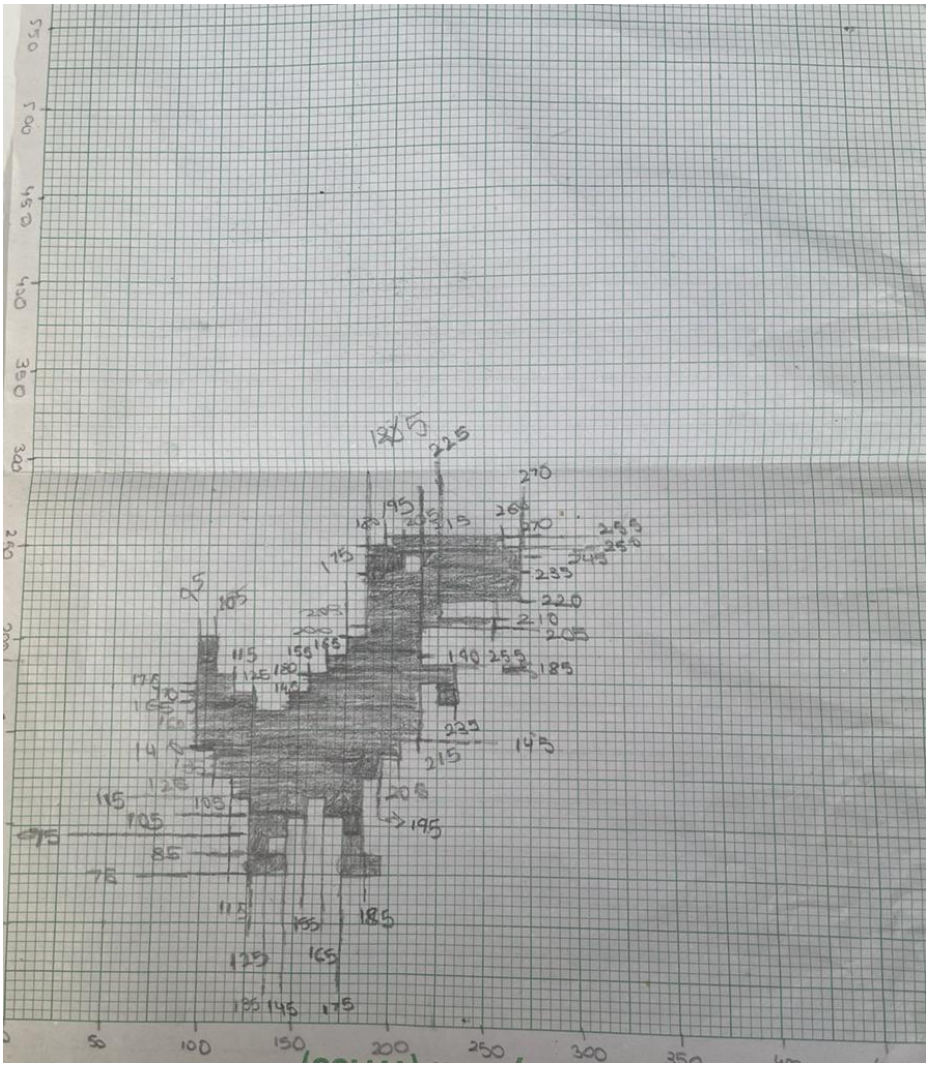


Figure 2.9 (image on graph paper)

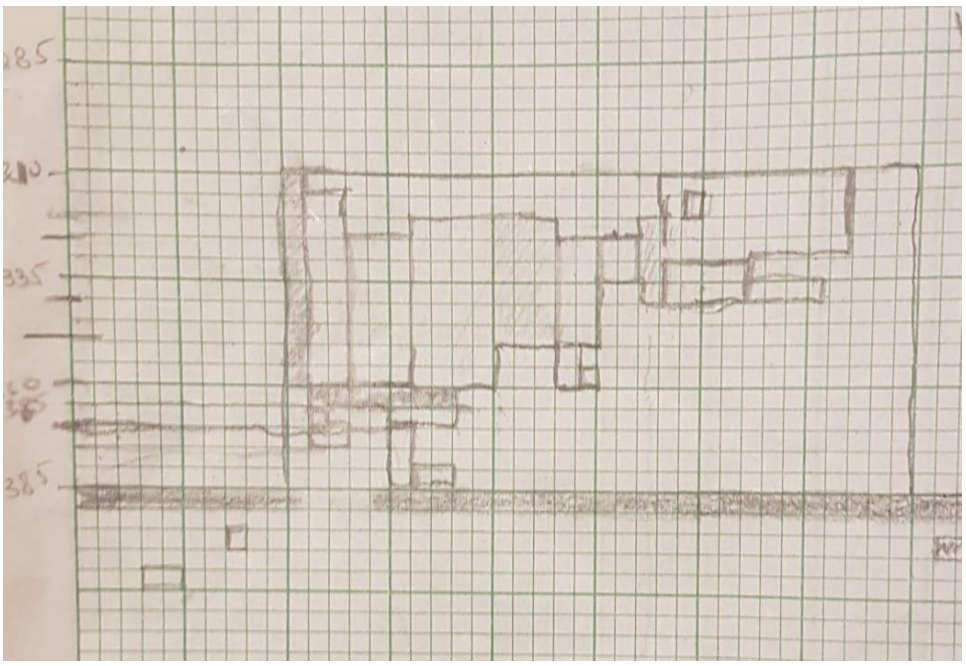


Figure 3.0 (Duck state on graph)

3) Obstacle FSM state diagram

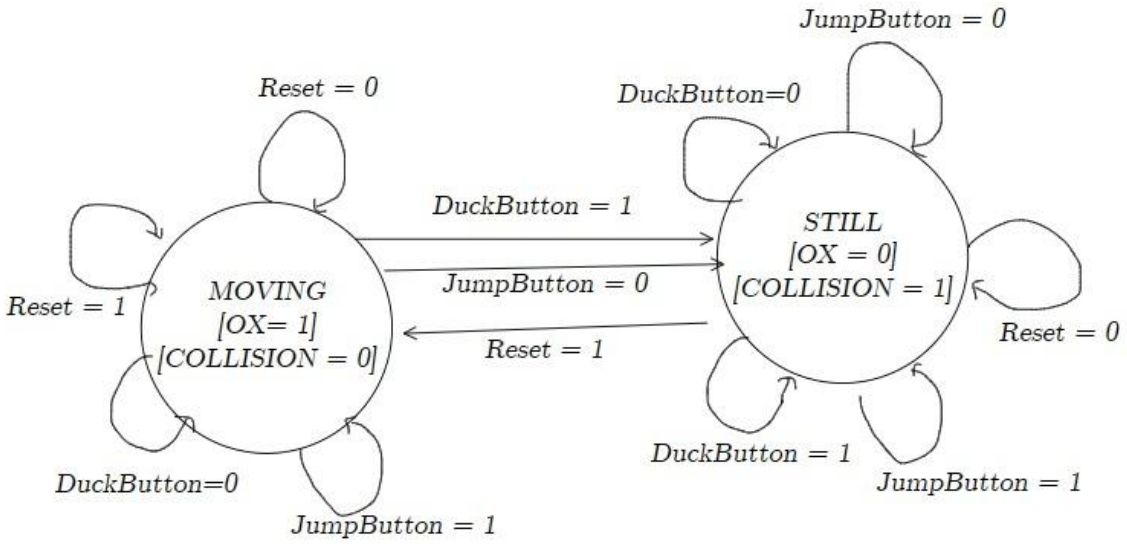


Figure 3.1 (state diagram for obstacle)

Description and Assumptions:

-ox is an integer that changes the position of the cactus based on some conditions.

-if ox=1 in the state diagram, ox is increasing on every positive edge of the obstacle clock else if ox=0, the obstacle stays at its place.

- The state only changes when the dino and obstacle are adjacent to each other so the state diagram represents that scenario here and it is only then when the jump button and duck button affects the obstacle's state.

Block diagram for verilog module of obstacle:

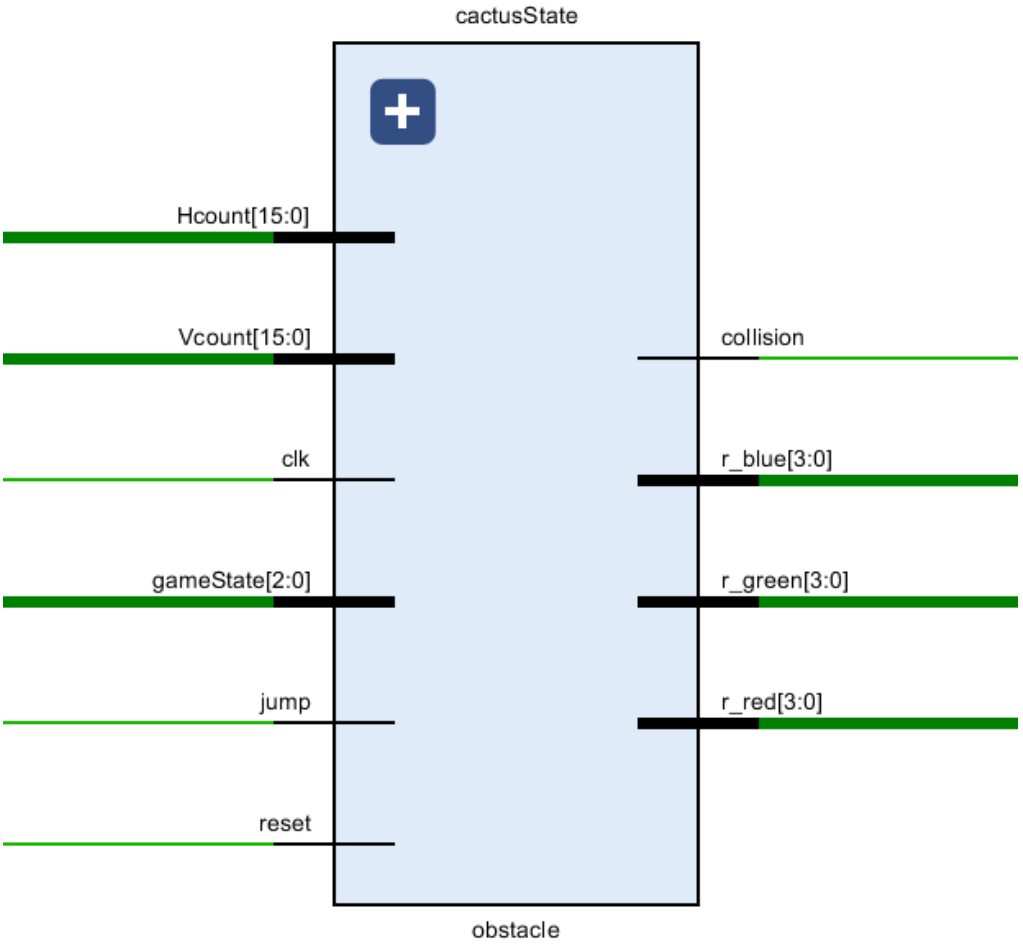


Figure 3.2 (Block Diagram of obstacle)

Module description:

The module generates an obstacle clock using the clock divider module that is supposed to move the cactus from the right of the screen to the left. The movement depends on the jump and reset condition. If reset input is given the cactus starts its movement again from the right of the screen. If jump button is pressed the cactus continues its right to left movement otherwise it is paused when it collides with the dino, this is only when no jump input is given is duck input is given instead of jump. The collision is detected in this way. The movement is dependent on the integer ox, that adds 10 denary values to itself at every positive edge of obstacle clock and continues doing so if jump and duck input is given at correct times. The object was generated thru graph paper.

Cactus image on graph paper:

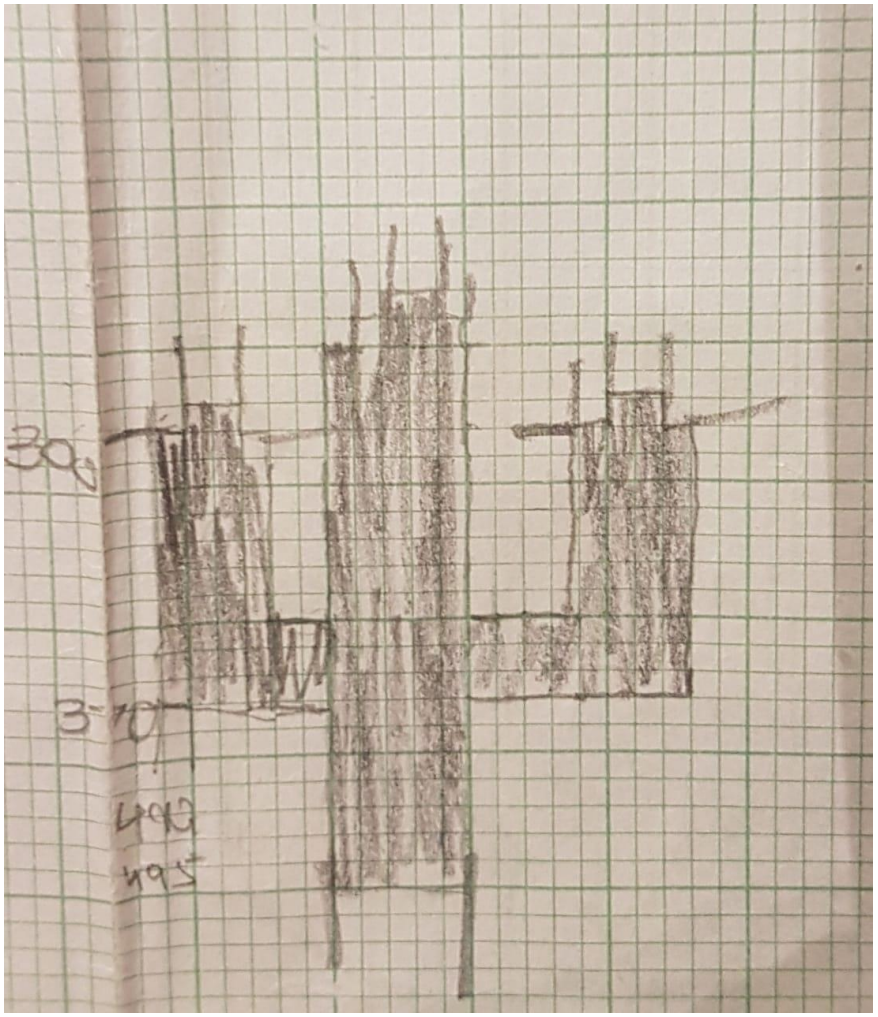


Figure 3.3 (Cactus image on graph paper)

MainScreen module block Diagram:

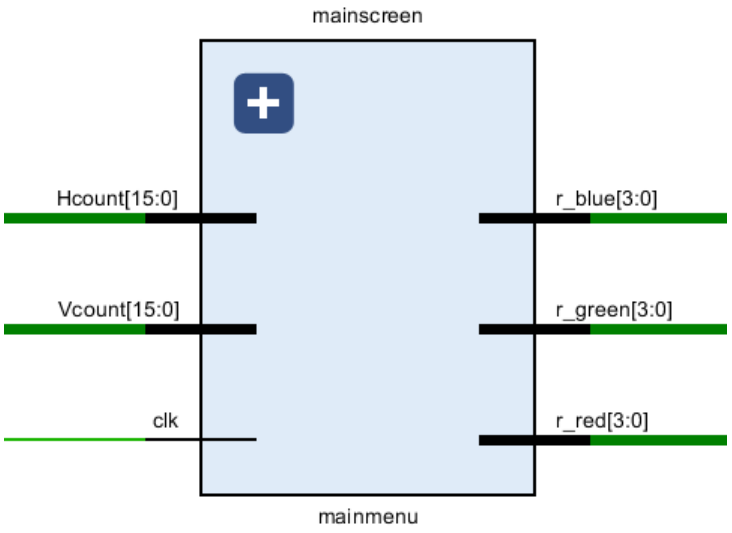


Figure 3.4 (Block diagram for main screen)

Module Description:

The module generates the pixels for the main screen where it displays the text “STArT” in a box. We generated this through the graph paper too. The main screen is only displayed when the start button is off

Main screen on graph paper:

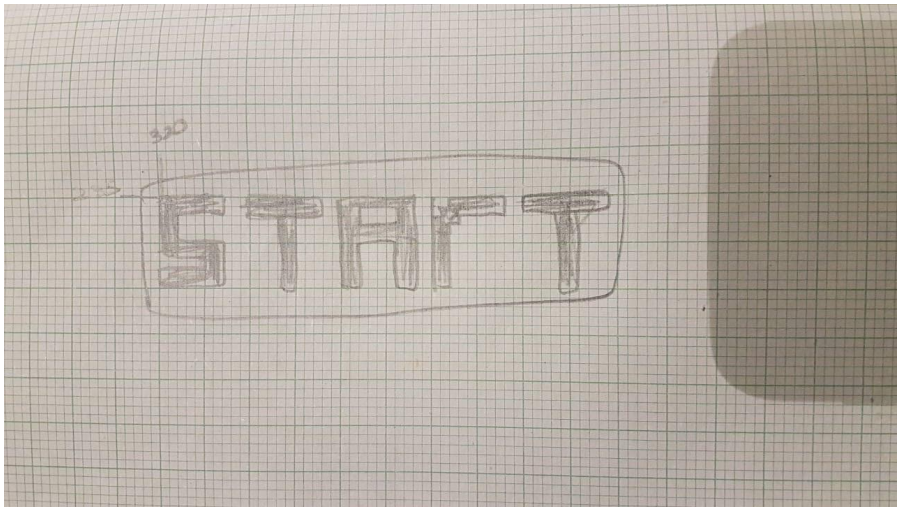


Figure 3.5(Main screen on graph paper)

Game over screen module block diagram:

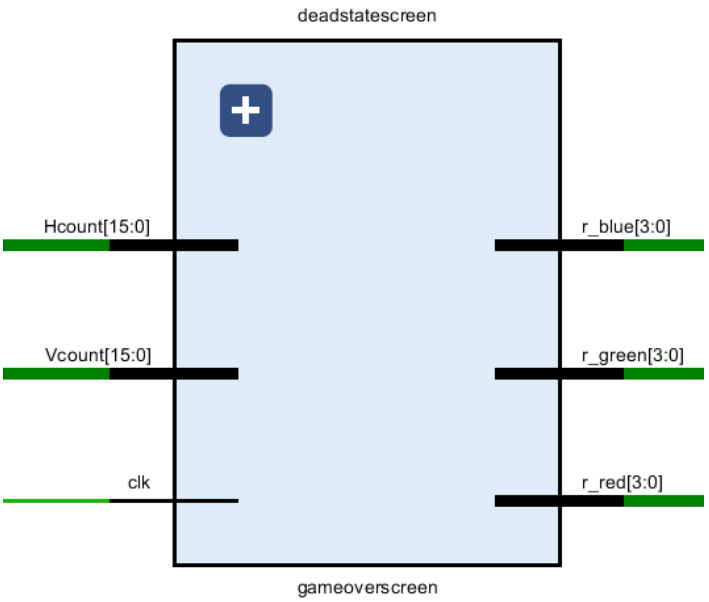


Figure 3.6 (game over block diagram)

Module Description:

The module generates the pixels for the main screen where it displays the text “GAME OVER”. We generated this through the graph paper too. The main screen is only displayed when the start button is on and the dino and cactus are collided

Game over screen on graph paper:

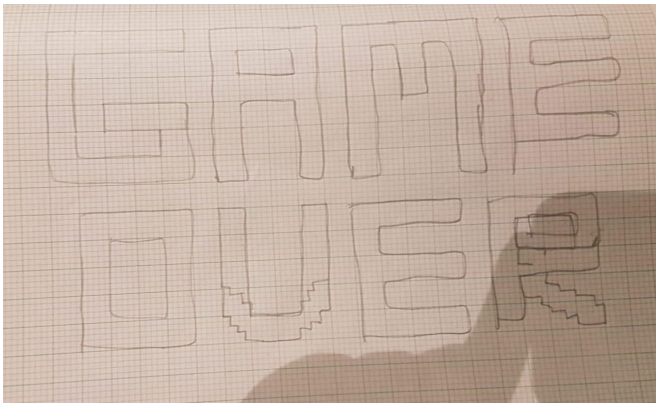


Figure 3.7 (game over screen on graph paper)

OUTPUT BLOCK

The clock_divider() module generates a clock of 25MHz which is sent to Horizontal_counter() and vertical_counter() modules. The horizontal_counter() module increases the Hcount and vertical_counter module increases the Vcount when Hcount reaches its maximum value (right most corner of the screen) and generates an enable signal. This is how all the pixels of the screen are accessed. The draw modules of each object assigns an RGB value to every pixel of the screen by using registers which is called by a Top level module top.v() that calls all the modules of the game which generates and updates the output of the screen coming from the Control Block. The display of each object is controlled by select statements for each RED, GREEN, and BLUE output registers. The select statement has different combinations of buttons and outputs to select what to display on the screen when a certain condition is met.

Pin configuration for output block:

Figure 3.8 (pin configuration for LEDs and clock)

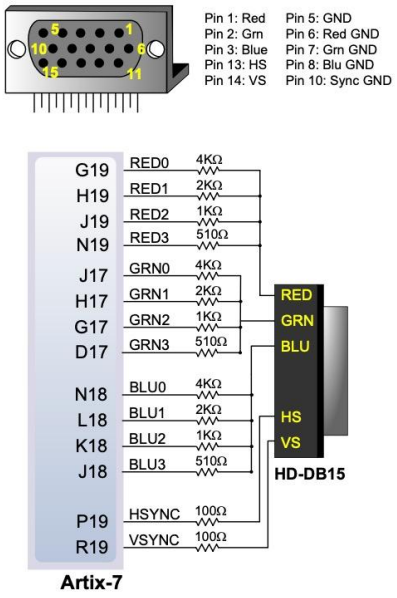


Figure 3.9 (VGA configuration pins)

Block diagrams for top level module

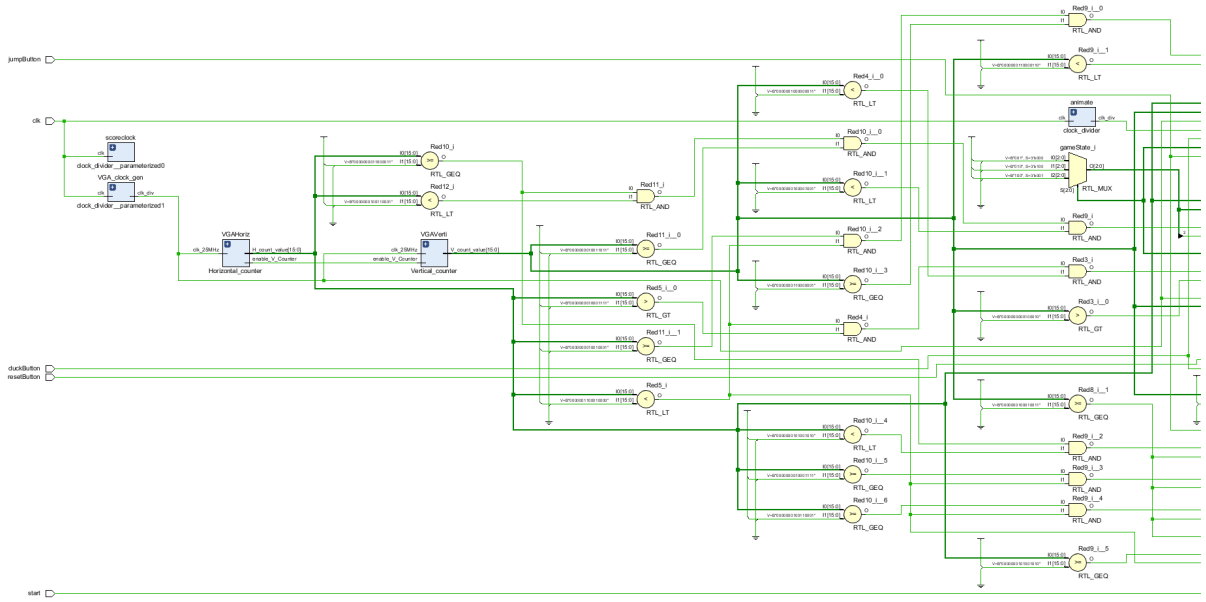


Figure 4.0 (top level module block diagram part1)

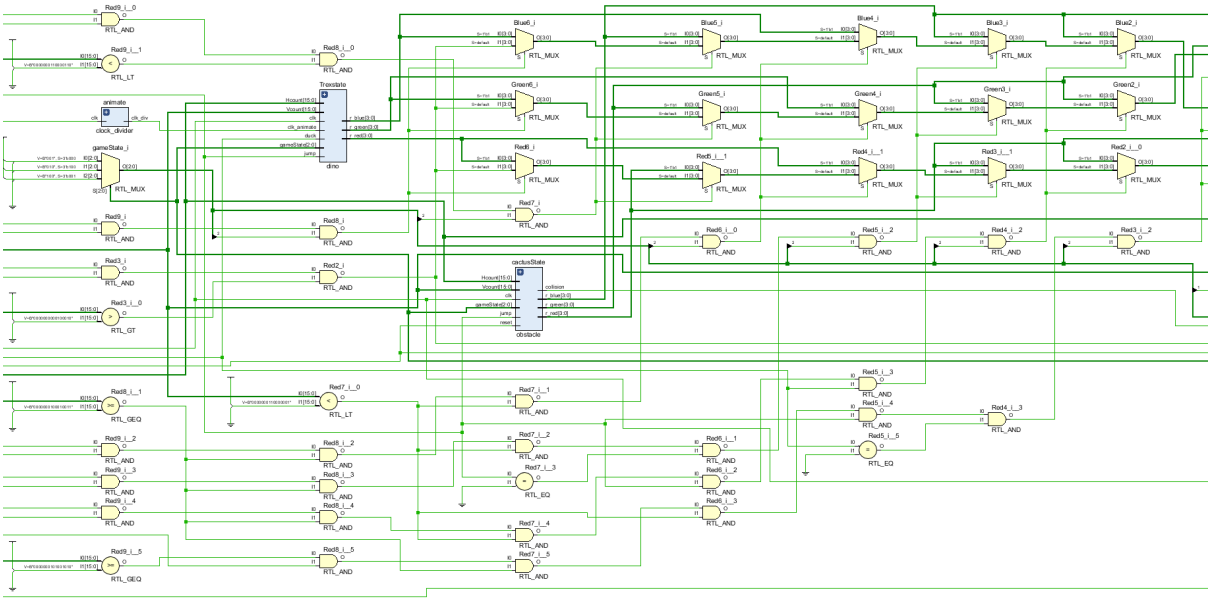


Figure 4.1 (top level module block diagram part 2)

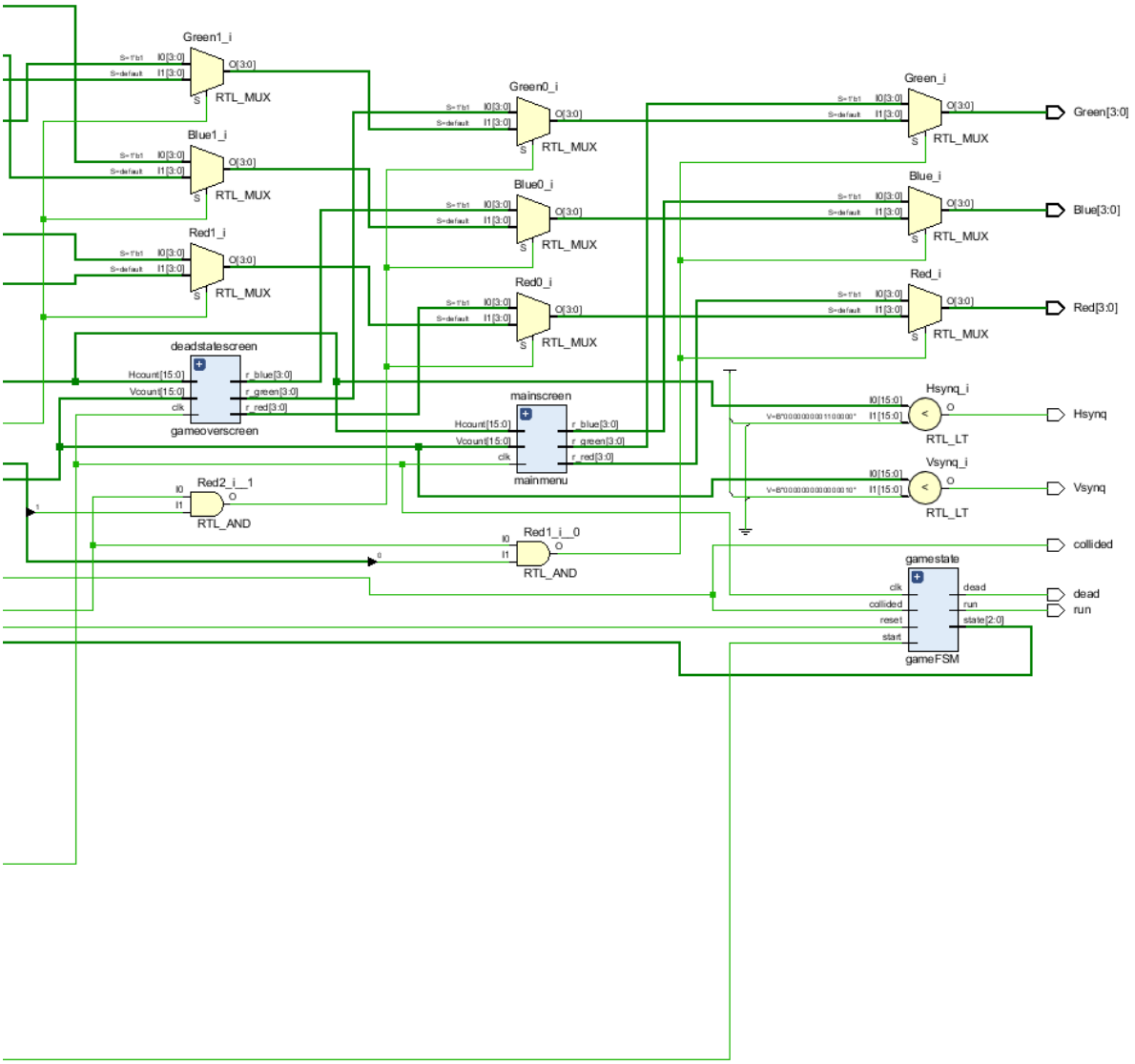


Figure 4.2 (top level module block diagram part3)

Module Description:

We call the clock divider module to generate animation clock and VGA clock , then go onto call Horizontal_counter and Vertical_counter modules to calculate Hcount and Vcount. We also call DinoFSM, GameFSM, obstacleFSM, gameoverscreen and mainmenuscreen modules in the top level module. Then we go on to assign Hsynq and Vsync values using following assignments:

assign Hsynq = (Hcount<96)? 1'b1:1'b0;
assign Vsynq = (Vcount<2)? 1'b1:1'b0;

Lastly we assign values to the Red, Green and Blue nets of the VGA connector. The values are assigned on the basis of conditions met in each select statement.

Clock Divider Module Block Diagram:

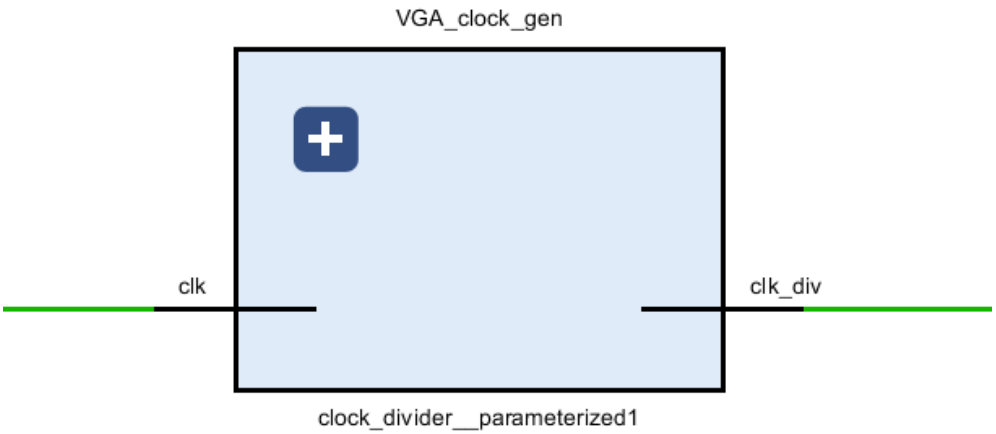


Figure 4.3 (Block Diagram for clock divider)

Module Description:

The module takes an overloaded parameter div_val and negates the value of original clock until a count variable reaches that div_val. Div_val is calculated by the formula:

Div_val= (Current frequency/2*Required Frequency) -1

This way it divides the clock to our required frequency.

Block diagram for vertical_counter:

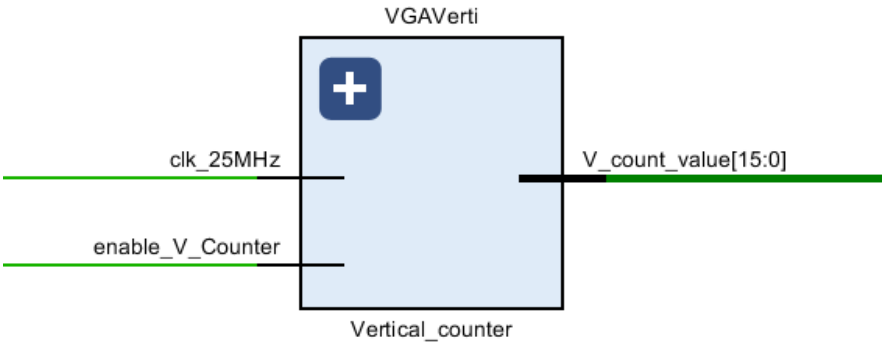


Figure 4.4 (block diagram)

Block Diagram for Horizontal_counter:

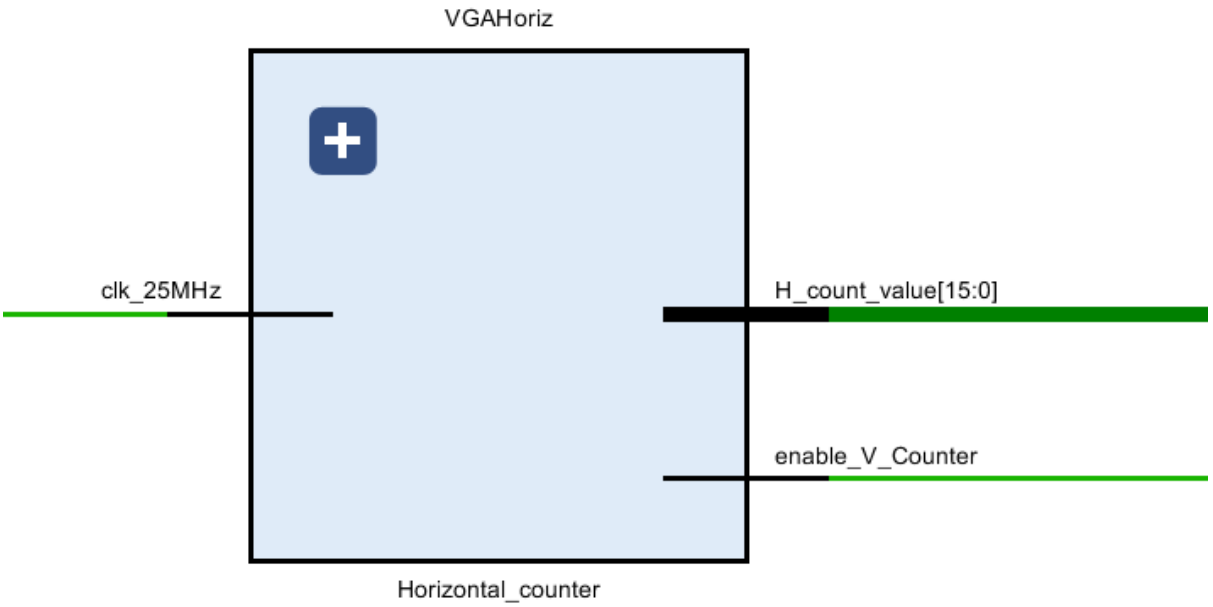


Figure 4.5 (Block Diagram)

REFERENCES

[1] How to Create VGA Controller in Verilog on FPGA? | Xilinx FPGA Programming Tutorials , 28 Nov 2018. Accessed on: Dec. 05, 2021. [Video file]. Available: https://www.youtube.com/watch?v=4enWoVHCykI

[2] VGA image driver (make a face) on an Intel FPGA, 8 Nov 2020. Accessed on: Dec. 05, 2021. [Video file]. Available: https://www.youtube.com/watch?v=mR-eo7a4n5Q&t=13s