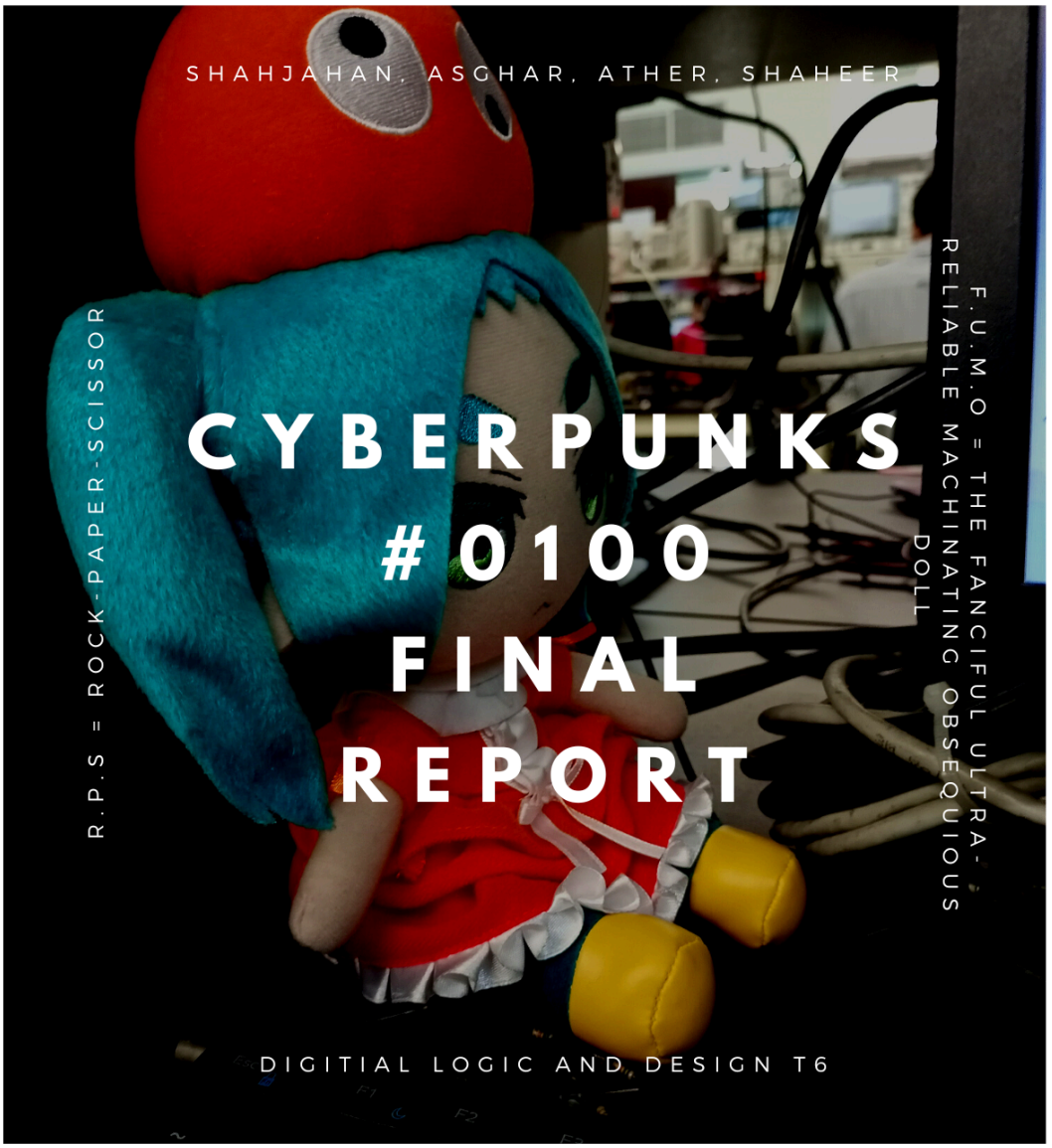
D L D

PROJECT X - THE PLAYFUL R.P.S MACHINE AND F.U.M.O!



DLD FINAL REPORT

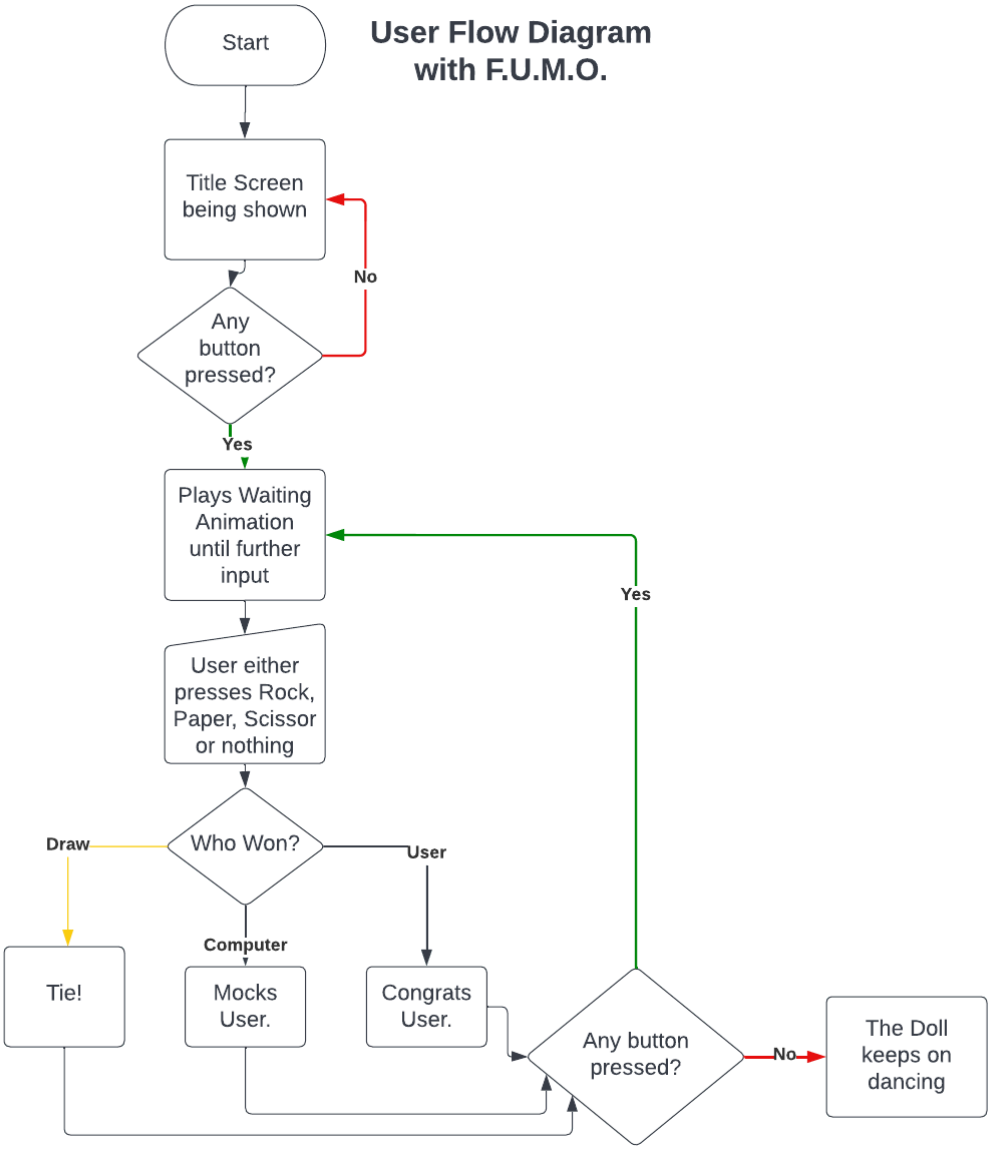
[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

Project Title:

Project X - The Playful R.P.S Machine and F.U.M.O!
*(R.P.S = Rock-Paper-Scissor)
*(F.U.M.O = The Fanciful Ultra-reliable Machinating Obsequious Doll)

User Flow Diagram:

Our goal is to make an arcade machine which you can play rock-paper-scissors against. We also want to make a cute doll come to life by having it represent different things like dancing to celebrate its loss, or sulking when the computer loses to make your day better. [2]



DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

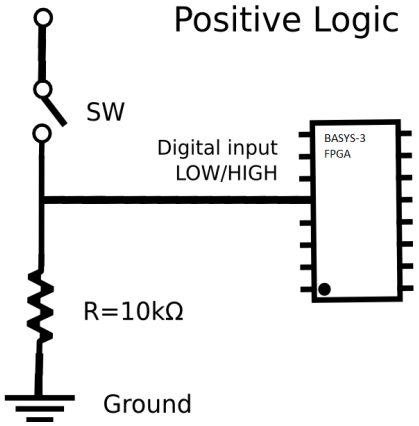
Basic Description of the above flow chart:
Our primary goal is implementing the game with output being a VGA monitor so we will be discussing user-experience with that assumption first and foremost.

When the system is powered up, the player will be shown our game's "title screen" welcoming them. When any button is pressed (except "Reset", as that button is implemented as a good practice) , the FPGA will play a looping animation between "rock", "paper", "scissor" and will also decide on any one of them as well in the background. This waiting animation will last an indefinite period of time until the user decides to give an input with the use of a button representing either Rock, Paper or Scissor. Or perhaps reset it to the title screen using the reset button.

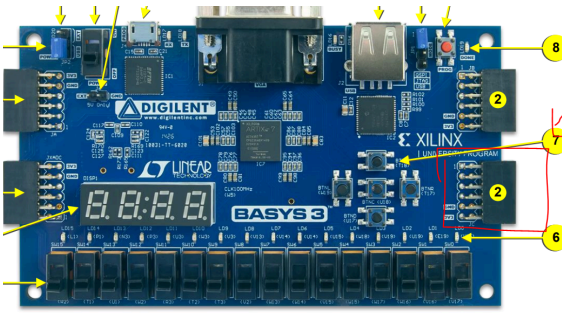
The computer will compare both of these answers and give an output in the form of displaying on a monitor using VGA and having a real-life doll act simultaneously.

Input Block:

We will be taking 4 push buttons as our Input to our FPGA Basys-3. Each Button will be "Rock", "Paper", "Scissors", "Reset". We plan to use pull-down resistors to implement a positive logic circuit to provide input signal to our FPGA.

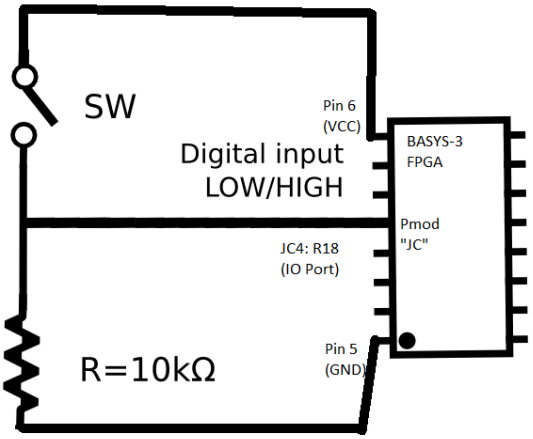


To implement this circuit onto our FPGA, we will be using "JC" Pmod to give an input to our FPGA.



We plan to use a father-father package-pin and take 1 VCC and 1 GND to power our

breadboard power rail, and then implement our above logic there. The Logic Diagram for one "input is attached" to give an understanding.



We have also tested it practically, we programmed our FPGA in a way that it would light an LED when it would detect the input signal being provided at the R18 port of Pmod-JC.

Verilog Design Code:

```
module Input_project(
    input A,
    output B
    );

assign B = A;
endmodule
```

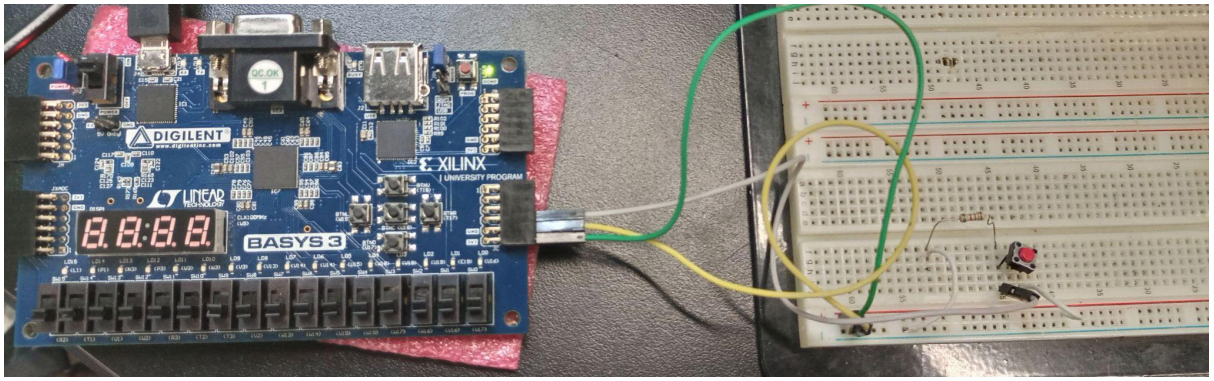
Constraint File:

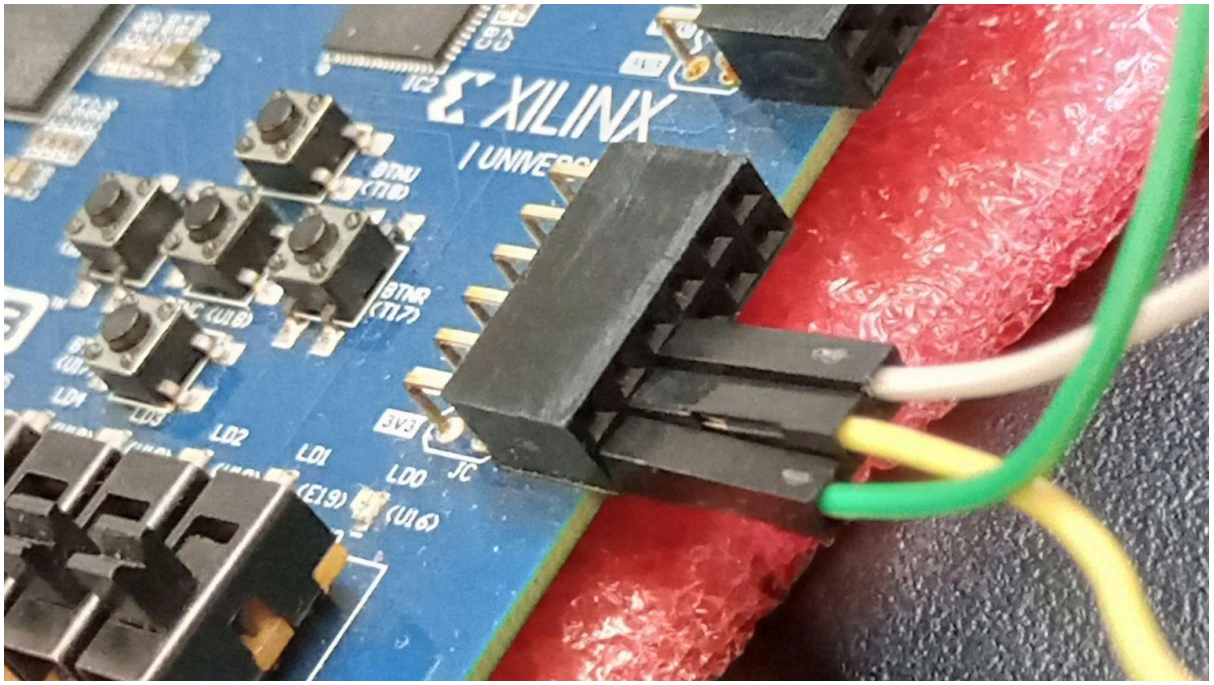
```
set_property IOSTANDARD LVCMOS33 [get_ports A]
set_property IOSTANDARD LVCMOS33 [get_ports B]
set_property PACKAGE_PIN U16 [get_ports B]
set_property PACKAGE_PIN R18 [get_ports A]
```

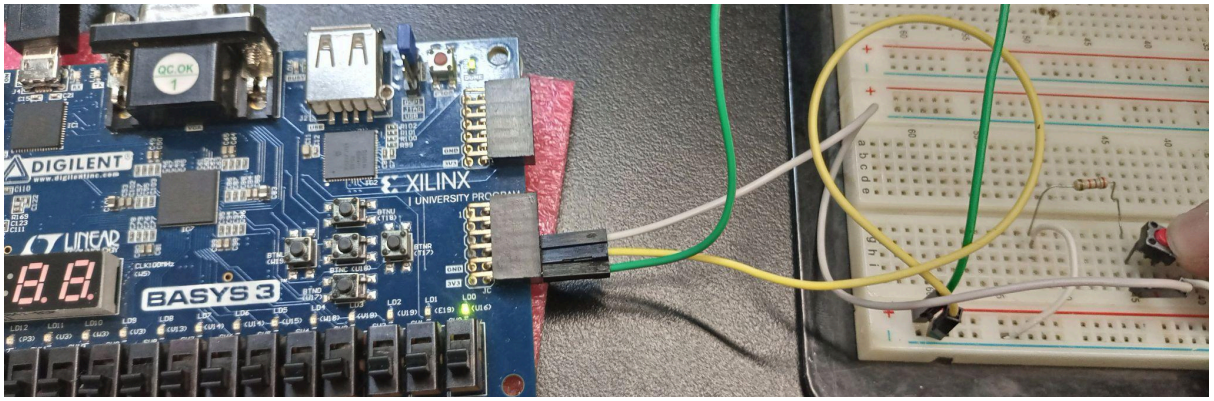
DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

Pictures attached as proof:





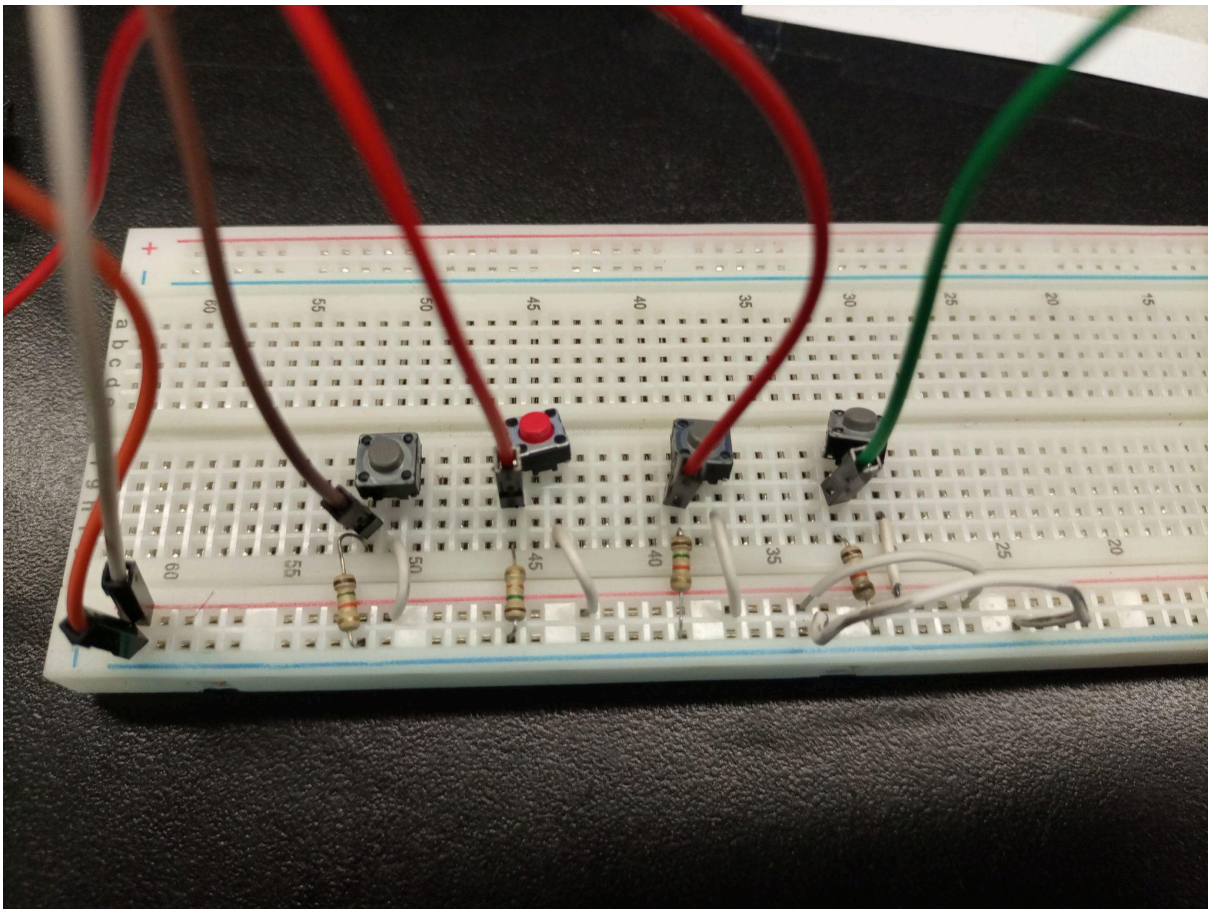


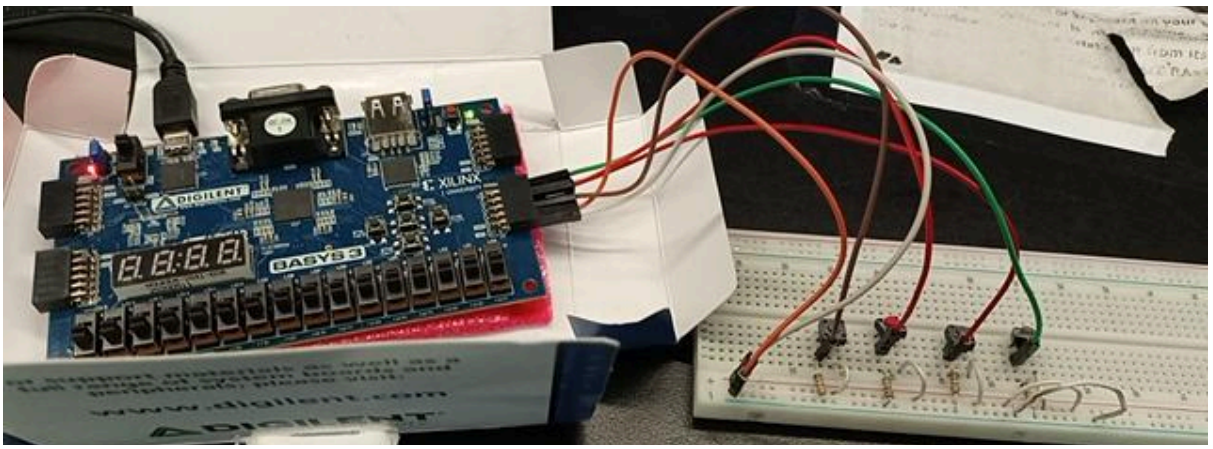
DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

Pin-Configuration for all 4-buttons:

We plan to connect every input circuit in parallel and give an input of Start, Rock, Paper, Scissor and Reset through the port JC3: N17, JC2: M18, JC1: K17, JC11: R18 respectively to our FPGA.





DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

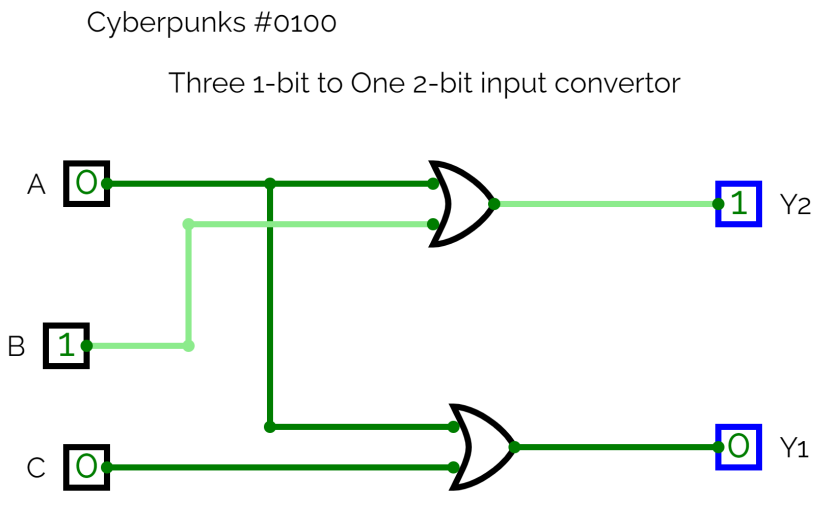
Compressing Three 1-bit inputs into One 2-bit Input for ease later:

We will convert the Three 1-bit input into One 2-bit input so that we have easier time plotting future k-maps using input values. Two buttons being pressed together is a don't care condition.

A	B	C	Y2	Y1
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	x	x
1	0	0	1	1
1	0	1	x	x
1	1	0	x	x
1	1	1	x	x

Y2 = A + B

Y1 = C + A



[4]

Output Block:

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

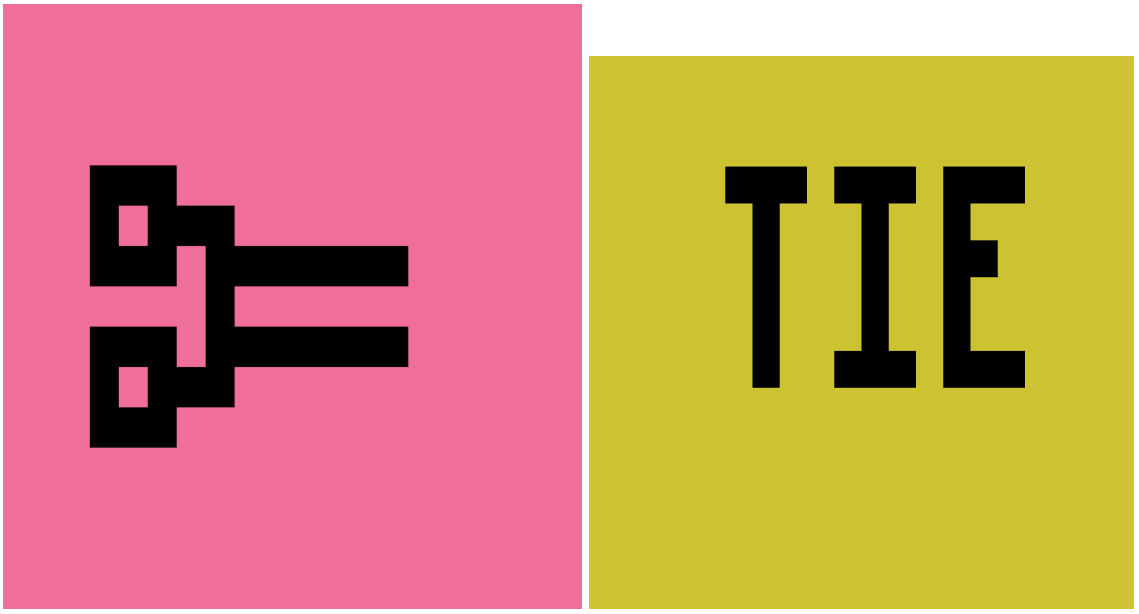
What does our output module care about:

Before we get to the final control block at the end. There are major 8 outputs each output module might show. Title Screen, Waiting, Computer threw Rock, Computer threw Paper, Computer Threw Scissors, You won, You lost, Tie. Each output module will just be given a 3-bit number giving the information of "what output do I want", and VGA/Servos will play their respective output that they must give at the point.

VGA Operation:

Our primary goal is implementing the game with output being a VGA monitor so will be discussing user-experience with that assumption first and foremost.

The following are the displays that are planned to be implemented:



DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

The plan was to divide the 640 x 480 display screen into a 20 x 15 grid which gives us individual pixels of size 32x32. We will manipulate the pixels using the pixel generator module to generate the images as displayed. The colours will be implemented with the help of RGB hexadecimal codes for them. The module checks pixel by pixel, and will display the right colours at the dedicated spots accordingly.

We worked using two important modules, the pixel generator and the VGA top. As mentioned, the pixel generator module managed which pixels were to be colored in what way according to different flag values passed to this module. This leads to the printing of different screens.

Pixel generators work like this (picture is for one screen):

```
always @(flag)
 begin
 case (flag)
    1:
    begin
if ((pixel_x==0) || (pixel_x==639) || (pixel_y==0) || (pixel_y == 479))
  begin
          red <= 4'hF;
          green <= 4'hF;
          blue <= 4'hF;
          end
          else begin
            //for black boundarioes
           if ((pixel_x >= 64 & pixel_x <=96 & pixel_y>= 32 & pixel_y <= 192) ||(pixel_x >= 96 & pixel_x <=160 & pixel_y>= 160 & p
           begin
           red <= video_on ? 4'h0 : 4'hF;
           green <= video_on ? 4'h0 : 4'hF;
           blue <= video_on ? 4'h0 : 4'hF;
           end

           else
           begin


               // for Orange background
               red <= video_on ? 4'hA : 4'h0;
               green <= video_on ? 4'h1 : 4'h0;
               blue <= video_on ? 4'h6 : 4'h0;


          end
```

For different flag values, different screen pixels are displayed

The VGA top module connects all the Hsync, Vsync together. It takes the states as the input. Based on the states, it chooses the correct flag value and passes it in the pixel generator module, allowing the correct screen to be displayed at the required state.

```
always @ (posedge clk)
begin

if (states == 4'b0000)
begin
Tflag=1;

end

else
    begin
    //if (states= 4'b0001) looping waitin

    if (states== 4'b0010) begin //paper
                Tflag=2;
          end
    else
        begin
        if (states== 4'b0011) begin
              Tflag= 3;

              end
       else
         begin
          if (states== 4'b0100)
          begin
          Tflag= 4;
           end
          else
          begin
          if (states== 4'b0101)
            begin
           Tflag=5;
```

the states received will define the flag values. Hence the screens will be displayed on VGA according to the state.

VGA module Verilog File with code to reference as well: https://habibuniversity-my.sharepoint.com/:u:/g/personal/sh07554_st_habib_edu_pk/EbCuKKqbde9Bm5ovl7yC1DwBg1aKdRyJZMv9LzhKebBAeA?e=5baBAZ

FPGA Servo Operation:

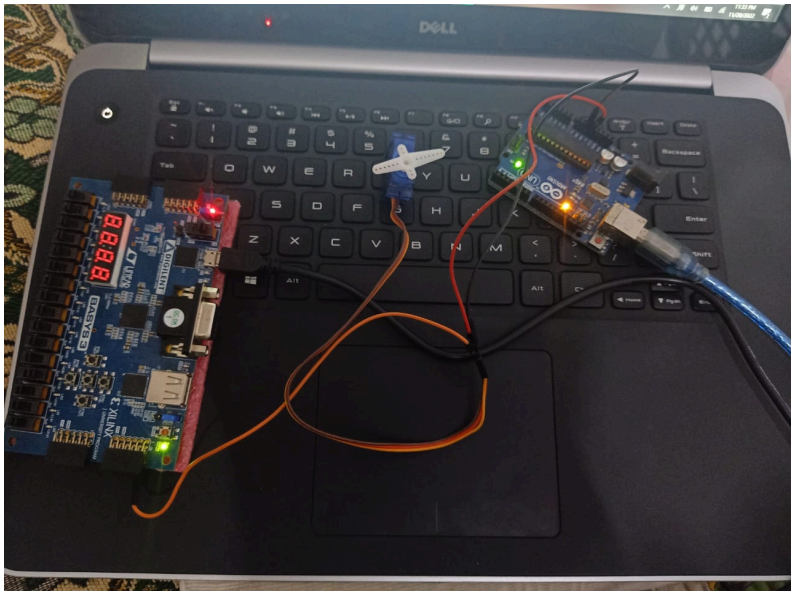
We plan to make this game more interesting by presenting the game status by appropriate movement of a doll, a physical character representing itself as a computerised 2nd player of the game. As the algorithm of our game is already somehow explained in basic description at the beginning, we plan to expand this idea further such that, we will buy a doll from market, and display the game status (Off, Waiting, Won, lost, Draw) by movement of the doll, which will be achieved by implanting/fixing servos in the arms of doll. Servos would be connected to FPGA through PMOD connections, and would be controlled as per the planned motions for each state.

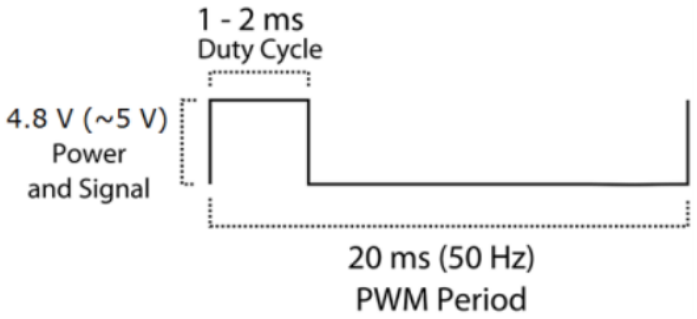
Servos are specifically designed electronic motors which are intelligent that can move to and sustain some specific angle of rotation. This functionality is achieved by an integrated IC and a potentiometer integrated in the package.

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

Till now, we have studied briefly the operation of servos, especially SG90, which we will be using for our project. Servos basically are connected via three wires, among which two act as power source, while third one(Yellow in colour) is the signal wire which operates the servo to a given angle. Servos operate by PWM (Pulse Width Modulation) signals, Sg90 has refresh rate (frequency of signal) 50Hz, and it's duty cycle varies from 0.9ms to 2.1ms, this is the total allowed/useful duty cycle range, and variating duty cycle of signal changes the angle of rotations. Range of Rotation for Sg90 is from 0 degree to 180 degree, which is very suitable for our project. Based on these specifications, we have designed a verilog code to generate a PWM signal with variable duty cycle, which allows users to set the rotational displacement of servo to any specific angle, which we plan to demonstrate in the upcoming evaluation slot. As a reference, Image of servo operation is attached below. Due to the fact that servo requires 5V power supply, and FPGA outputs 3.3V, therefore we used Arduino to get 5V supply, and would later use a 5V battery pack for the project.



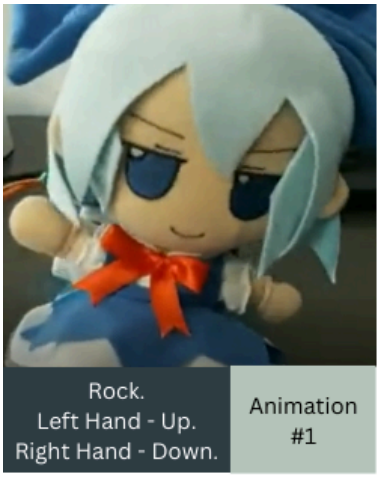


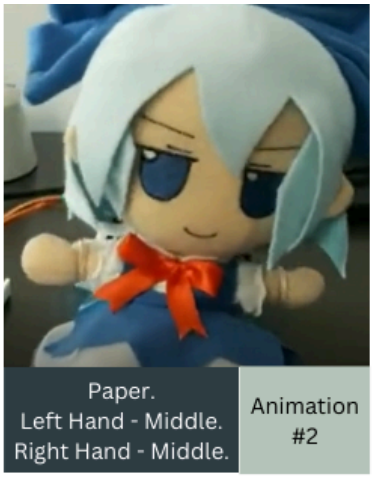
"But how will the doll be shown?"

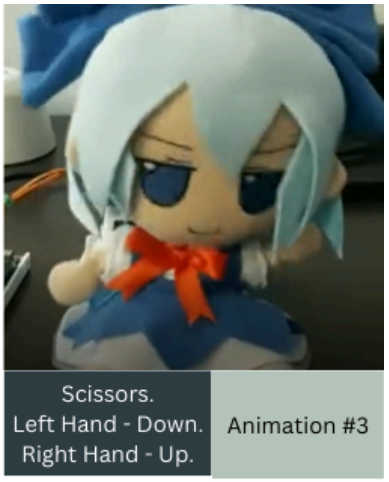
DLD FINAL REPORT

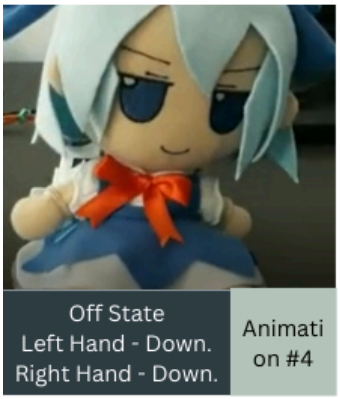
[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

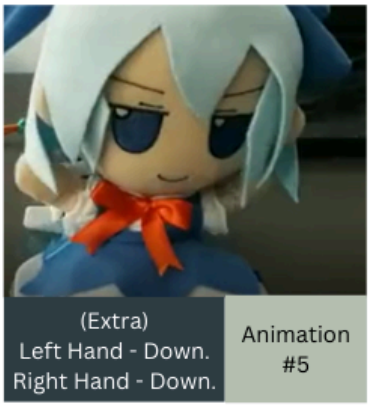
Attached below is our idea on how the dancing doll will be implemented. Idea on how the servos will be given signals and power accordingly.











Basic Sketch:
We plan to attach the Servos legs to the arms of the doll and have them rotate accordingly in different animation states. As you can see in Animation #1, the left arm servo can be rotated at the angle of 135 degree while the right one can be rotated at a 45 angle only. Animation#2 has both servos at 90 degrees.

It will follow the entirety of our main state transition diagram. Only changes to the 2nd "animation" state diagram will be made. We plan to have the doll show output side by side with Monitor with the help of servos. Below is some description to show you how.

[000] Title Screen =By default, the doll will stay still (Animation #4) and won't do anything.
[001] Waiting = Pressing the "any" button starts the game. Doll will go on a dancing loop from Animation #1 -> #2 -> #3-> #4 -> #5 -> #4 -> #5 (dancing) in the span of 3 seconds

[010] Rock = Animation # 1 (No inner looping)

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

[011] Win = Doll Dances (#4 -> #5 -> #4 -> #5 -> #4 -> #5 -> #4).
[100] Lost = Doll Dances (#1 -> #2 -> #3 -> #2 -> #1 -> #2 -> #3 -> #4)

[101] TIE =: (#4 -> #2 -> #3 -> #2 -> #3) [feels like the doll is saying hi].

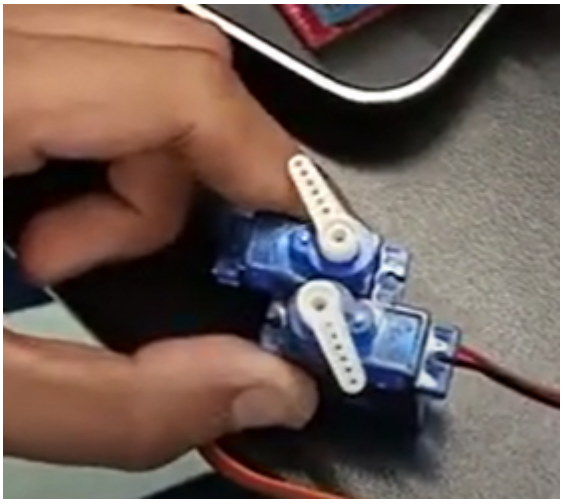
[110] Paper = Animation # 2

[111] Scissor = Animation # 3.

Implementation:

https://www.youtube.com/watch?v=Z8oiNFp2EZY 3:43, here you can observe the clear animation states depending on what signal I give the servos!

Entire Servo Operation Done successfully, implemented and explained by us as well: https://youtu.be/1FpdLu-z5bM



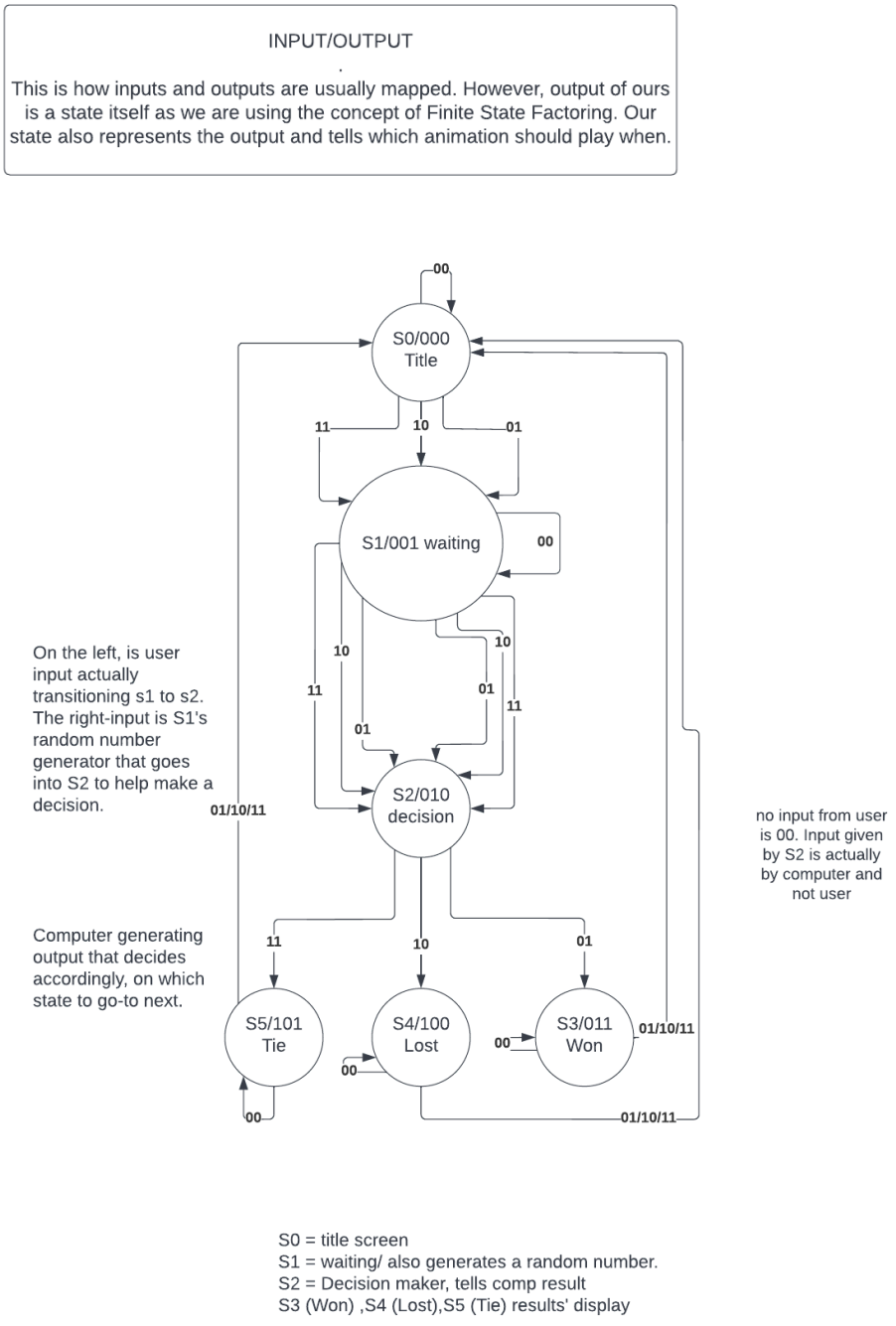
Control Block:

Main State Transition Diagram: [5]

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

The shareable link of these state transition diagrams are attached on the reference page!! The inputs at 010 state that transitions to either Win, lose or tie is from the decision-making module and not the user.

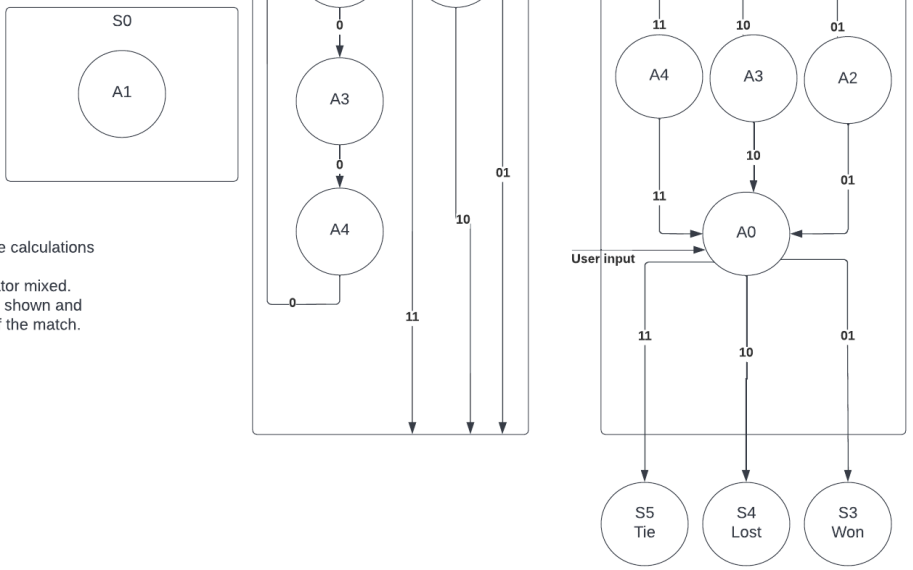


State transition diagram for VGA:

Attached below is our basic sketch of the state transition diagram for the VGA that helped us envision how VGA will show different sets of pictures depending on the

state of the Top-Level-Module in Verilog.

Doing Finite State Factoring.
Meaning Each State is made up of series of states as well. To make an animation, we need to play a series of "images" in a loop which can be defined as a "State" in themselves.
.
Ax = represents an image usually
A0 = Blank Screen /decision
A1 = Title Screen.
A2 = Rock
A3 = Paper
A4 = Scissor
A5 = Congrats! You won.
A6 = You lost! imagine losing, can't be me.
A7 = DRAW! Retry.
Random Number Generator = generates a random number



Inside of S2 " rough idea" to make calculations simpler:
User and random number generator mixed.
The result of the computer will be shown and THEN, we will show the results of the match.
.
Inside=
000 = no input
001 = win/rock
010 = win/paper
011 = win/scissors
100 = lose/rock
101 = lose/paper
110 = lose/scissors
111 = draw.
.
output =
01,10,11

MAIN FSM (FINITE STATE MACHINE), GATE-LEVEL IMPLEMENTATION:

STATE TABLE:

Current State will be my outputs, which will be going into their respective output modules.
Boxes filled with red lines don't care as don’t states don’t exist.
The inputs at 010 state that transitions to either Win, lose or tie is from the decision-making module and not the user.

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

#	Current State			Inputs		Next State			Output		
	A	B	C	D	E	A	B	C	X1	X2	X3
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	1	0	0	0
2	0	0	0	1	0	0	0	1	0	0	0
3	0	0	0	1	1	0	0	1	0	0	0
4	0	0	1	0	0	0	0	0	0	0	1
5	0	0	1	0	1	0	1	0	0	0	1
6	0	0	1	1	0	0	1	0	0	0	1
7	0	0	1	1	1	0	1	0	0	0	1
8	0	1	0	0	0	0	1	0	0	1	0
9	0	1	0	0	1	0	1	1	0	1	0
10	0	1	0	1	0	1	0	0	1	1	0
11	0	1	0	1	1	1	0	1	1	1	1
12	0	1	1	0	0		0	0	0	1	1
13	0	1	1	0	1	0	0	1	0	1	1
14	0	1	1	1	0	0	0	1	0	1	1
15	0	1	1	1	1	0	0	1	0	1	1
16	1	0	0	0	0	0	0	0	1	0	0
17	1	0	0	0	1	0	0	1	1	0	0
18	1	0	0	1	0	0	0	1	1	0	0
19	1	0	0	1	1	0	0	1	1	0	0
20	1	0	1	0	0	0	0	0	1	0	1
21	1	0	1	0	1	0	0	1	1	0	1
22	1	0	1	1	0	0	0	1	1	0	1
23	1	0	1	1	1	0	0	1	1	0	1
24	1	1	0	0	0						

25	1	1	0	0	1						
26	1	1	0	1	0						
27	1	1	0	1	1						
28	1	1	1	0	0						
29	1	1	1	0	1						
30	1	1	1	1	0						
31	1	1	1	1	1						

Excitement Equations:

D(A+1) = BC'D

D(B+1) = BC'D' + A'B'CE + A'B'CD

D(C+1) = C'E + BE + AE + AD + B'C'D + BCD

Output Equations:

X1 = A

X2 = B

X3 = C

Getting Rock, Paper and Scissor from 4-bit Pseudo-Random Number Generator using Linear Feedback Shift Registers:

We created a module that will give us a number anywhere between 0-15 on each positive edge of the clock cycle, and the order will be completely random and won't repeat itself "until" it has exhausted all 16 numbers. After which, it will give the 16 numbers again in the same exact order. Each four-bit number at each instant will be converted to their relevant 2-bit counterpart representing whether they represent rock, paper or scissor.

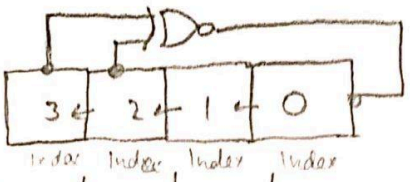
DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

for seed 0001

repeat now

1, 3, 7, 14, 13, 11, 6, 12, 9, 2, 5, 10, 4, 8, 0, 1



$2^n - 1$

$2^4 - 1 = 15$

Number	3	2	1	0
1	0	0	0	1
3	0	0	1	1
7	0	1	1	1
14	1	1	1	0
13	1	1	0	1
11	1	0	1	1
6	0	1	1	0
12	1	1	0	0
9	1	0	0	1
2	0	0	1	0
5	0	1	0	1
10	1	0	1	0
4	0	1	0	0
8	1	0	0	0
0	0	0	0	0

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

Random Number Generator Priority Selector from 4-bits to make it 2-bit information representing Rock (01), Paper(10) or Scissor(11):

	A	B	C	D	O1	O2
0	0	0	0	0	1	0
1	0	0	0	1	0	1
2	0	0	1	0	1	0
3	0	0	1	1	1	1
4	0	1	0	0	1	0
5	0	1	0	1	0	1
6	0	1	1	0	1	1
7	0	1	1	1	0	1
8	1	0	0	0	1	0
9	1	0	0	1	1	1
10	1	0	1	0	1	0
11	1	0	1	1	0	1
12	1	1	0	0	1	1
13	1	1	0	1	0	1
14	1	1	1	0	1	0
15	1	1	1	1	1	1

O_1 = D' + A'B'C + AB'C' + ABC

O_2 = D + A'BC + ABC'

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

```
module lfsr4(
    input clk,
    input reset,
    input clear,
    output [1:0] Doll_Answer,
    output reg [3:0] lfsr = 4'b0001
    );
    always @(posedge clk or posedge reset)
        if(reset)
            lfsr <= 4'b0001;
        else if(clear == 0)
            lfsr <= lfsr;
        else begin
            lfsr[3:1] <= lfsr[2:0];
            lfsr[0] <= lfsr[3] ~^ lfsr[2];
        end

    //D' + A'B'C + AB'C' + ABC
    assign Doll_Answer[1] = (~lfsr[0] || (~lfsr[3] && ~lfsr[2] && lfsr[1]) || (lfsr[3] && ~lfsr[2] && ~lfsr [1]) || (lfsr[3] &

    //D + A'BC + ABC';
    assign Doll_Answer[0] = ( (lfsr[0]) || (~lfsr[3] && lfsr[2] && lfsr[1]) || (lfsr[3] && lfsr[2] && ~lfsr[1]) );


endmodule
```

The following code works likes this : the lfsr4 module takes in the arguments of a slow Hz clock , a reset , clear (load) , Two_bit output of Doll_Answer and 4 bit output register of lfsr which is set to seed value of 1.

In the always block whenever there is a positive edge or when reset button is pressed the if statements are realised . if the loop is triggered by the reset button then it will reset the lfsr register back to the seed value . However if clear button is pressed during positive edge the value of lfsr register will remain unchanged . Furthermore to act as a pseudo-random variable the next two statements are very important : In the else condition when only positive edge triggers the loop as stated in the name of lfsr (linear feedback shift register) , it will do both task in parallel using non-blocking statements i.e : shift the bits in the indexes of 2,1 and 0 to the indexes of 3,2 and 1. Moreover it will also using Gate implementation to Exclusive - NOR out the bits in the indexes of 3 ,2 for which the result is then given to the index 0 of the lfsr register .

After the loop we use gate implementation again to convert the 4 bit answer in lfsr register to 2 bit output using our previously derived equations from the truth table .

Videos referenced :

https://www.youtube.com/watch?v=C82JyCmtKWg

https://www.youtube.com/watch?v=ZIoTmcKPl4w

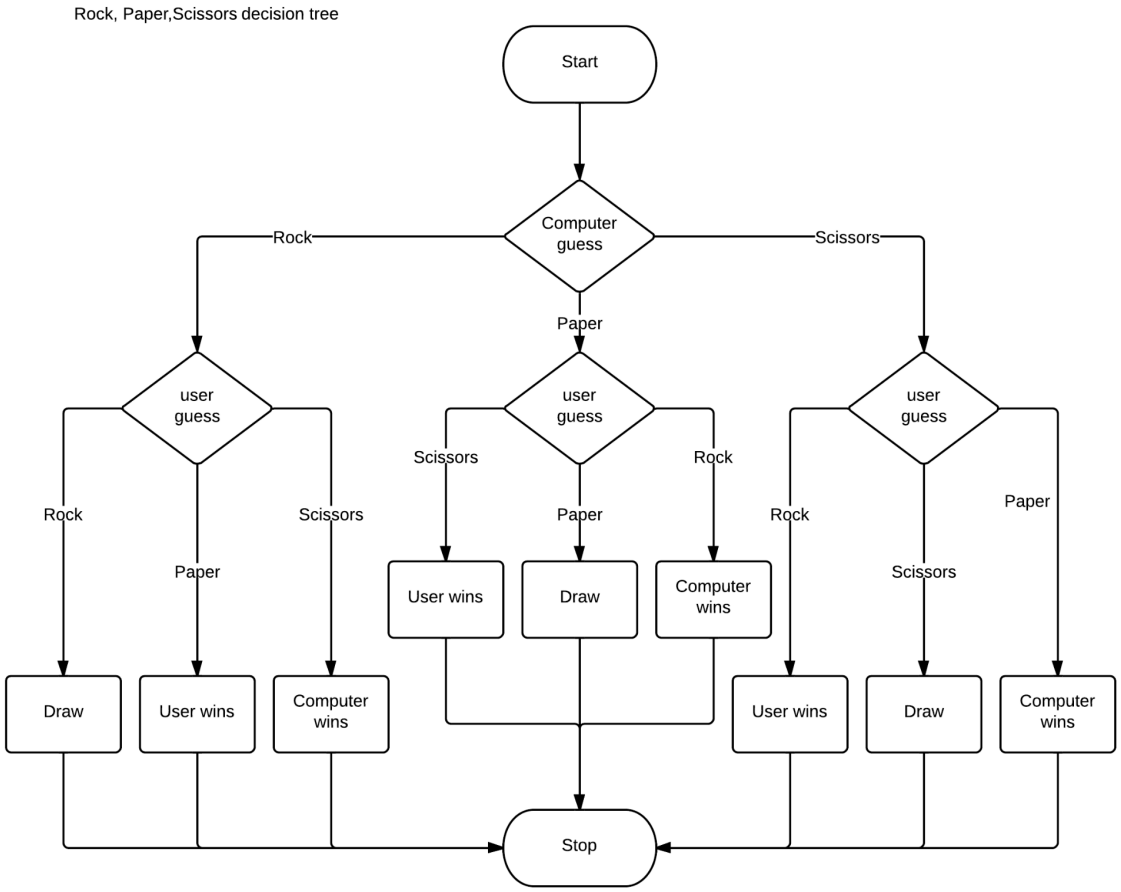
https://www.youtube.com/watch?v=Ks1pw1X22y4

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

https://www.youtube.com/watch?v=l6d5oi9b7lM

The following flow-chart will be used to decide the winner of the match:



Decision Maker, comparing 2 bits from user input and 2 bits from Random Number Generator :

Inputs: 01 - Rock, 10 - Scissor, 11 - Paper

Output: 01 - win, 10 - lose, 11 - tie

(Remember 00 input can't be from either user or Computer so don't care output in those case)

Also, the output from this module represents the decision.

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

	I1	I2	O1	O2	D1	D2
0	0	0	0	0	X	X
1	0	0	0	1	X	X
2	0	0	1	0	X	X
3	0	0	1	1	X	X
4	0	1	0	0	X	X
5	0	1	0	1	1	1
6	0	1	1	0	1	0
7	0	1	1	1	0	1
8	1	0	0	0	X	X
9	1	0	0	1	0	1
10	1	0	1	0	1	1
11	1	0	1	1	1	0
12	1	1	0	0	X	X
13	1	1	0	1	1	0
14	1	1	1	0	0	1
15	1	1	1	1	1	1

D1 = A'D' + B'D' + BC' + ACD

D2 = A'C' + B'C' + AD' + BCD

Extra:

General comments on challenges faced:

Project Scale: This was amongst one of the most interesting Uni Projects so far due to how heavy it "was" yet was equally fun. I wished it was extended further into the semester so we could spend more time just refining the project instead of having 1 day after finals only, but I guess that's university experience in a nutshell. Being forced to think about our sequential logic and FSM, various states when we only

have barely any idea of what even a Moore Machine is, was definitely daunting on many levels.

Verilog: We were pretty confident in our Verilog skills and expected to struggle purely with registers for pseudo-random number generators as we were taught those in the last hectic week. However, unlike labs, where you are handling many modules and many FSMs which have "FSMs" within themselves resulting in FSM Factoring, it becomes extremely easy to fall into the habit of thinking things "abstractly" like in your normal high-level software programming language. Referring to having a module in the top-level module as "calling a function" which wasn't exactly the right way to look at it and that's why we faced many various problems when it came to manipulating "always", "initial", "blocking" and "non-blocking" statements. I guess, this project was the "actual" exposure for us to the power of parallel and sequential programming, and why Verilog is actually good.

Code Repository:

Main Project:
https://docs.google.com/document/d/1mMNt7o-MtOfh-CjJVZQpDK4PZY1l2mabpcyGlY5NrsM/edit?usp=sharing
VGA Separately:
https://habibuniversity-my.sharepoint.com/:u:/g/personal/sh07554_st_habib_edu_pk/EbCuKKqbde9Bm5ovl7yC1DwBg1aKdRyJZMv9LzhKebBAeA?e=5baBAZ (Our Entire VGA module verilog file)

Servo Operation Code:

https://docs.google.com/document/d/1ai2WHQsPWw-76Ts1HpeX_niGanEwe_eDRgd4QtKREF4/edit?usp=sharing

DLD FINAL REPORT

[Shahjahan, Muhammad Shaheer, Syed Asghar Abbas Zaidi, Syed Muhammad Ather Hashmi]

References:

[1] https://lucid.app/lucidchart/3dbf1ed9-6de9-4759-8774-59cbed0f521c/edit?viewport_loc=-10%2C-10%2C1645%2C821%2C0_0&invitationId=inv_04b5360e-77d0-4805-b56b-10ab248f85b7

[2] https://www.youtube.com/watch?v=Wpd5-Yd4p3M (MAIN Inspiration!, recommended to turn down the volume)

[3] https://www.pixilart.com/draw/8-bit-grid-mario-bross-gabriel-87bfcce40372c0b#

[4] https://circuitverse.org/simulator/edit/3-1-bit-to-2-bit-input-convertor

[5] Main State Transition Diagram: https://lucid.app/lucidchart/776a2dcb-3271-4867-99fa-a3672d843be5/edit?viewport_loc=-116%2C-1063%2C2165%2C944%2C0_0&invitationId=inv_0c389f4d-a263-4c41-979d-8d034fb2ecb4

Nested State Transition Diagram: https://lucid.app/lucidchart/d2c81900-bc45-49ee-bd3c-59e1f23458b0/edit?viewport_loc=-466%2C-307%2C1707%2C744%2C0_0&invitationId=inv_44a4cafb-1b6d-4816-85eb-df8fe7c5b805

[6] Our original Proposal attached for reference: https://docs.google.com/document/d/1w63DvykMbG3CFrssoWx0T675sa0qsa1R/edit?usp=sharing&ouid=108784058789660023611&rtpof=true&sd=true

[7] DLD Video: Giving an overview of the entire module, how we will take input, Servo and VGA implementation, 4-bit random number generator and so on https://www.youtube.com/watch?v=Z8oiNFp2EZY

**Any other thing used in making of this report can be provided if asked*

Thanks for Reading all the way! And for such a great time in DLD!

And all the project mates who saw this project all the way through the end, of trying to make this fun project to life. Who respectively contributed to different modules and inserted their own bits of personality with their respective strengths and weaknesses!