Digital Logic and Design

Project Progress Report

“Eat Up”

Logic Mates

Team Members:

• Saad Qureshi 06408 T1 L2

• Saad Fahim 06583 T1 L4

• Khuzaima Ali Khan 06466 T1 L4

• Affan Ullahhabib 06358 T1 L4

Description (Updated)

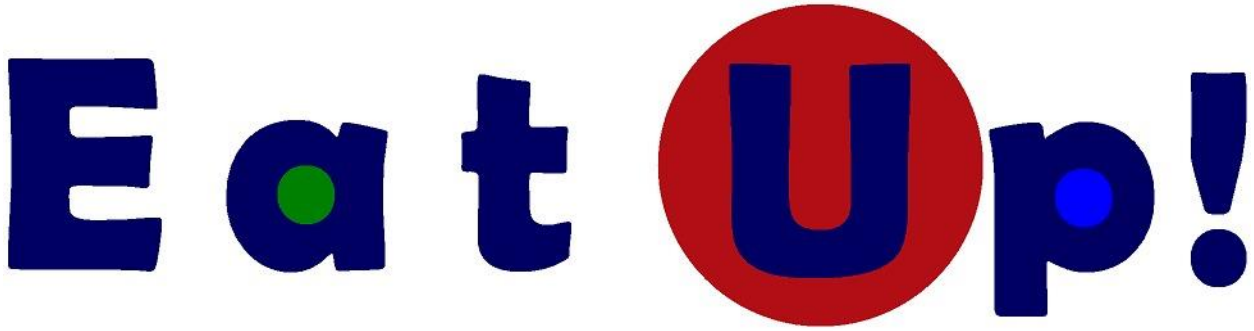
For our Digital Logic and Design Project we made an endless running game called “Eat Up” where the player is a red circle. The objective of this game is to eat as many blue circles as they can while watching out for the green circles.

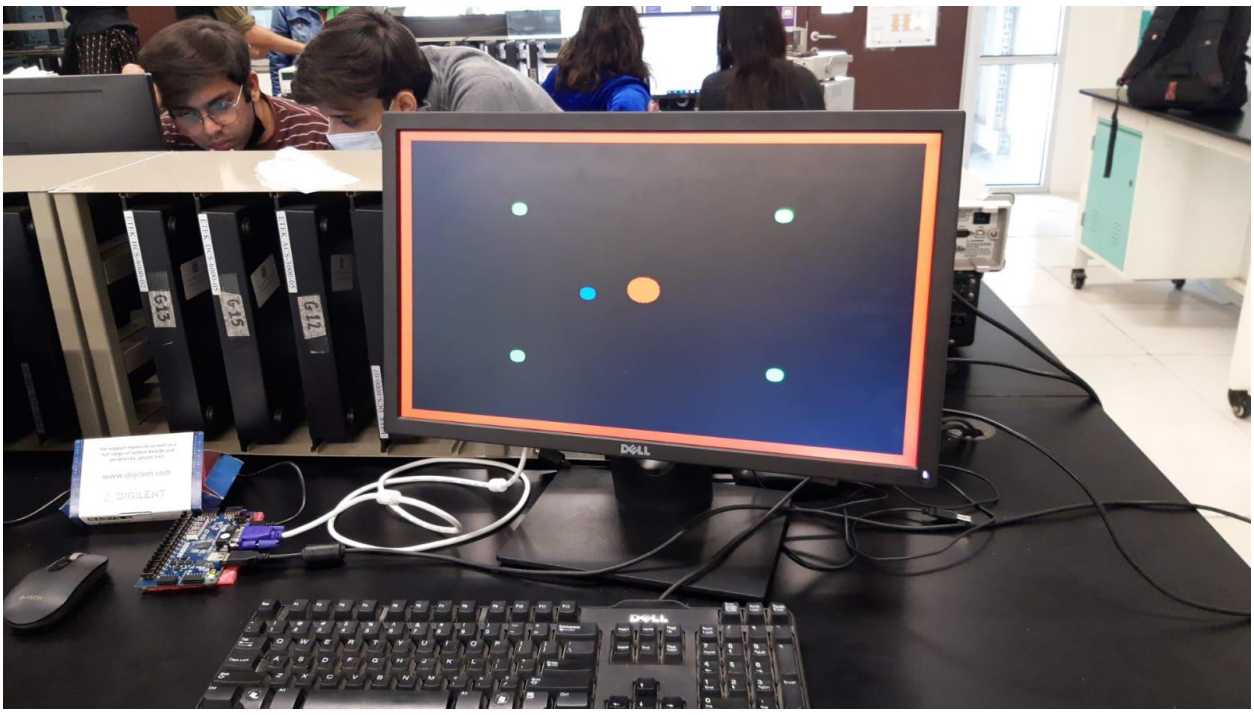
When game is started the all circles are in embargo until the user presses Enter.

Each time the player circle collides with a blue circle their radius increases and each time they collide with a green circle their radius decreases. If the circle size decreases 5 times the player loses. The blue circle stays in its place while the green circles keep moving randomly.

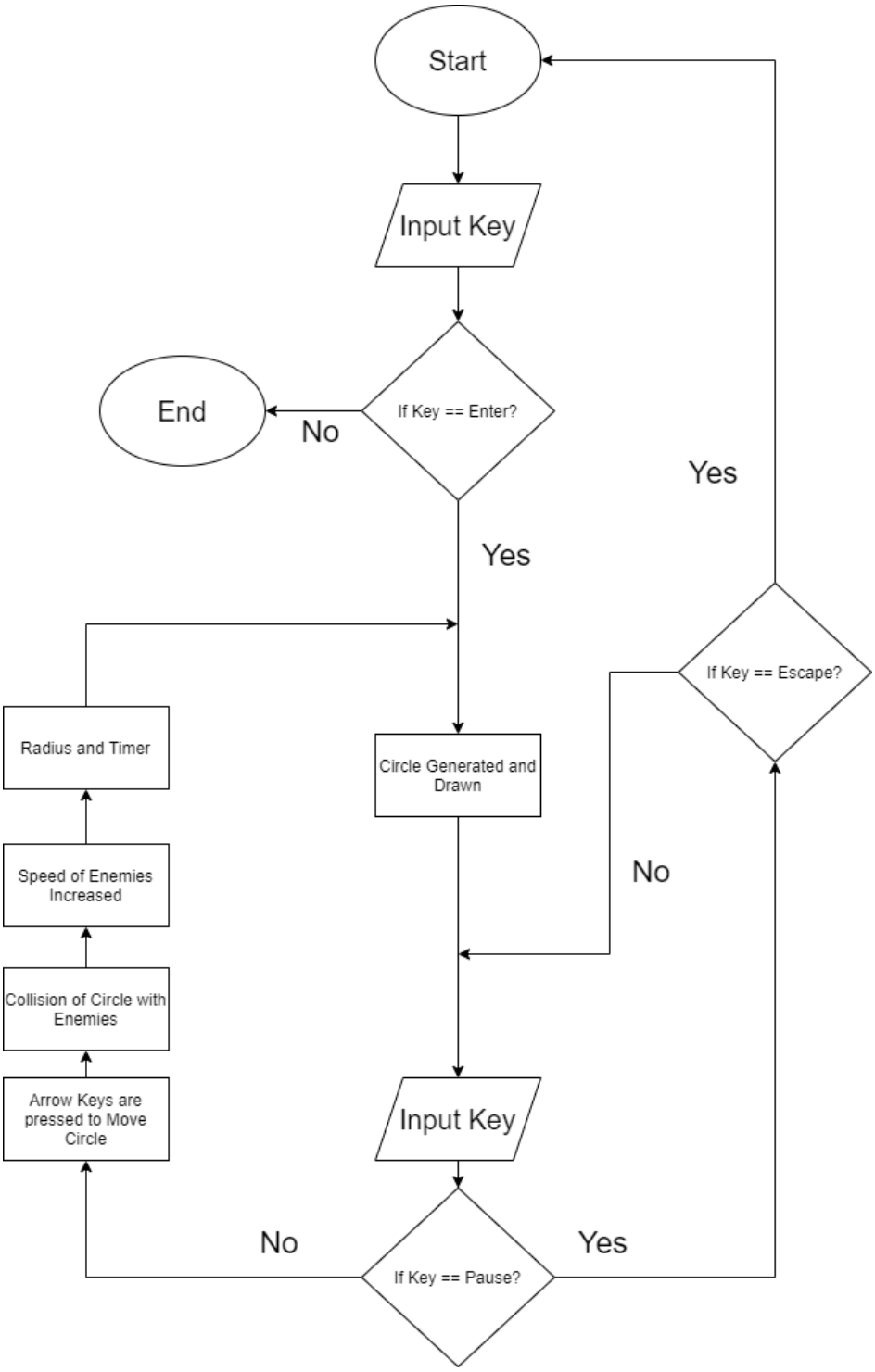
At any point the user can press the delete key to pause the game and press escape to return to the reset state. If they press enter while on pause the game resumes.

If the circle decreases to a radius of 10 the player loses the game and the game goes to pause screen. The player can press escape to start the game again.

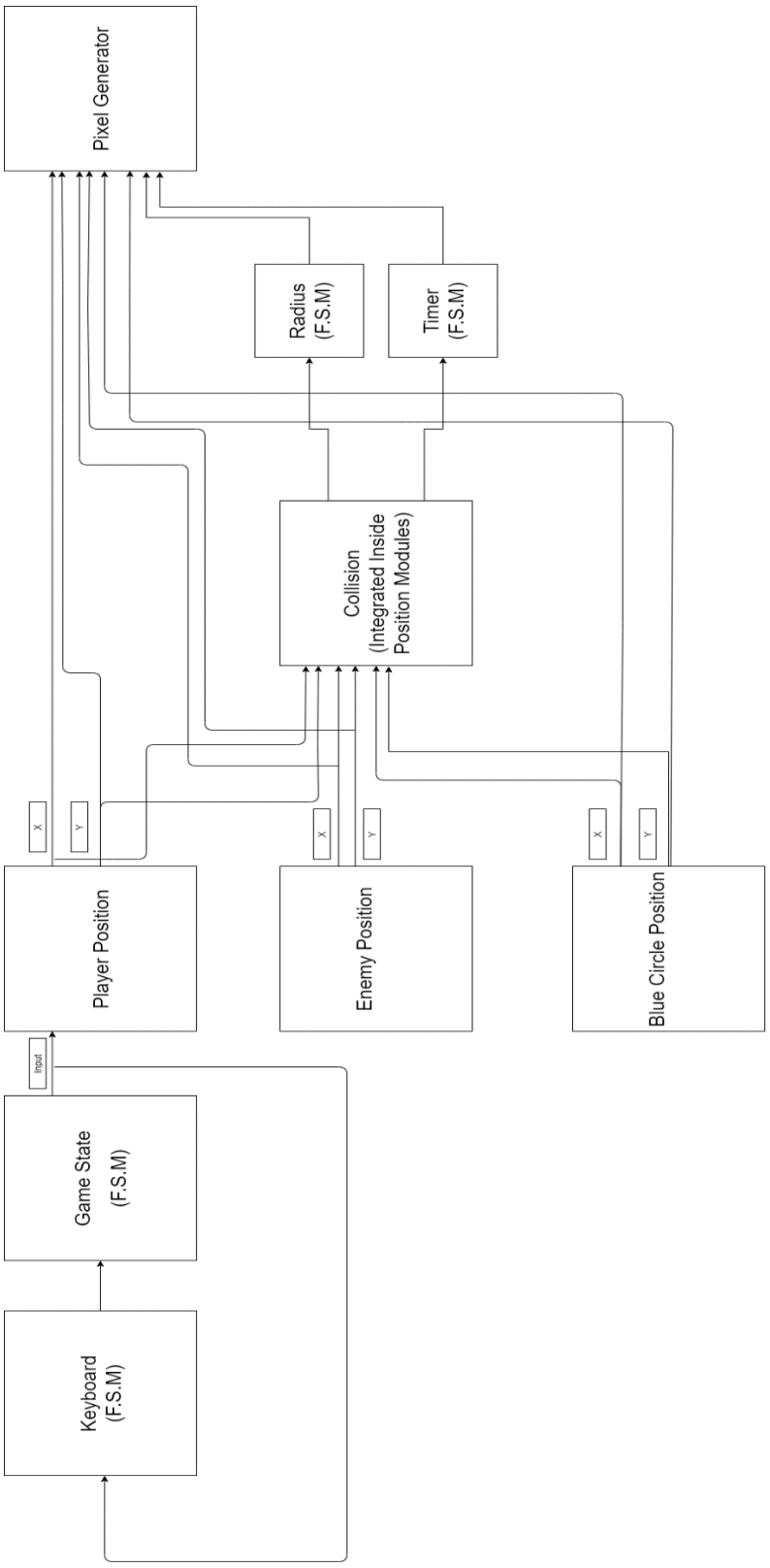




User Flow Diagram (Updated)



Block Diagram (Updated)



The game starts with the keyboard module which sends the signals for the keys into the appropriate module. The GameState F.S.M module changes the game according to the relative key. When the game is running the movement and collision modules are responsible for the control of circles on screen. The x and y coordinates for the circles are sent to the pixel generator to be displayed on screen.

Control and Input Block:

The major F.S.Ms the game is using for the control blocks and input blocks are:

- GameState(Completely Gate Level)
- Radius
- Keyboard
- Timer
- Movement

GameState F.S.M (Design and Analysis)

The GameState F.S.M has a completely gate level implementation.

It has three inputs which come from the Keyboard module:

- Enter
- Pause
- Escape

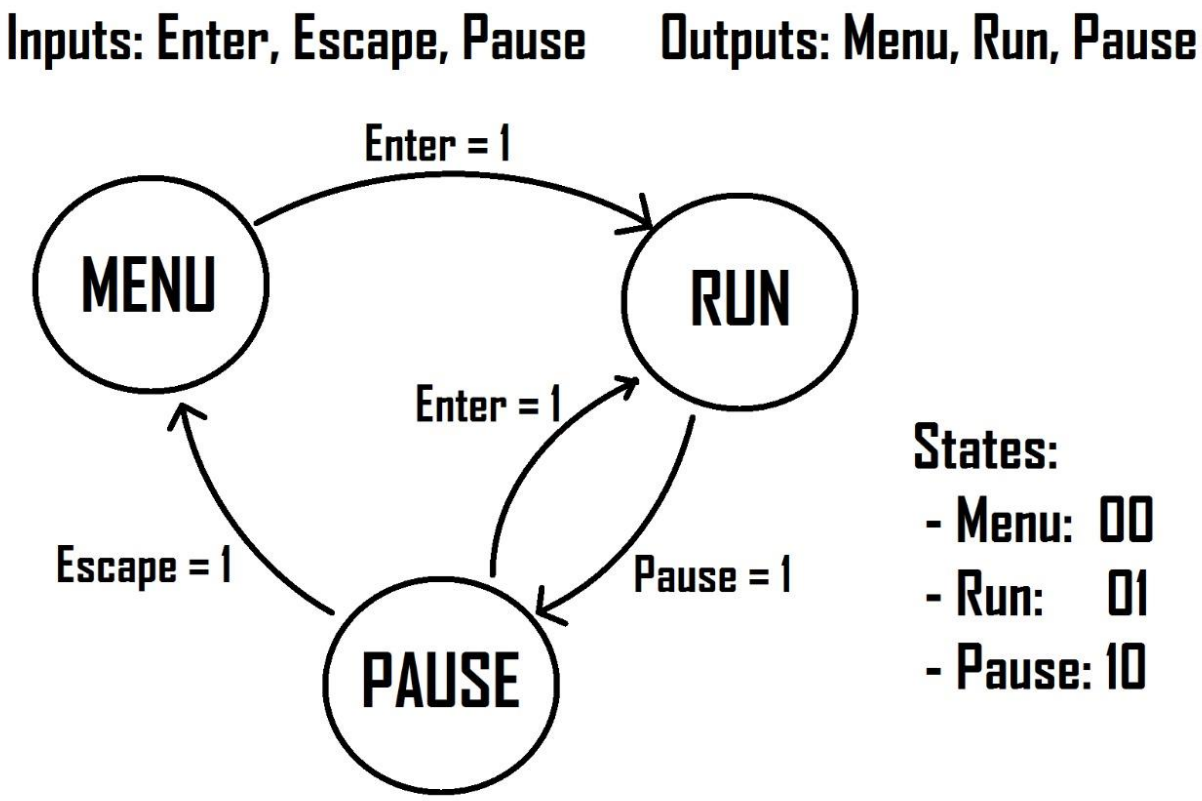
The states and outputs are:

- GameRun
- GamePause
- GameMenu

For this F.S.M a D-Flip-Flop is being used. The rest of the modules initialize all the necessary variables when GameState is GameMenu.

This F.S.M is a Moore F.S.M.

State Transition Diagram:



State Transition Table:

Present States		Inputs			Next State		Output		
A	B	Enter (C)	Escape (D)	Pause (E)	A(t+1)	B(t+1)	Menu	Run	Pause
0	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0
0	0	0	1	1	0	0	1	0	0
0	0	1	0	0	0	1	1	0	0
0	0	1	0	1	0	0	1	0	0
0	0	1	1	0	0	0	1	0	0
0	0	1	1	1	0	0	1	0	0
0	1	0	0	0	0	1	0	1	0
0	1	0	0	1	1	0	0	1	0
0	1	0	1	0	0	1	0	1	0
0	1	0	1	1	0	1	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	1	1	0	0	1	0	1	0
0	1	1	1	1	0	1	0	1	0
1	0	0	0	0	1	0	0	1	0
1	0	0	0	1	1	0	0	1	0
1	0	0	1	0	0	0	0	0	1
1	0	0	1	1	1	0	0	0	1
1	0	1	0	0	0	1	0	0	1
1	0	1	0	1	1	0	0	0	1
1	0	1	1	0	1	0	0	0	1
1	0	1	1	1	1	0	0	0	1

Characteristic Equations:

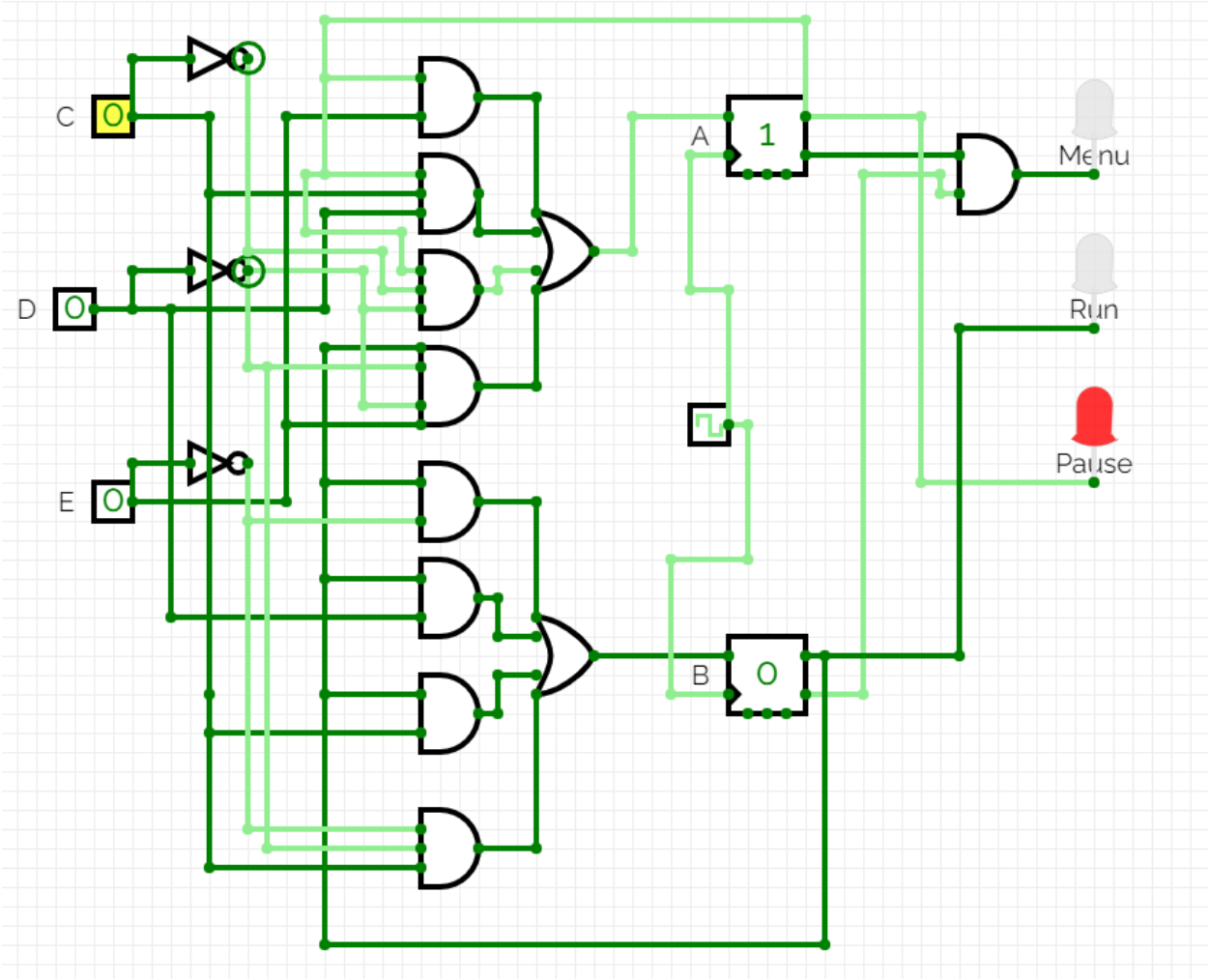
- Next State Equations:

A(t+1) = A&E | A&~C&~D | A&C&D | B&~C&~D&E

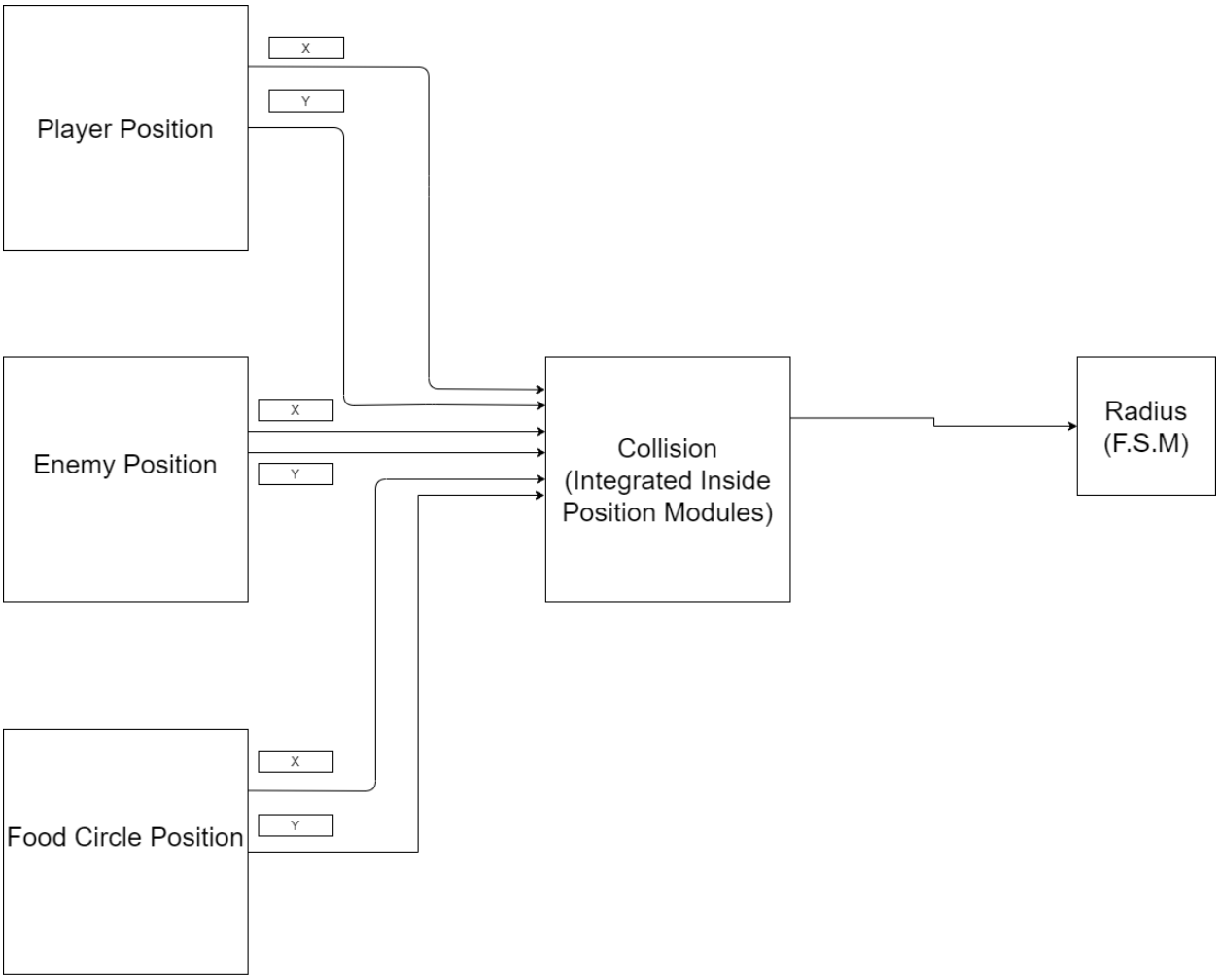
B(t+1) = B&~E | B&D | B&C | C&~D&~E;

- Output Equations:
  Menu = ~A&~B
  Pause = A
  Run = B

Circuit Diagram:



Movement F.S.Ms (Player Circle, Enemy Circle, Food Circle)



All the F.S.Ms below update on positive edge with a delay of 0.3 seconds for smooth movement.

Player Circle

When GameState is GameMenu the F.S.M initializes the player circle x and y center points which are 10-bit registers to the center of the screen.

When GameState is GameRun the F.S.M moves the player circle according to the key pressed by user within the boundaries of the screen.

The boundaries have width and height of 15 pixels.

Enemy Circle

There are four enemies being displayed on screen. Each move with a velocity (also 10-bit registers) in x and y direction at a delay of 0.3 seconds using clock divider.

If the enemy collides with a boundary of the wall, the two's compliment of the velocity is taken in the respective direction.

If enemy collides with player, a collision signal to the radius F.S.M and the ball is bounced off the player by taking two's compliment of the velocity in the respective direction. Whenever the enemy collides with the player their speed increases by 2.

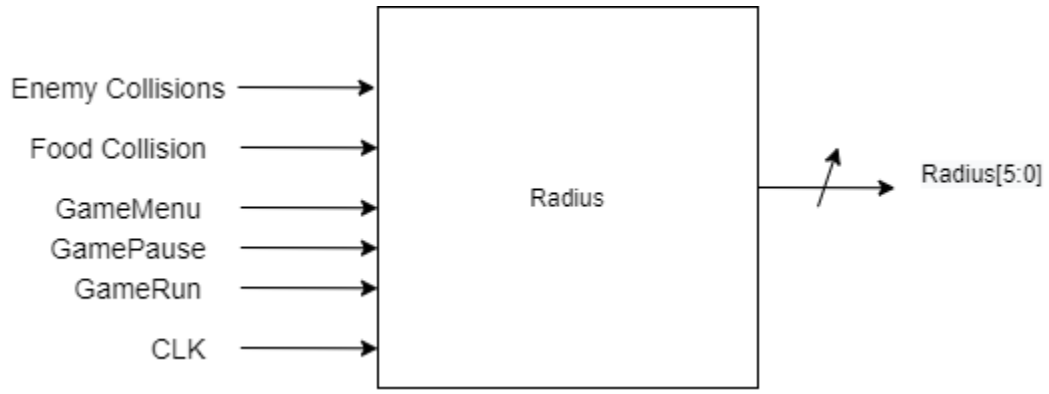
Food Circle

There is one food circle being displayed on screen at a time with a fixed position.

If the player collides with this circle, a collision signal is sent to the radius F.S.M to increase the radius. At the same time, the x and y coordinates of the circle are changed to another position.

There are total of 8 positions where the blue circle will appear.

Radius F.S.M

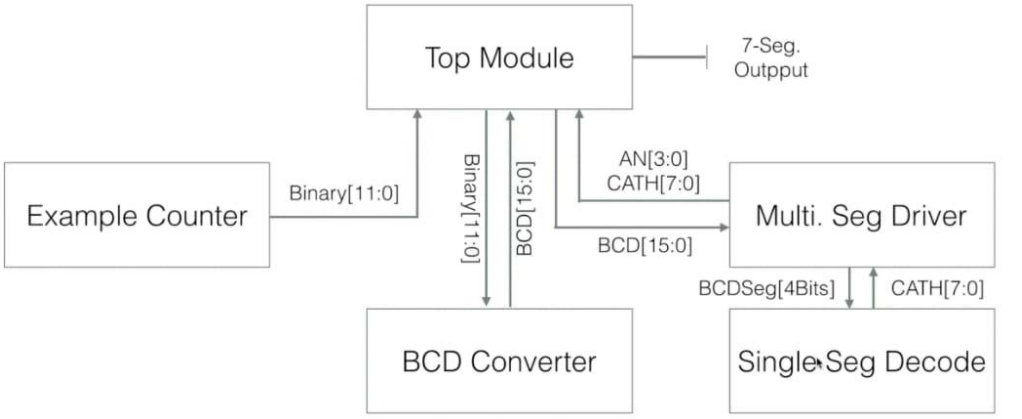


When GameState is GameMenu the F.S.M initializes the player circle radius (6-bit register) to 20.

When GameState is GameRun the F.S.M performs the task of decreasing the radius for player ball when there is a collision with red circles. If there is a collision with blue circles the radius increases by 2. If the radius is 10 the Keyboard module sends a signal of pause to the GameState module which pauses the game.

When GameState is GamePause the F.S.M does nothing.

Timer F.S.M (Design and Analysis)



It contains of 5 modules in total: example counter, bcd counter, multi segment driver, single segment decoder and top module.

Example counter is used to generate a count from 0 to 4099 at a frequency of 25 Mhz which is achieved by using a clock divider.

When GameState is GameMenu the timer is initialized to 0 and when GameState is GameRun timer is incremented. When GameState is GamePause timer is paused.

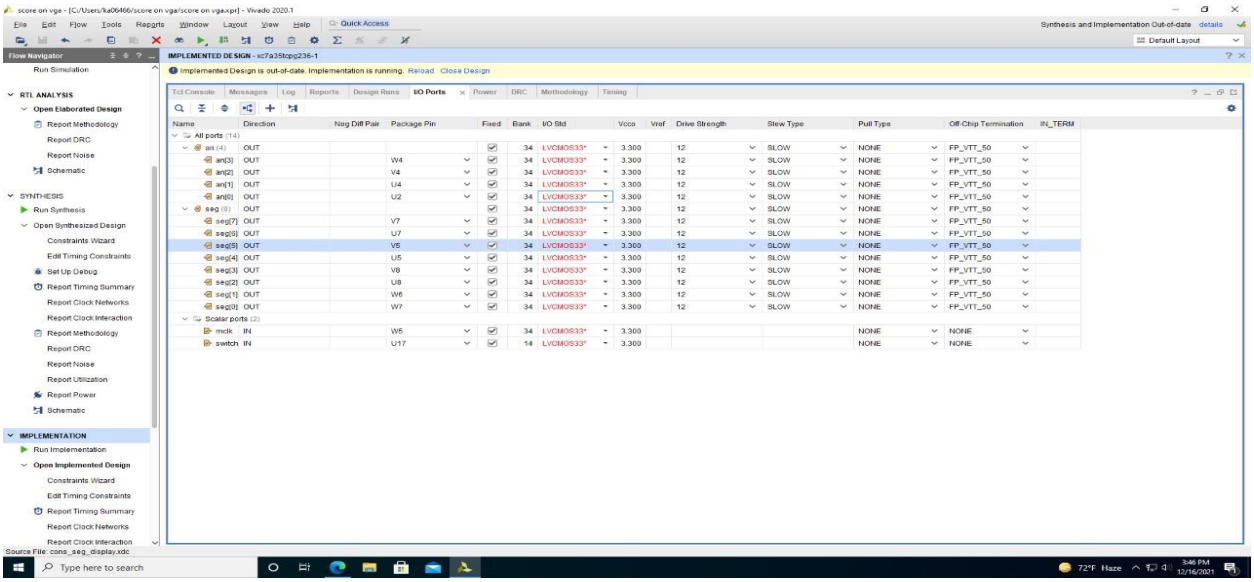
The example counter is called in the top module which transfer the count to the bcd converter. The bcd converter converts the count in binary to bcd and send back the converted count to the top module.

Then, the top module passes the bcd count to the multi segment driver by instantiating the multi segment module which instantiates a single seven segment decoder which decodes the converted count to segment code specifically for different anodes aka particular seven segments.

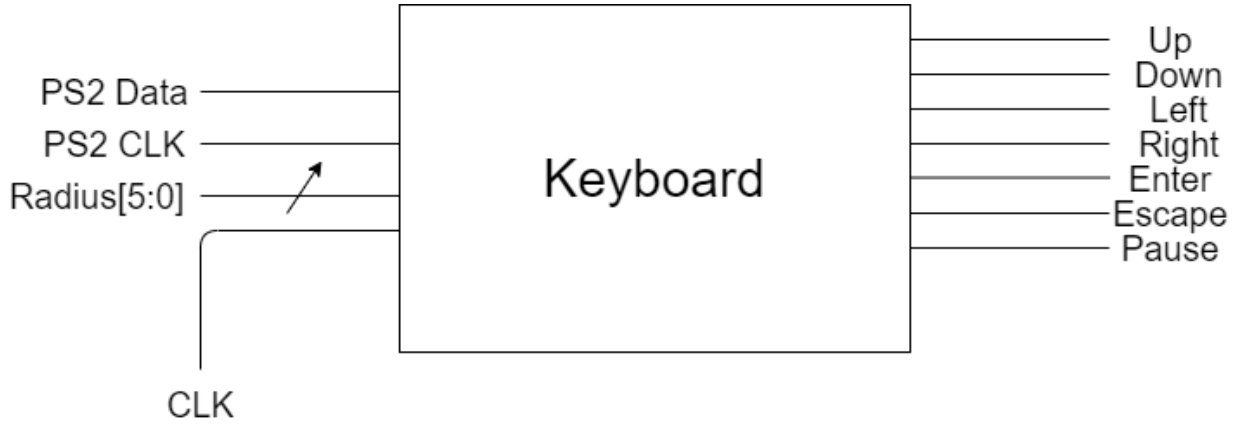
Initially all the four seven segments are set at zero and then the count start from the right most when it gets to 9, the left one increments to 1 and so on.

Constraints:

Constraints for timer outputs are:



Input Block (Keyboard F.S.M)



This has 4 inputs:

- FPGA Clock
- PS2 CLK
- PS2 DATA
- Radius

The code first reduces the frequency to 250 time, lesser than the FPGA frequency which is 100 MHz. It then checks that if all the 8-bits are received or not. It does parity bit checker. We designed the code to use arrow keys as input and output/high if any one of the arrow keys is pressed.

If the key pressed is left, 8-bit code would be passed, with start bit, end bit and a parity bit.

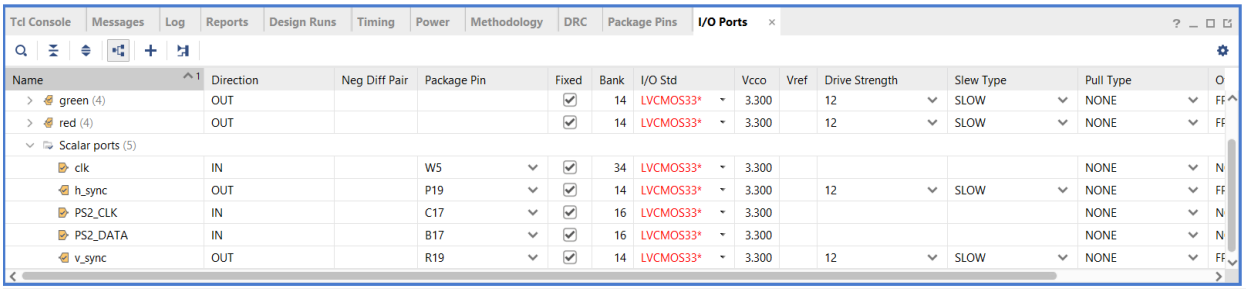
In the end, we added if/else conditions which output a high if a specific key is pressed.

The outputs for the F.S.M are:

- Up
- Down
- Left
- Right
- Escape
- Enter
- Pause

The keyboard module has been further modified to check whether our current radius is 10 which means the game has been lost and a pause signal will be sent to the GameState module to pause the game.

Constraints:



Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	O
green (4)	OUT			☑	14	LVCMOS33*	3.300		12	SLOW	NONE	FF
red (4)	OUT			☑	14	LVCMOS33*	3.300		12	SLOW	NONE	FF
Scalar ports (5)												
clk	IN		W5	☑	34	LVCMOS33*	3.300				NONE	N
h_sync	OUT		P19	☑	14	LVCMOS33*	3.300		12	SLOW	NONE	FF
PS2_CLK	IN		C17	☑	16	LVCMOS33*	3.300				NONE	N
PS2_DATA	IN		B17	☑	16	LVCMOS33*	3.300				NONE	N
v_sync	OUT		R19	☑	14	LVCMOS33*	3.300		12	SLOW	NONE	FF

Clock Divider (0.3):

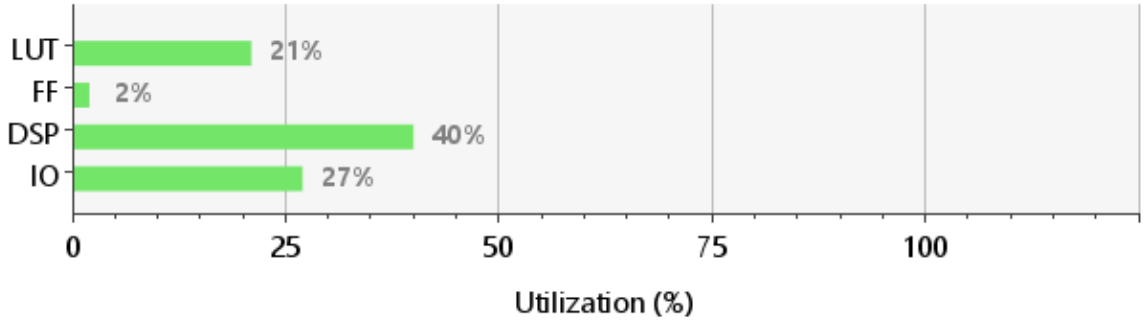
Using the formula div value = input/2xdesired – 1 we get the following div value

Div Value = 833332

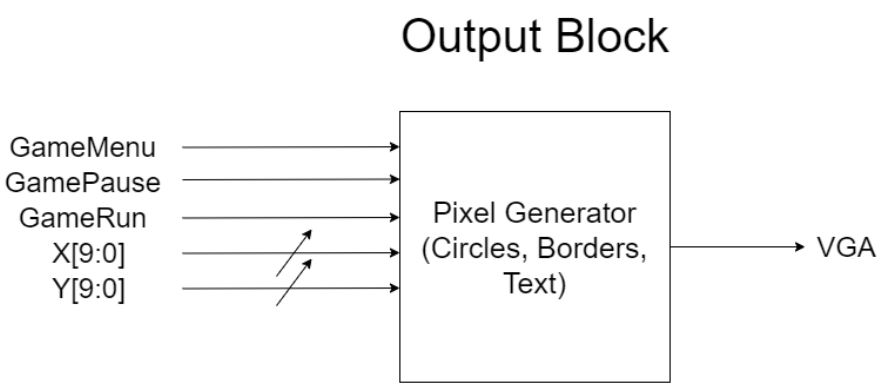
The 26-bit count register increments up to this value to update the clock every 0.3 seconds.

The other clock divider used is the one that slows the clock to 25MHZ.

Utilization Report:



Output Block:



Resolution: 640x480

The output block takes the signals of the current GameState and x and y Centre coordinates for all the circles being displayed.

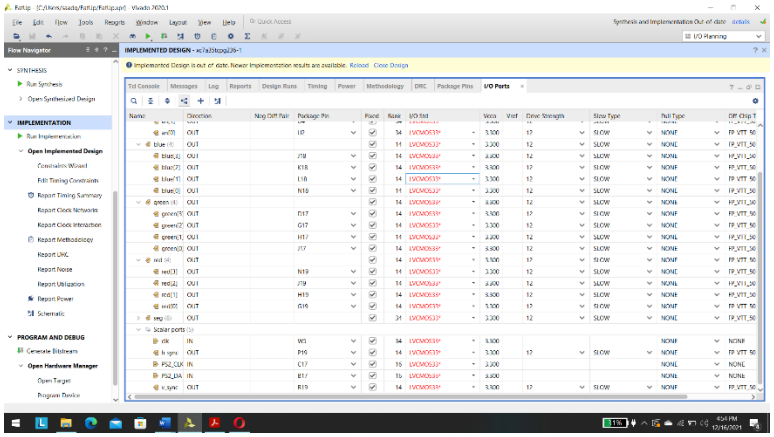
For drawing circles, the following equation is used:

```
video_on?((pixel_x - x)*(pixel_x - x) + (pixel_y - y) * (pixel_y - y) <= r * r ? 4'hF:4'h0):(4'h0);
```

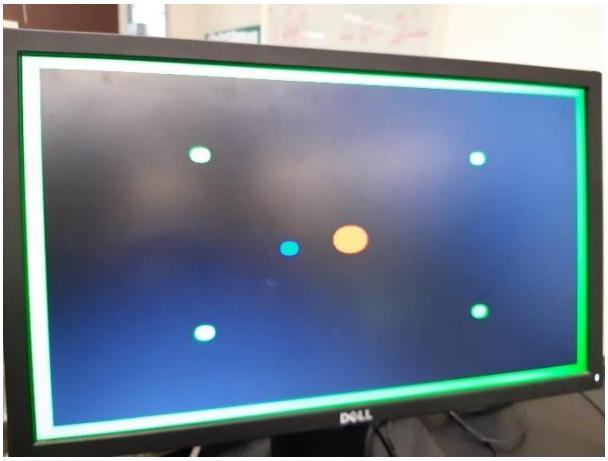
For drawing blue enemy circles, an or gate is added for all equations of circles on screen.

```
blue <= (eq1) | (eq2) | (e3) | e4)
```

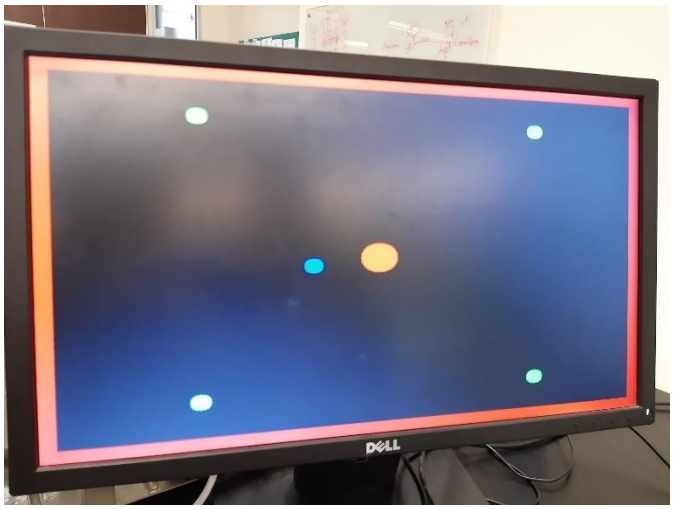
Constraints:



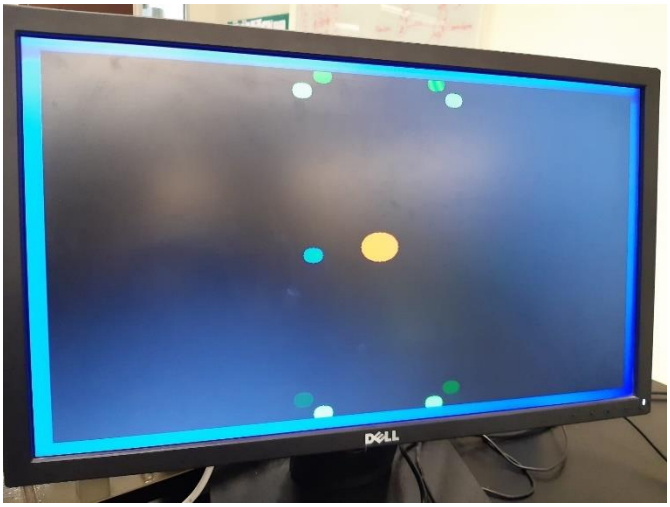
For GameMenu green borders are displayed:



For GamePause red borders are displayed:



For GameRun blue borders are displayed:



References:

https://www.instructables.com/PS2-Keyboard-for-FPGA/

Lab 11 DLD

https://www.youtube.com/watch?v=Bh_1WQWI2Q0&feature=youtu.be

Video Demo:

https://www.youtube.com/watch?v=46foIAfIwKM&t=37s&ab_channel=KhuzaimaAliKhan