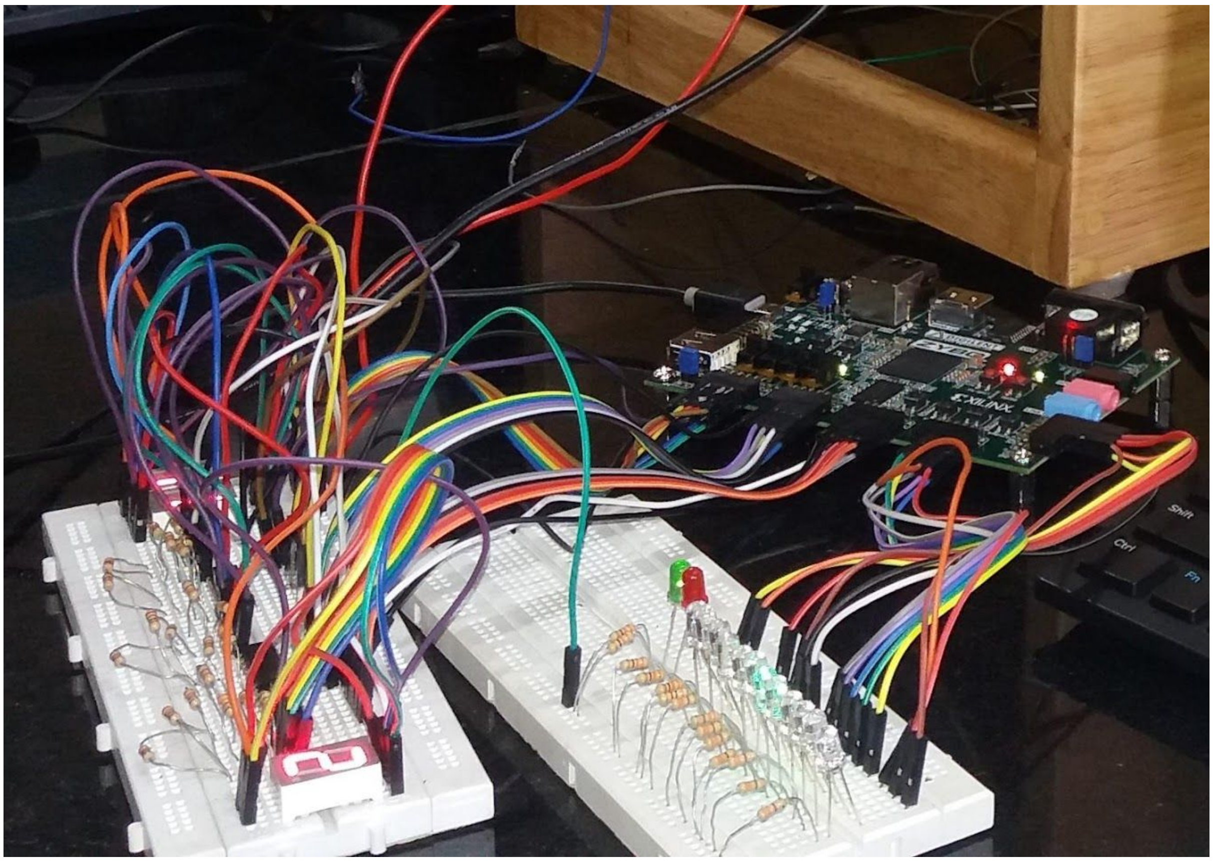
MINI PROJECT REPORT

EE LAB (EE 2702) - III Semester



MONO PONG

Himanshu Rai (111601032)
Ayush Jain (121601005)

PROJECT ABSTRACT

PONG

Pong was one of the first computer games to be created. This simple "tennis like" game featured two paddles and a ball, the goal being to defeat your opponent by being the first one to gain 10 points. A player gets a point once the opponent misses a ball. The game could be played with two human players, or one player against a computer controlled paddle. The game was originally developed by Allan Alcorn and released in 1972 by Atari corporations. Soon, Pong became a huge success, and became the first commercially successful game, On 1975, Atari release a home edition of Pong (the first version was played on Arcade machines) which sold 150,000 units. Today, the Pong Game is considered to be the game which started the videogames industry, as it proved that the video games market can produce significant revenues.

MONO PONG

Mono Pong is a variant of the original Pong game in which instead of moving in the two dimensional space, the ball moves in only one dimension (along a straight line) and both the paddles are controlled by a single player. If the player hits the ball with the paddle, it bounces to the other side, increasing his score by one and instead if he missed, the game gets over.

THE PROJECT

Our project consist of designing a Mono Pong game using Verilog HDL with the help of an FPGA for the implementation. The game would be single player controlled and will display the score of the user as he progresses. It has start and stop buttons to operate the game and two buttons for playing (acts as paddles for hitting the ball). The display on the game also shows the level as user progress through the game. With each increase in level the speed of the ball increases, making the game harder to play. The level itself increases by one every time the user score increases by ten.

CONTRIBUTIONS

The project was made with equal contributions by the members. The idea of the game was given by Ayush. Then after finalising it (the topic), we together worked out the logic of the implementation to be done. Then we wrote the HDL and then debugged (the hardest part because the game being dynamic we can’t use simulation to check for errors and hence took the longest time). Finally, this report was made by Himanshu with suggested edits from Ayush.

INTRODUCTION

The project involves three separate modules, **hex_to_sev_seg**, **clock_generator** and **Mono_Pong**, The main module is **Mono_Pong**. It contains the logic to display the motion of the ball and to update the level and score as the game progresses. **clock_generator** as the name suggests is the module for the implementation of clock, which controls speed of the ball and also synchronises various tasks throughout the program. Finally, the last module **hex_to_sev_seg** is basically a hexadecimal to seven segment decoder. It accepts a hexadecimal number as input and gives output which can be given to a seven-segment display to display the corresponding number.

The outer interface of the circuit consists of taking input from the user from two dedicated push buttons. One of the push buttons act as a right paddle for hitting the ball and the second push button acts as the left paddle. The user has to press the appropriate key at the right moment to hit the ball and to send it to other side. Successful hit scores a point for the player, whereas missing the ball ends the game.

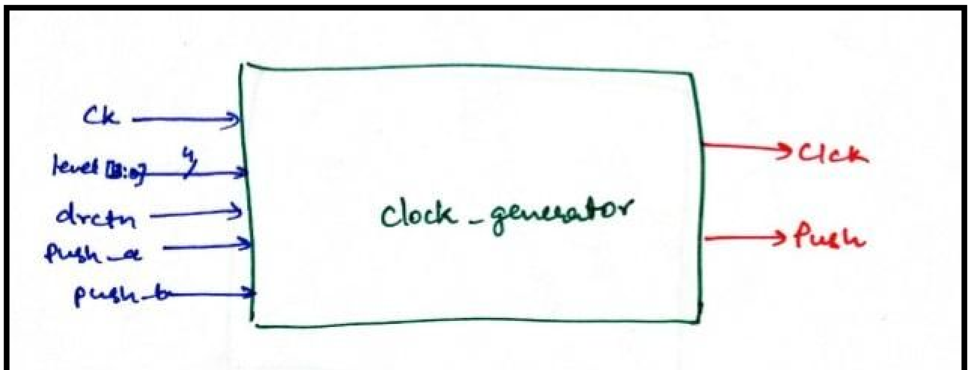
Apart from these two buttons, there are two other pushbuttons to start and stop the game. Pressing the start button, starts the game and turns on a green LED as an indicator for the game in progress. When player gets out or the game is not being played a red indicator LED keeps glowing.

Player can track his progress through the game with the help of three seven segment displays-one if which shows the current level and the other two shows the current score of the player. We have used hexadecimal to seven segment decoders, because it can display decimal digits as usual, as well as additional hexadecimal digits. To display the score the ones digit is shown as normal decimal but the tens position is displayed as hexadecimal. So, using only two seven segment displays we could display user score upto 159, instead 99. If the player gets out, the score indicator keeps on displaying the score at which he got out, until he starts a new game.

The detailed implementation and block diagrams of various modules is explained below.

MODULE : clock_generator

Module **clock_generator** generates the clock used in the program to synchronise the various activities. The block diagram of the module is shown below.



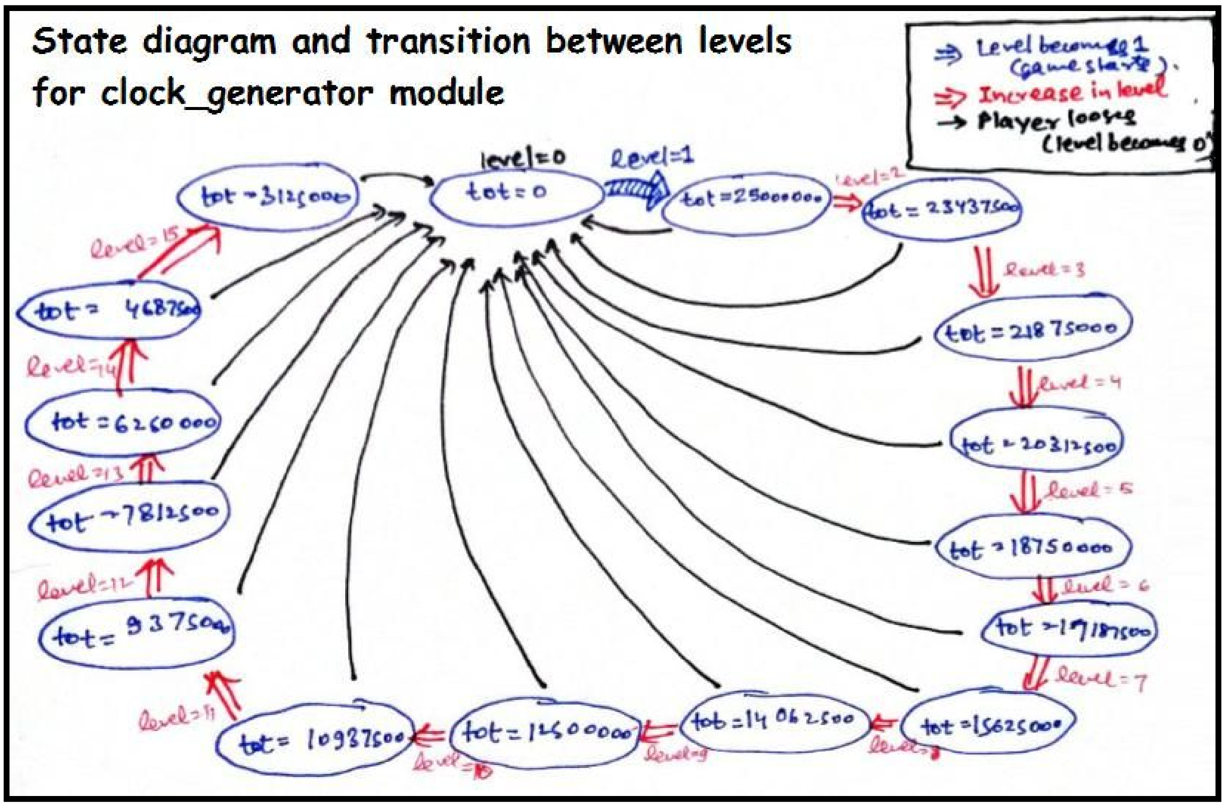
Block Diagram of clock_generator module

It accepts input from the Zybo clock, through the main module. The other inputs are the current level, the direction in which the ball is currently moving and the user inputs (corresponding to swinging of paddles to hit the ball). And as output it gives a clock whose frequency is modified according to the level. The other output denotes whether user was able to hit the ball (by pressing appropriate push button according to the direction of the motion of the ball) when the ball reaches at one of the ends.

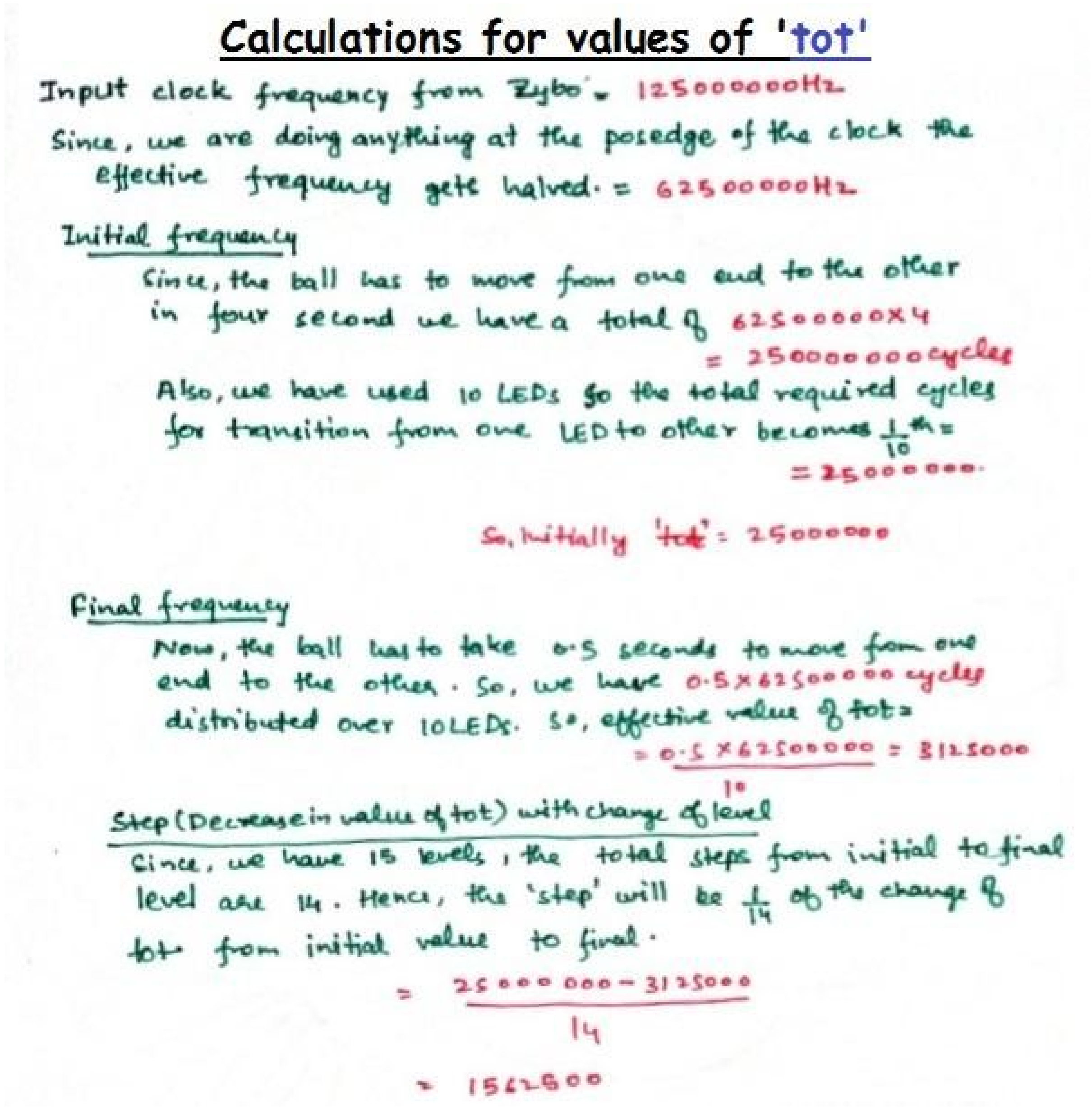
IMPLEMENTING THE CLOCK

The clock is implemented with the help of a counter **count** and which counts upto a certain value (the counter gets incremented by one at each positive edge of the input Zybo clock) and when this value becomes equal to or greater than **tot** (which is a register storing some predefined values and changes values asynchronously everytime the **level** input changes) the output clock changes its state from HIGH to LOW or vice-versa and the counter **count** is again synchronously set to zero. The **tot** register is initially initialised to 0 (when the **level** is zero), so the output clock frequency is half the input frequency from the zybo clock (because the level of the output clock changes only at the posedge of the Zybo clock). When the player starts the game (the input **level** becomes 1) the value of **tot** is updated and made 25000000, so that the ball takes 4 seconds to reach from one end to the other. With each increase in **level** the value of **tot** is decreased by 1562500, so that when the player

reaches 15th level the ball takes 0.5 seconds to move from one end to the other. So, with each level the speed of the ball increases uniformly. Now when the player misses the ball, the input **level** becomes 0, so that the output clock frequency again becomes half the frequency of the input Zybo clock.
When the game is not being played, the **level** is kept 0 and the high output clock frequency allow us to take input from the player to start the game the moment he presses the **start** button. Also, if the player doesn't get out, and keeps on playing, the level won't increase after the 15th **level** (However, the speed increases too much that the player ever won't reach **level** 15).
The following figures shows the calculations for the value of **tot** and the state diagram for the module.



Since, for each state there are only two possibilities either **level** increases or becomes zero. So, each state has two lines going out from it. However, the value of **tot** doesn't change after **level** 15 so, that has only one line going out from it.



Calculations for values of 'tot'

Input clock frequency from Zybo = 125000000Hz

Since, we are doing anything at the posedge of the clock the effective frequency gets halved. = 62500000Hz

Initial frequency

Since, the ball has to move from one end to the other in four second we have a total of 62500000x4 = 250000000cycles

Also, we have used 10 LEDs so the total required cycles for transition from one LED to other becomes $\frac{1}{10}$th = = 25000000.

So, Initially 'tot' = 25000000

Final frequency

Now, the ball has to take 0.5 seconds to move from one end to the other. So, we have 0.5x62500000 cycles distributed over 10LEDs. So, effective value of tot = $= \frac{0.5 \times 62500000}{10} = 3125000$

Step (Decrease in value of tot) with change of level

Since, we have 15 levels, the total steps from initial to final level are 14. Hence, the 'step' will be $\frac{1}{14}$ of the change of tot from initial value to final.

$$= \frac{25000000 - 3125000}{14}$$

$$= 1562500$$

IMPLEMENTATION FOR DETECTING THE USER INPUT

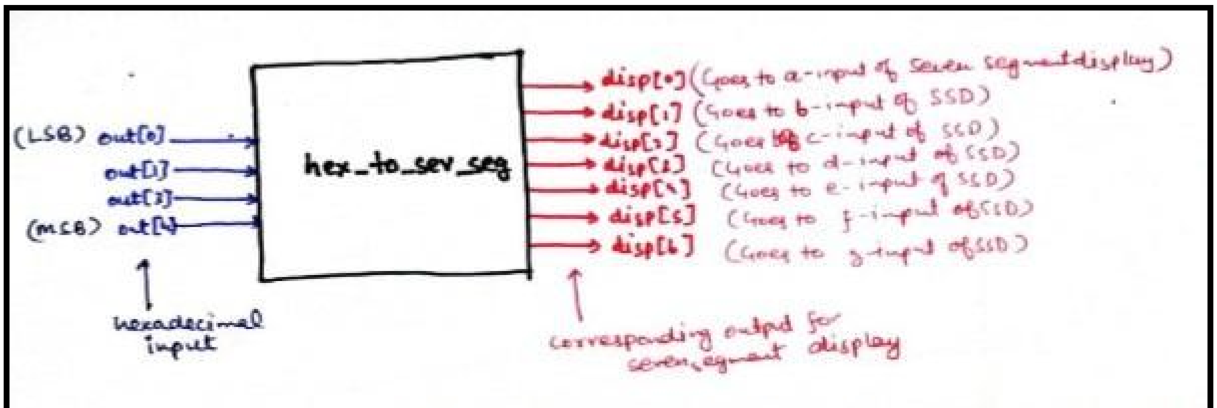
For the purpose of detecting user input we have used a counter **push_count,** which keeps track of the position of ball. Since, only ten possible positions of the ball are there (we have used ten LEDs), its a four bit counter and takes values from 1 to 9 (we have counted upto 9 instead of counting upto 10) . When the ball is at one of the ends its position **(push_count)** is 1, and when it is not at either of the ends, its position **(push_count)** is relative to the previous end that it visited. So, we detect whether player has pressed the pushbutton corresponding to the end (ball is at one of the ends if **push_count** is 1**)** at which the ball is currently present**.** Whenever an input is detected during the time the ball is at position 1 (**push_count** is 1**),** at the time of transition of **push_count** from 1 to 2, the value of the output **push** is made 1**.** This value is detected in the **Mono_Pong** module as user hitting the ball with the paddle and he continues to play. However, if no input from player is detected (or the user has tried to cheat by continuously pressing the input button as explained below), the output **push** is made 0. As a result of which no input is detected in the **Mono_Pong** module, and user gets out. At other transitions of **push_count** the value of **push** is made 0, so that no previous input gets double counted as next input. The idea of which of the two pushbutton is to be pressed can be determined from **drctn** input (the direction of motion of the ball).

IMPLEMENTATION FOR PREVENTING CONTINUOUS INPUTS

To check whether player has been continuously pressing the inputs, we check the player's input (from corresponding pushbutton according to the direction of the motion of the ball) when **push_count** changes from 9 to 1. If user has been continuously pressing the push button, **check_push_count** is set HIGH. And the player's input is no longer accepted (even if he presses the button at right time) and he gets out.

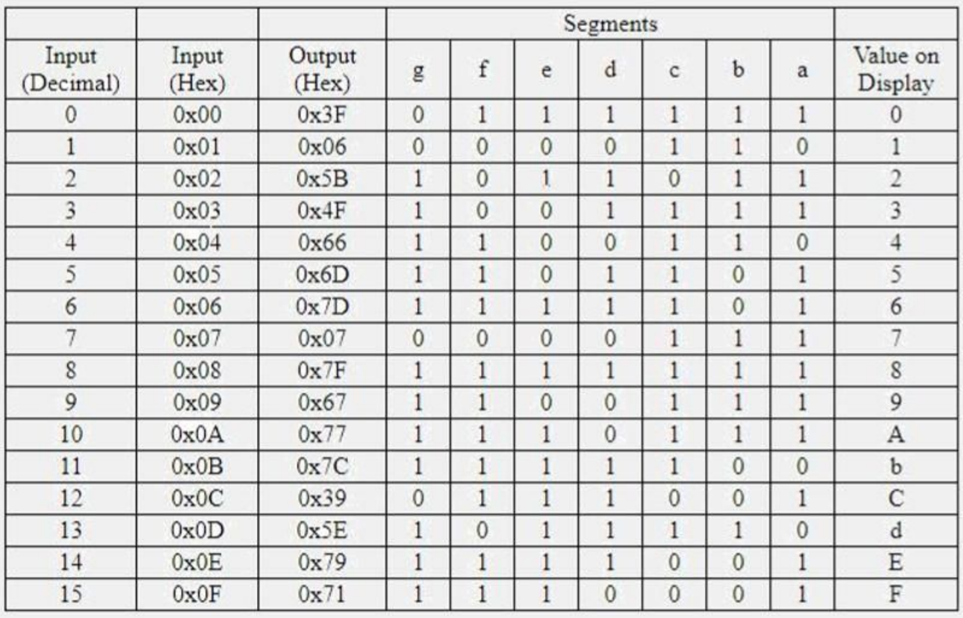
MODULE : hex_to_sev_seg

This module basically acts as hexadecimal to seven segment decoder and is used to display the score of the player and current level as the game progresses.
The block diagram for the module is as shown below.


hex_to_sev_seg
(LSB) out[0]
out[1]
out[2]
(MSB) out[4]
hexadecimal input
disp[0] (Goes to a-input of seven segment display)
disp[1] (Goes to b-input of SSD)
disp[2] (Goes to c-input of SSD)
disp[3] (Goes to d-input of SSD)
disp[4] (Goes to e-input of SSD)
disp[5] (Goes to f-input of SSD)
disp[6] (Goes to g-input of SSD)
corresponding output for seven segment display

Block diagram for hex_to_sev_seg module

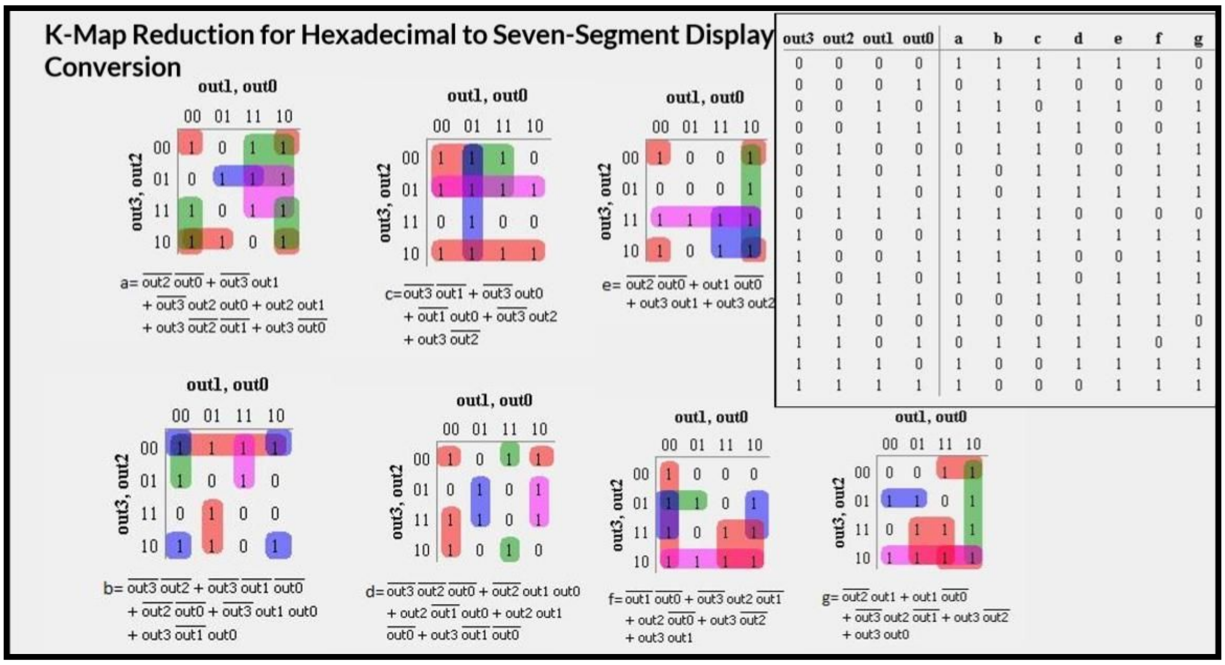
The following table shows shows the truth table for the inputs and outputs.



			Segments							
Input (Decimal)	Input (Hex)	Output (Hex)	g	f	e	d	c	b	a	Value on Display
0	0x00	0x3F	0	1	1	1	1	1	1	0
1	0x01	0x06	0	0	0	0	1	1	0	1
2	0x02	0x5B	1	0	1	1	0	1	1	2
3	0x03	0x4F	1	0	0	1	1	1	1	3
4	0x04	0x66	1	1	0	0	1	1	0	4
5	0x05	0x6D	1	1	0	1	1	0	1	5
6	0x06	0x7D	1	1	1	1	1	0	1	6
7	0x07	0x07	0	0	0	0	1	1	1	7
8	0x08	0x7F	1	1	1	1	1	1	1	8
9	0x09	0x67	1	1	0	0	1	1	1	9
10	0x0A	0x77	1	1	1	0	1	1	1	A
11	0x0B	0x7C	1	1	1	1	1	0	0	b
12	0x0C	0x39	0	1	1	1	0	0	1	C
13	0x0D	0x5E	1	0	1	1	1	1	0	d
14	0x0E	0x79	1	1	1	1	0	0	1	E
15	0x0F	0x71	1	1	1	0	0	0	1	F

Truth table for Hexadecimal to Seven-Segment Display Conversion

The following figure shows the Karnaugh Map reductions and the expressions for the outputs in terms of inputs.


K-Map Reduction for Hexadecimal to Seven-Segment Display Conversion

out3	out2	out1	out0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

$a = \overline{out2}\,\overline{out0} + \overline{out3}\,out1 + \overline{out3}\,out2\,out0 + out2\,out1 + out3\,\overline{out2}\,\overline{out1} + out3\,\overline{out0}$

$c = \overline{out3}\,\overline{out1} + \overline{out3}\,out0 + \overline{out1}\,out0 + \overline{out3}\,out2 + out3\,\overline{out2}$

$e = \overline{out2}\,\overline{out0} + out1\,\overline{out0} + out3\,out1 + out3\,out2$

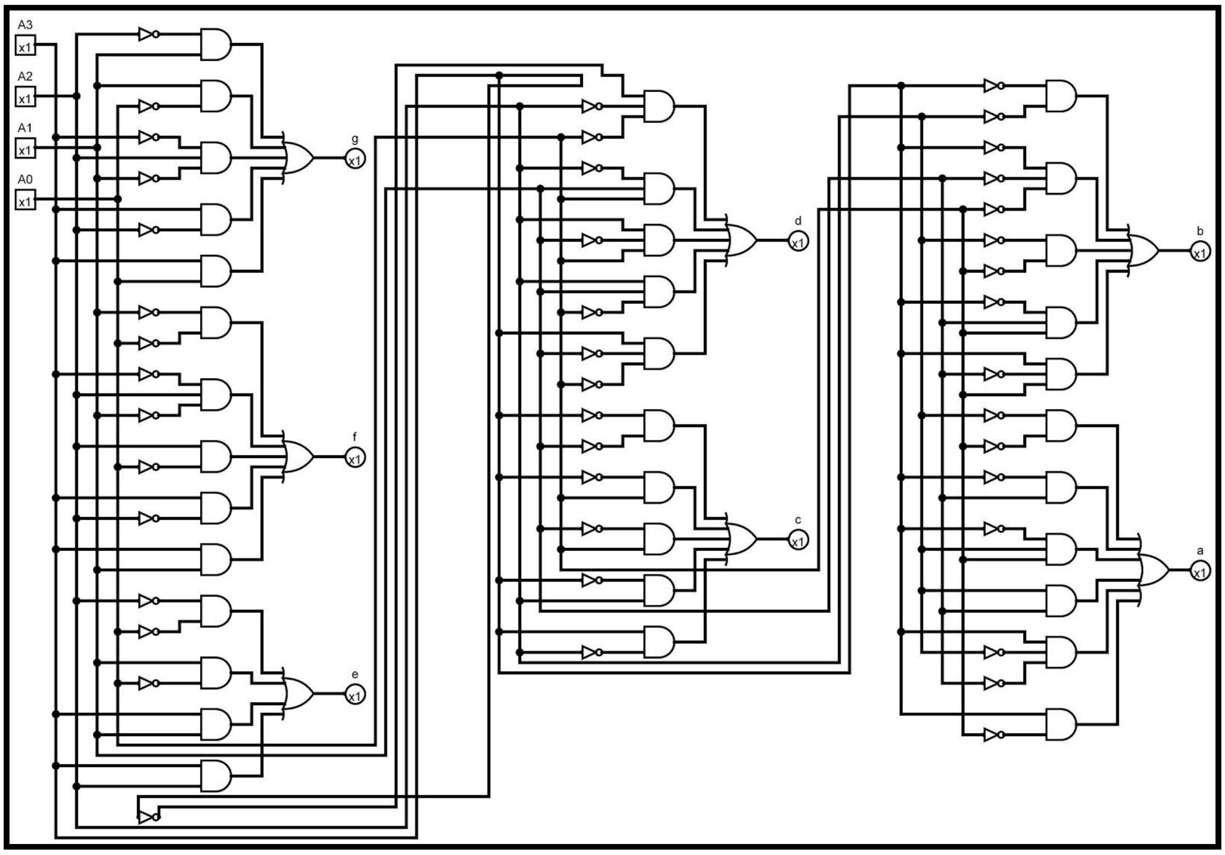
$b = \overline{out3}\,\overline{out2} + \overline{out3}\,\overline{out1}\,\overline{out0} + \overline{out2}\,\overline{out0} + \overline{out3}\,out1\,out0 + out3\,\overline{out1}\,out0$

$d = \overline{out3}\,\overline{out2}\,\overline{out0} + \overline{out2}\,out1\,out0 + out2\,\overline{out1}\,out0 + out2\,out1\,\overline{out0} + out3\,\overline{out1}\,\overline{out0}$

$f = \overline{out1}\,\overline{out0} + \overline{out3}\,out2\,\overline{out1} + out2\,\overline{out0} + out3\,\overline{out2} + out3\,out1$

$g = \overline{out2}\,out1 + out1\,\overline{out0} + \overline{out3}\,out2\,\overline{out1} + out3\,\overline{out2} + out3\,out0$

Following figure shows the combinational circuit for the module.

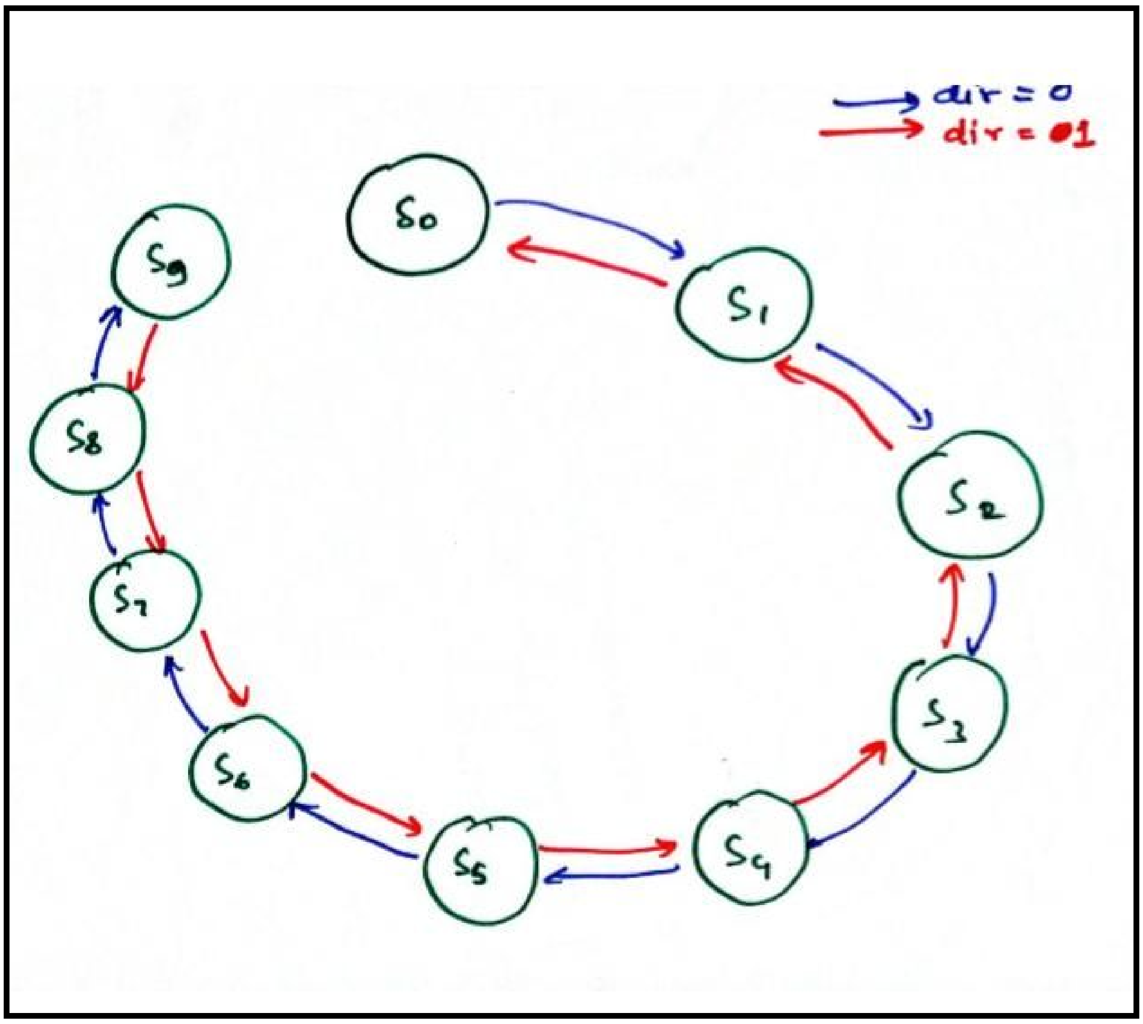


MODULE : Mono_Pong

This is the main module of the project. It contains the logic for starting or stopping the game whenever player pushes the **start** or **stop** button. The function of displaying the motion of the ball is also implemented within this module. Beside it also has counters for counting player's score and level. It displays these informations with the help of **hex_to_sev_seg** module which has been already described.

IMPLEMENTING THE MOTION OF THE BALL

The motion of the ball is internally implemented as ten- bit ring counter, which changes its direction (shift left and shift right) after every 10 shifts or counts. The value stored in the counter consists of single 1 and all 0s. The state diagram of the implementation is as shown below. Note that the subscript (following **S** denotes the position with respect to the rightmost bit that currently has 1 stored in it. **dir** decides the direction of motion of the ball (left shift or right shift). (**dir** is not shown in the state part, inside the circle to make the diagram smaller. So instead of having 10 states the implementation has 20 states).



IMPLEMENTATION FOR COUNTING SCORE

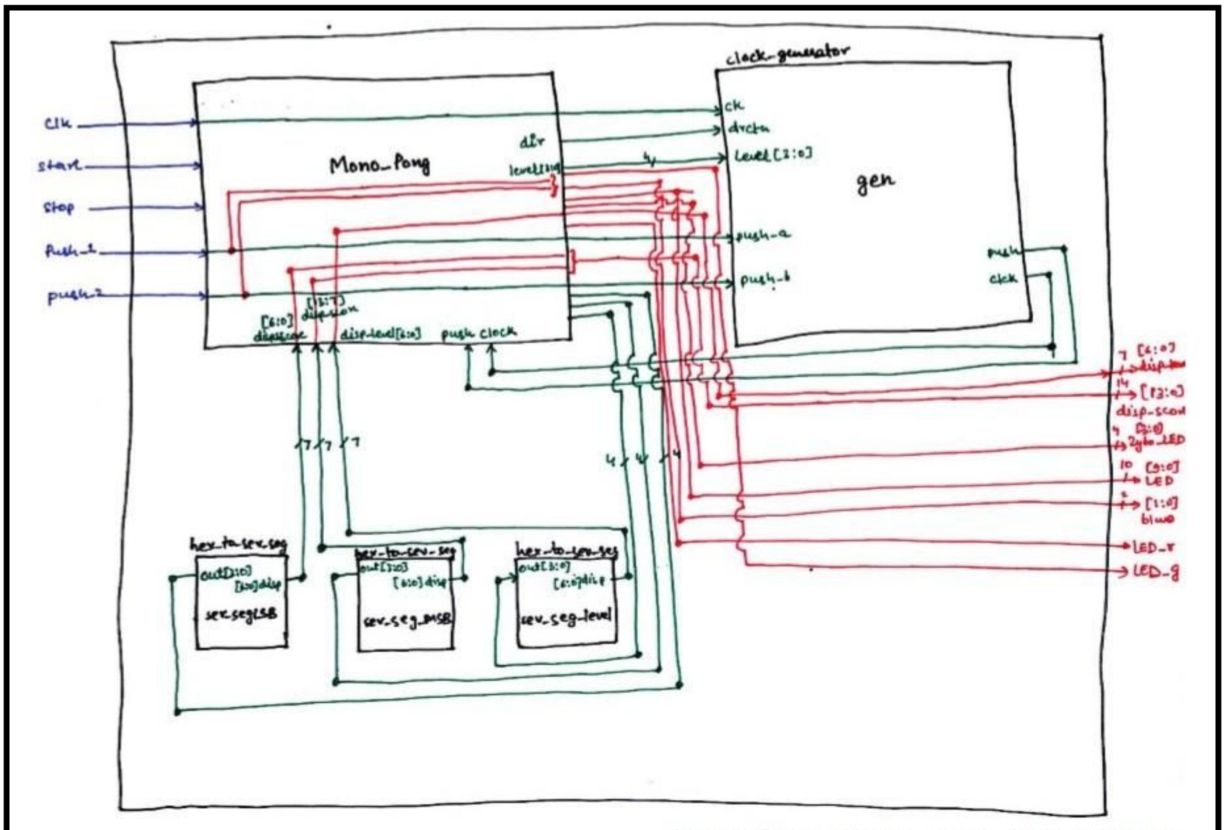
score counter is basically implemented as a 8-bit BCD counter which gets incremented every time **dir** changes. And it is also synchronously set to 0 everytime **start** changes to 1 from 0.

IMPLEMENTATION FOR COUNTING LEVEL

level here serves two functions. First it counts the level of the game. And second it also helps to keep track of whether the game is on (**level** is not zero) or not (**level** is zero).
Before player presses **start** the **level** is 0 (indicator red LED keeps glowing). When player presses **start**, **level** is incremented to 1, and the game starts. When player gets out, the **level** is again synchronously set to 0, indicating game has stopped.
When the game is being played **level** stores the current level. Everytime score increases by 10 (first four bits of user's score becomes 0) the level is incremented by 1. This increases the speed of the ball.

Interconnections among modules

The following figure shows the final interconnections among the modules (each module has been described earlier).



Inter Connections among the modules

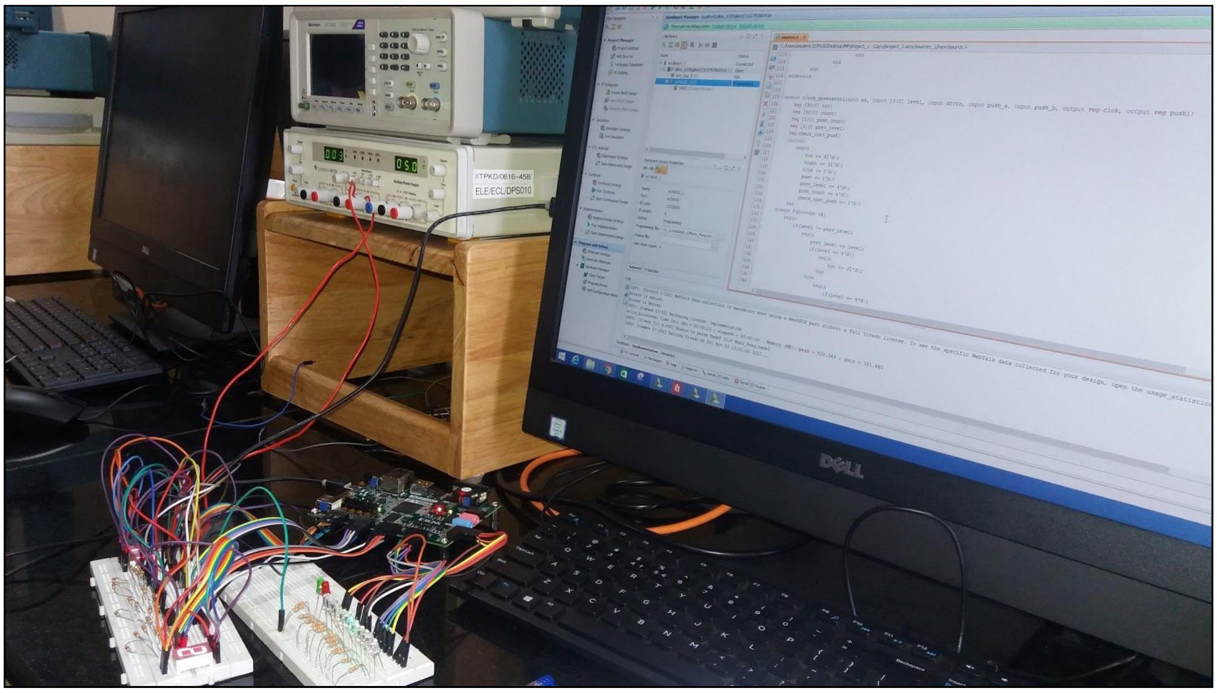
**Note-In order to view the image, click on it.

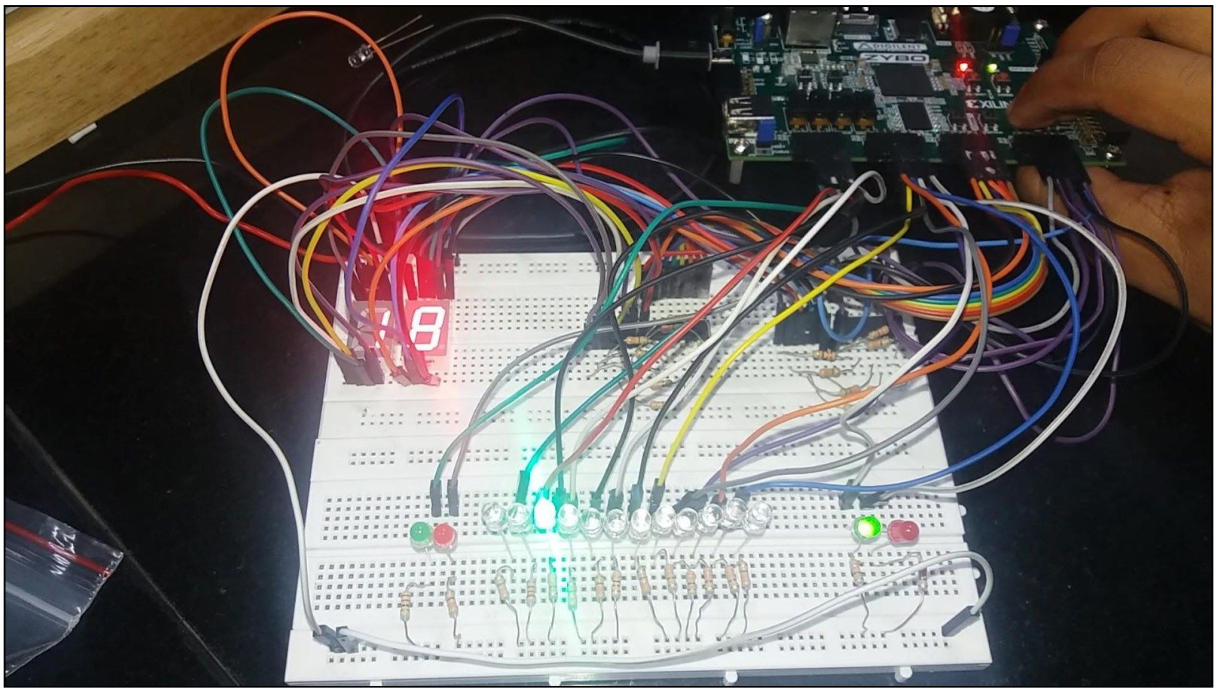
HARDWARES USED

- Digilent Zybo (Zynq-7000 FPGA) Board
- PC with Vivado Design Suite installed on it
- Breadboards
- LEDs (10 white, 1 red, 1 green, 2 blue)
- Seven Segment Displays (2 for score and 1 for level)
- ULN 2003 Driver ICs (one for each seven segment displays)
- Resistances
- Jumper Wires (for connections)
- Voltage Source

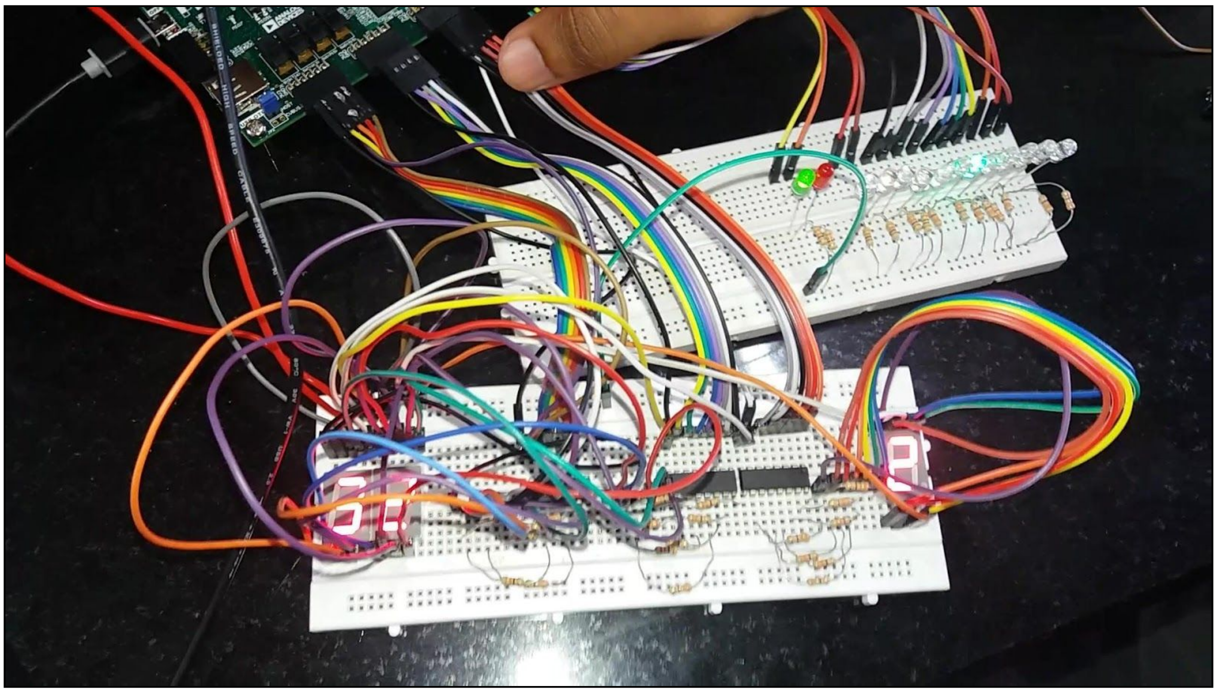
IMPLEMENTATION

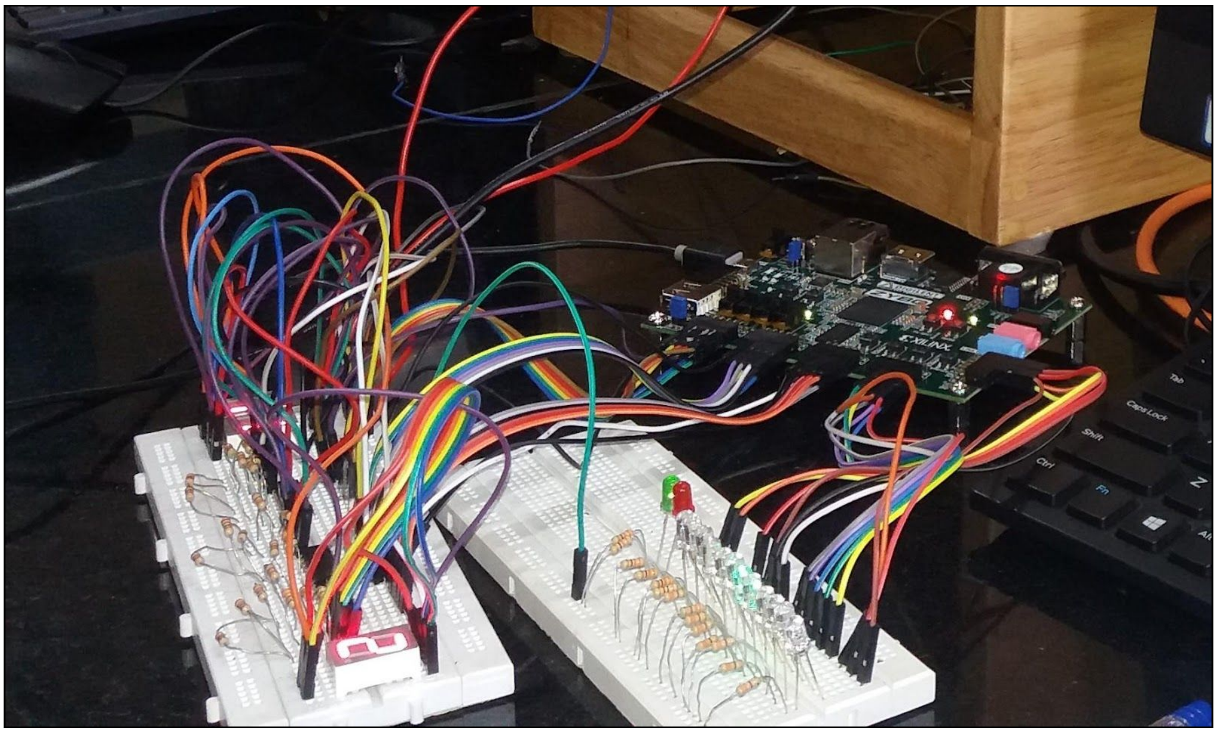
The following snapshots shows the actual implementation of the project.



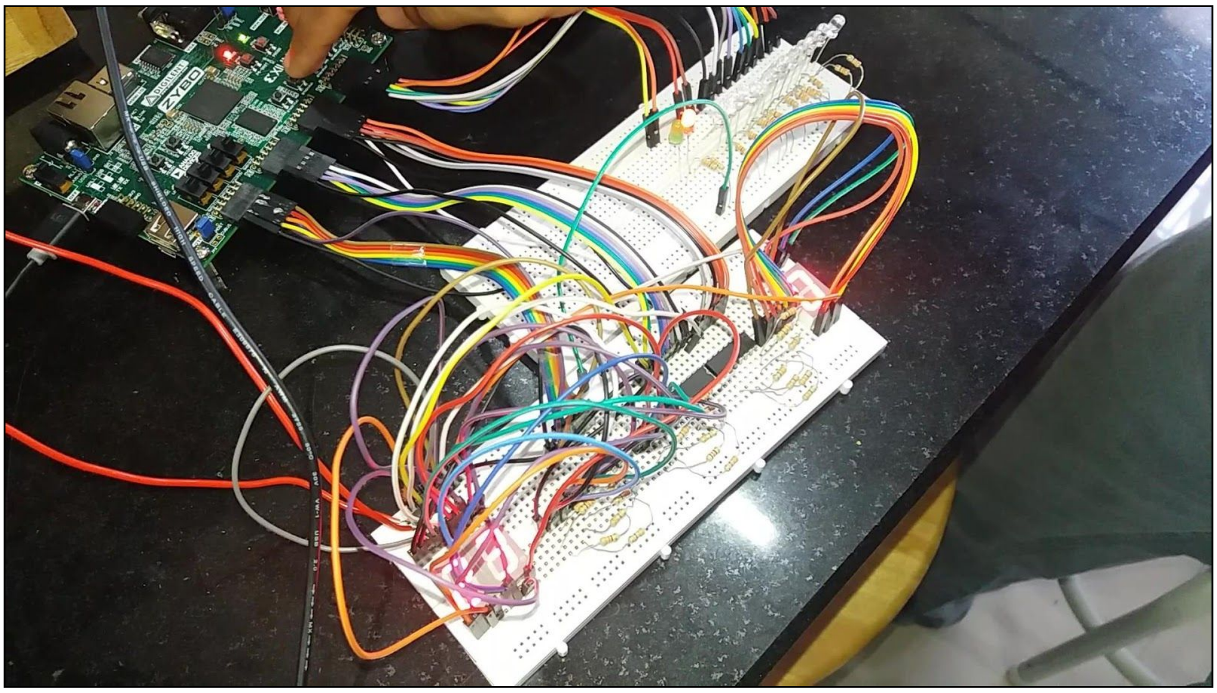


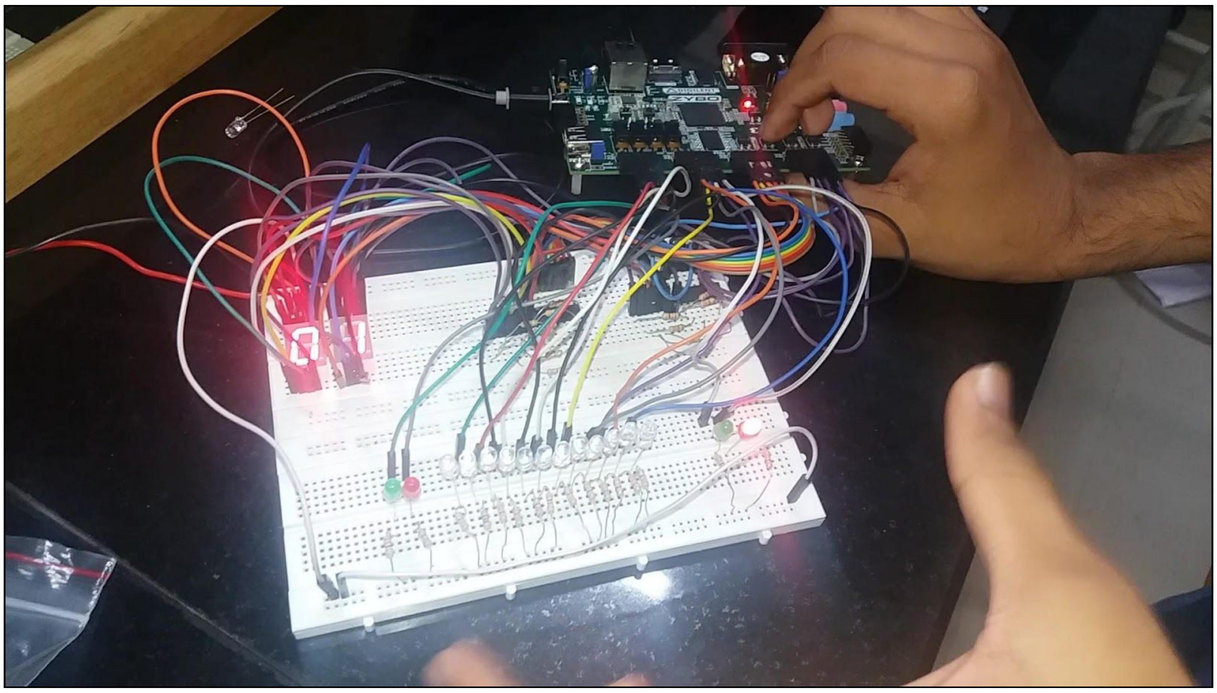
**Note-In order to view a snapshot, click on it.





**Note-In order to view a snapshot, click on it.





**Note-In order to view a snapshot, click on it.

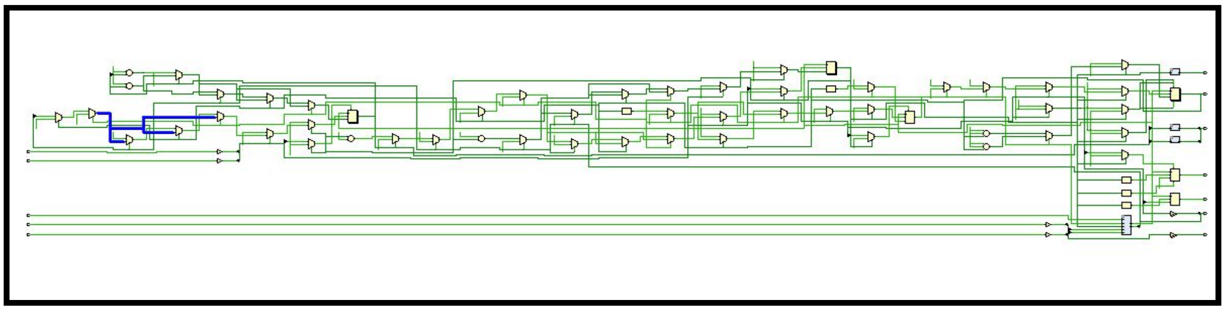
RTL SCHEMATIC

In digital circuit design, **register-transfer level** (**RTL**) is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals.

Register-transfer-level abstraction is used in hardware description languages (HDLs) like Verilog and VHDL to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived. Design at the RTL level is typical practice in modern digital design.

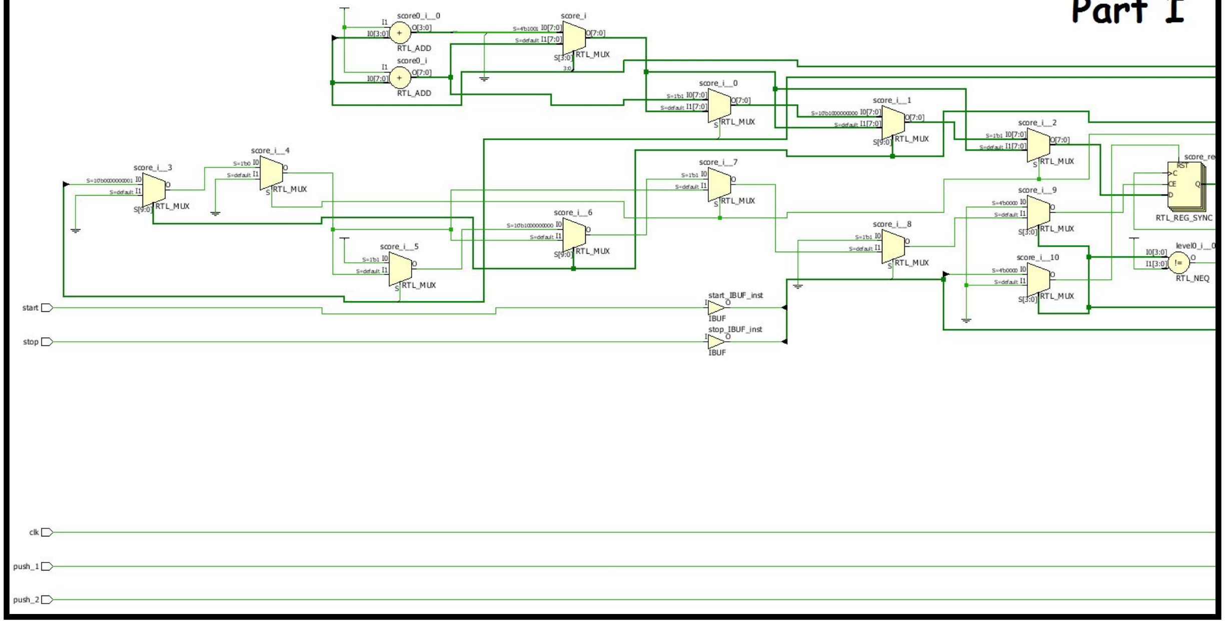
After the HDL synthesis phase of the synthesis process, we can display a schematic representation of our synthesized source file. This schematic shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, AND gates, and OR gates, that are independent of the targeted device. Viewing this schematic may help discover design issues early in the design process.

The RTL Schematic for the implemented source code is shown below.

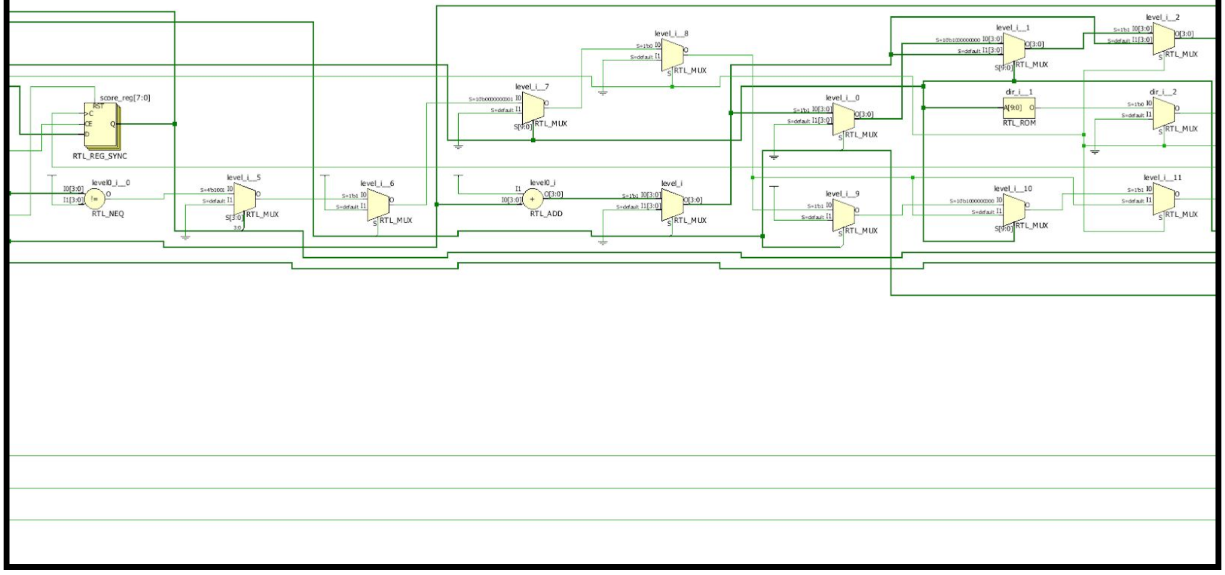


Since nothing is visible in the above schematic the following figures shows the zoomed in version of the above schematic in order from left to right.

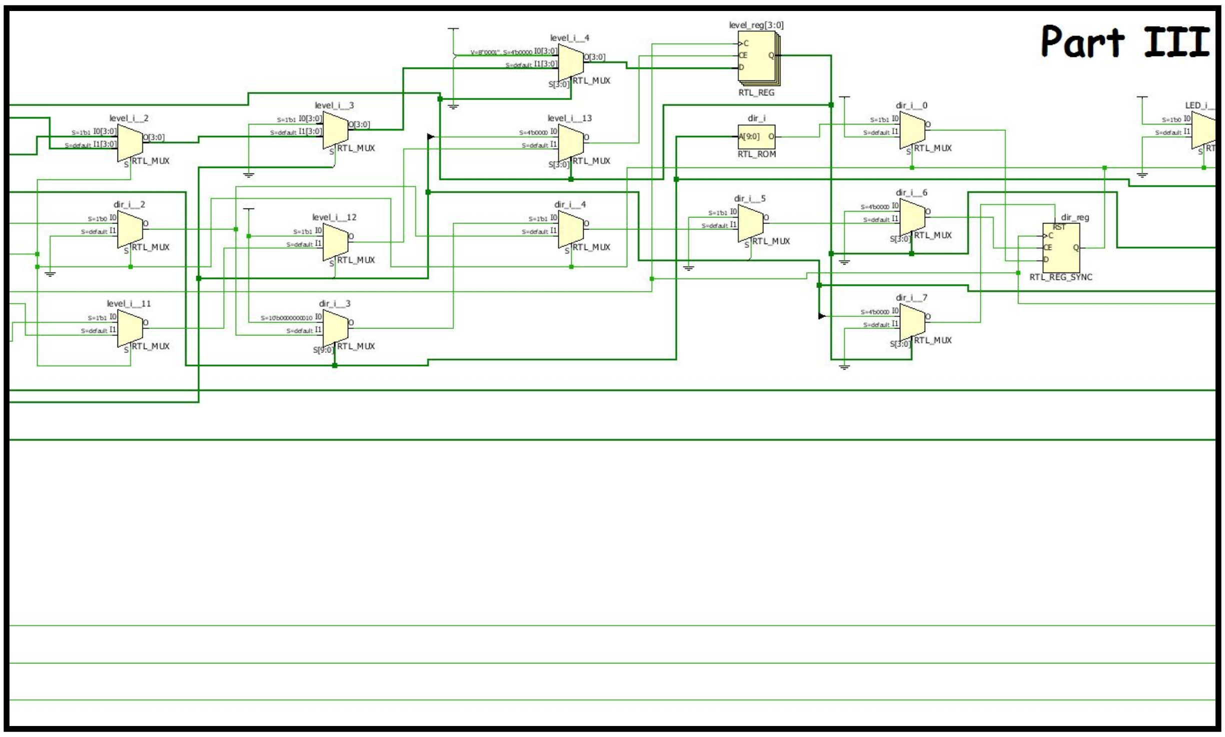
Part I



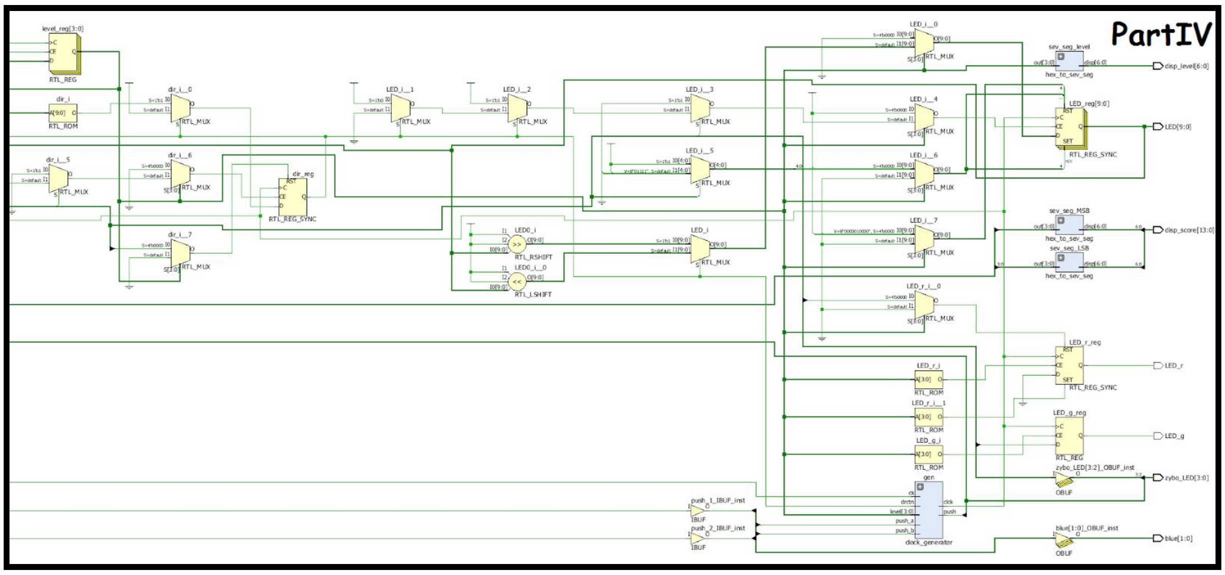
Part II



Part III



PartIV



CONCLUSION

The project was made and displayed in the lab.
We were able to implement all the features in the project that we thought would be feasible. Particularly, we were limited by the number of available ports on the Zybo, which prevented us from using 2-dimensional matrix of LEDs. So, we opted to go for Mono Pong instead of the original Pong game. Also connecting 2-dimensional matrix of LEDs on a breadboard would not have been feasible.
The major challenge was debugging the HDL code. Accepting user input only when he presses input at the right time was the difficult task. Every time we changed small portion of the code we had to go through the long process of running synthesis, running implementation and generating bitstream. Only then we were able to test the changes. This took a lot of time.

REFERENCES

We didn’t require any additional references or help. We used the concepts learnt in the lab over the semester for the implementation of our projects.

LINKS

Click here to view snapshots and videos related to the project.