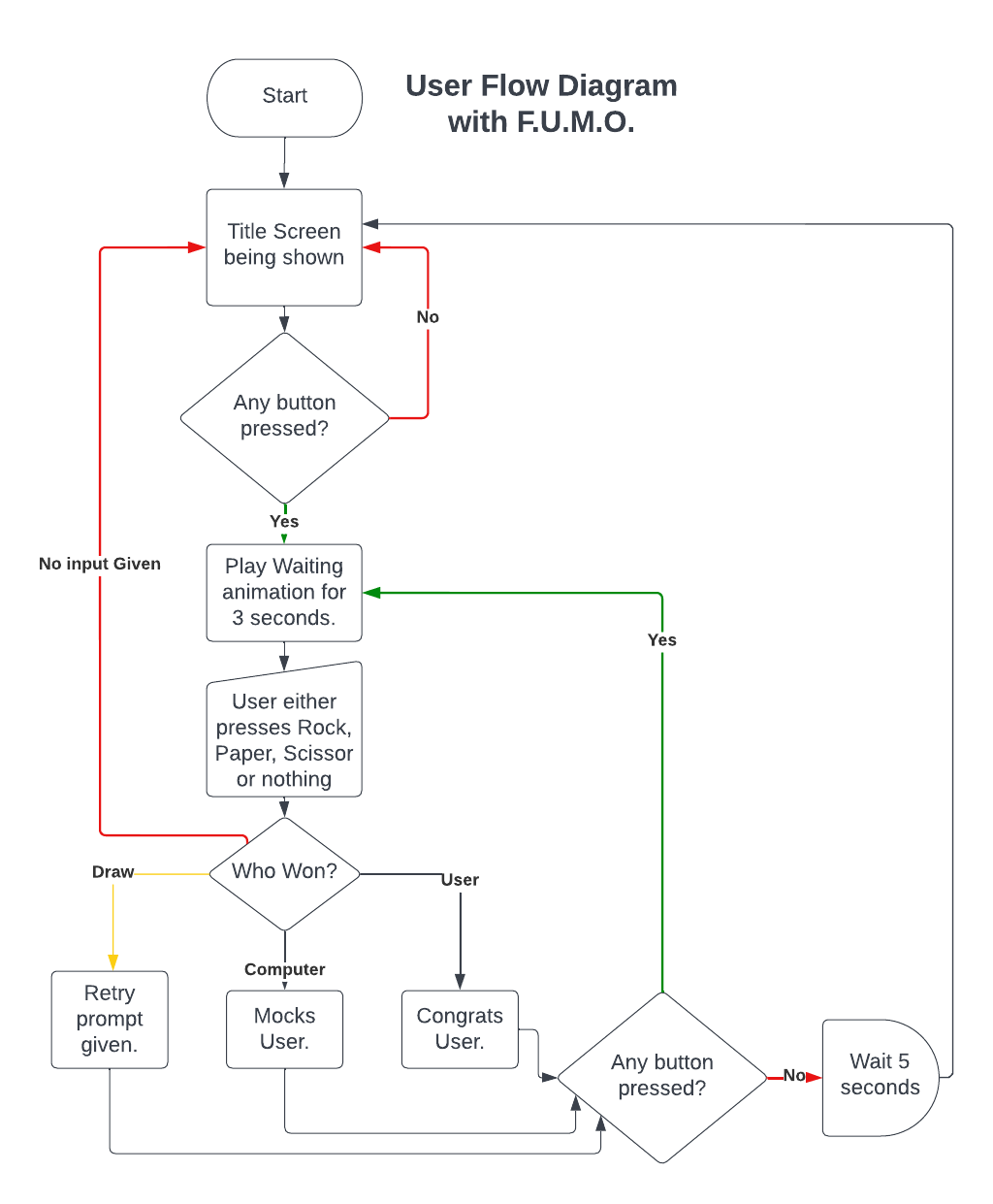
**Project Title:**

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]

****

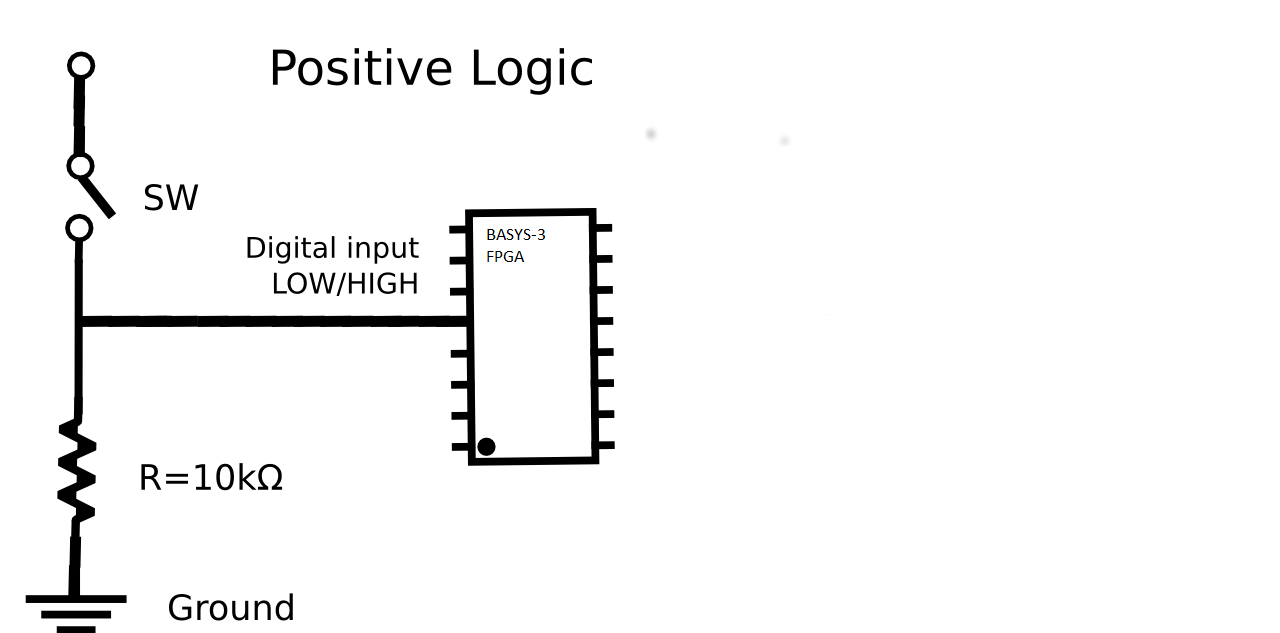
Basic Description of the above flow chart:  
Our primary goal is implementing the game with output being a VGA monitor so we will be discussing user-experience with that assumption first and foremost.

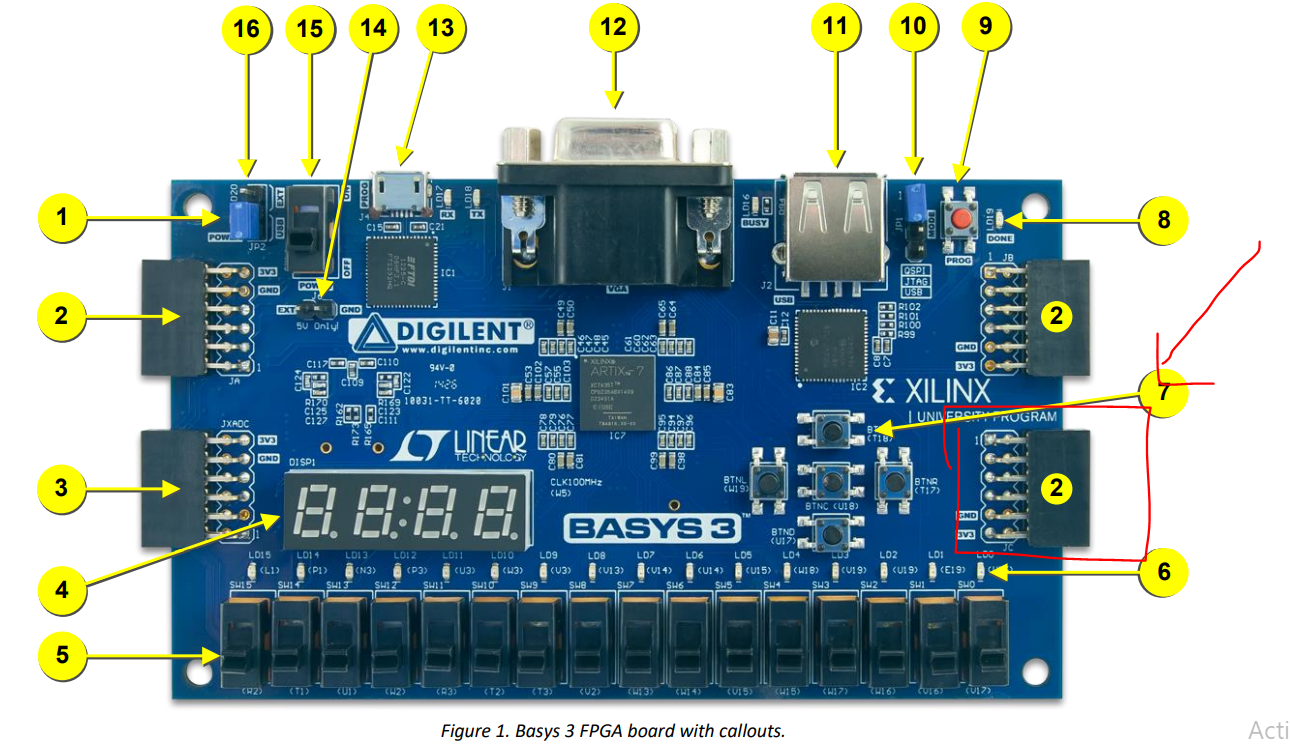
When the system is powered up, the player will be shown our game’s “title screen” welcoming them. When any button is pressed (except “Reset”, as that button is implemented as a good practice) , the FPGA will play a looping animation between “rock”, “paper”, “scissor” and will also decide on any one of them as well in the background. This waiting animation of 3 seconds will be a buffer time-period in which the user will give input with the use of a button representing either Rock, Paper or Scissor.

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

**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.

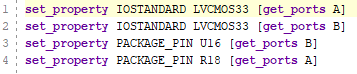
  
We plan to use a father-father package-pin and take 1 VCC and 1 GND to power our breadboard power rail, and then implement our above logic there. The Logic Diagram for one “input is attached” to give an understanding.



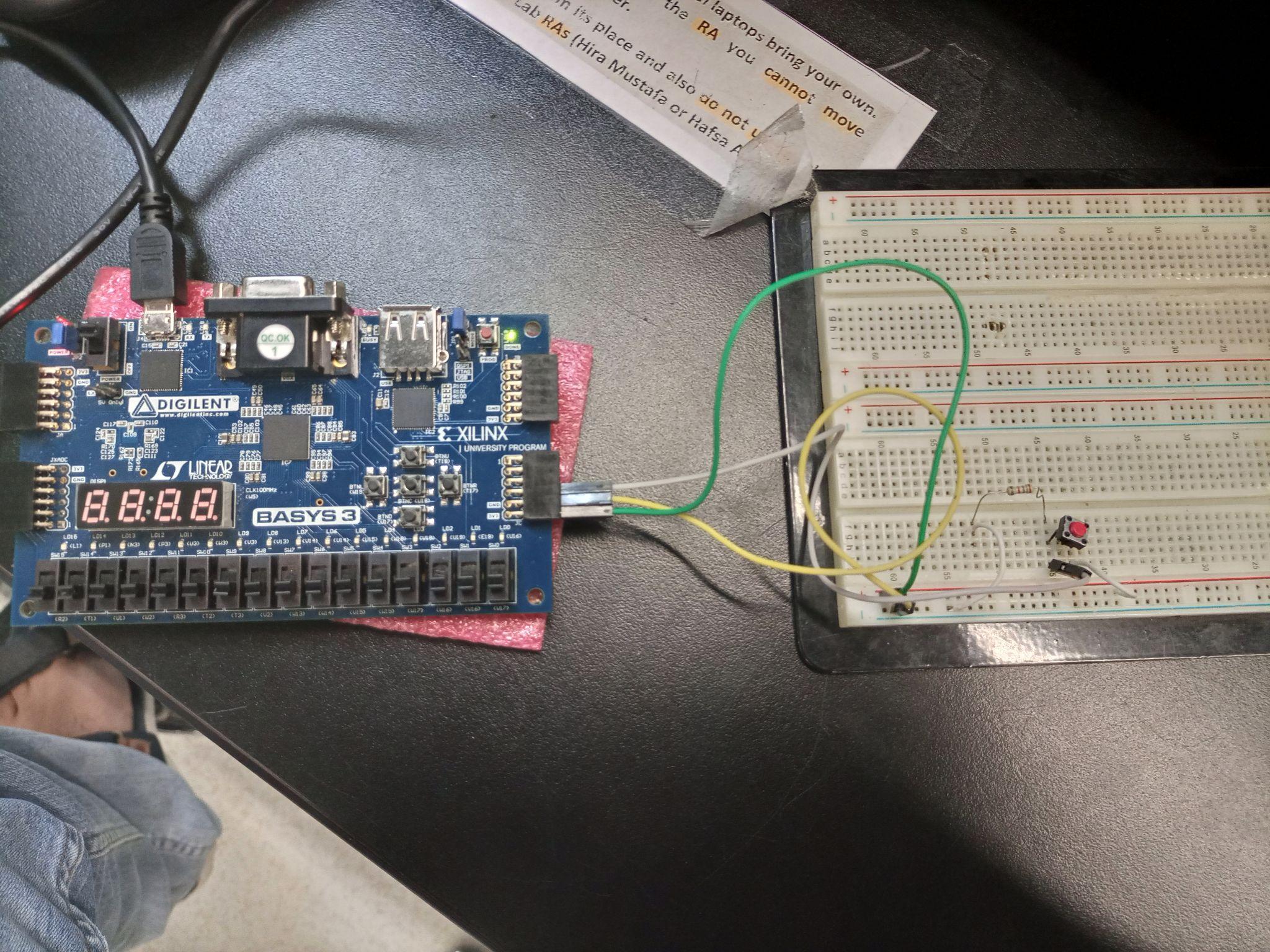
We have also tested it practically, we programmed our FPGA in a way that it would light an LED when it would detect the input signal being provided at R18 port of Pmod-JC.

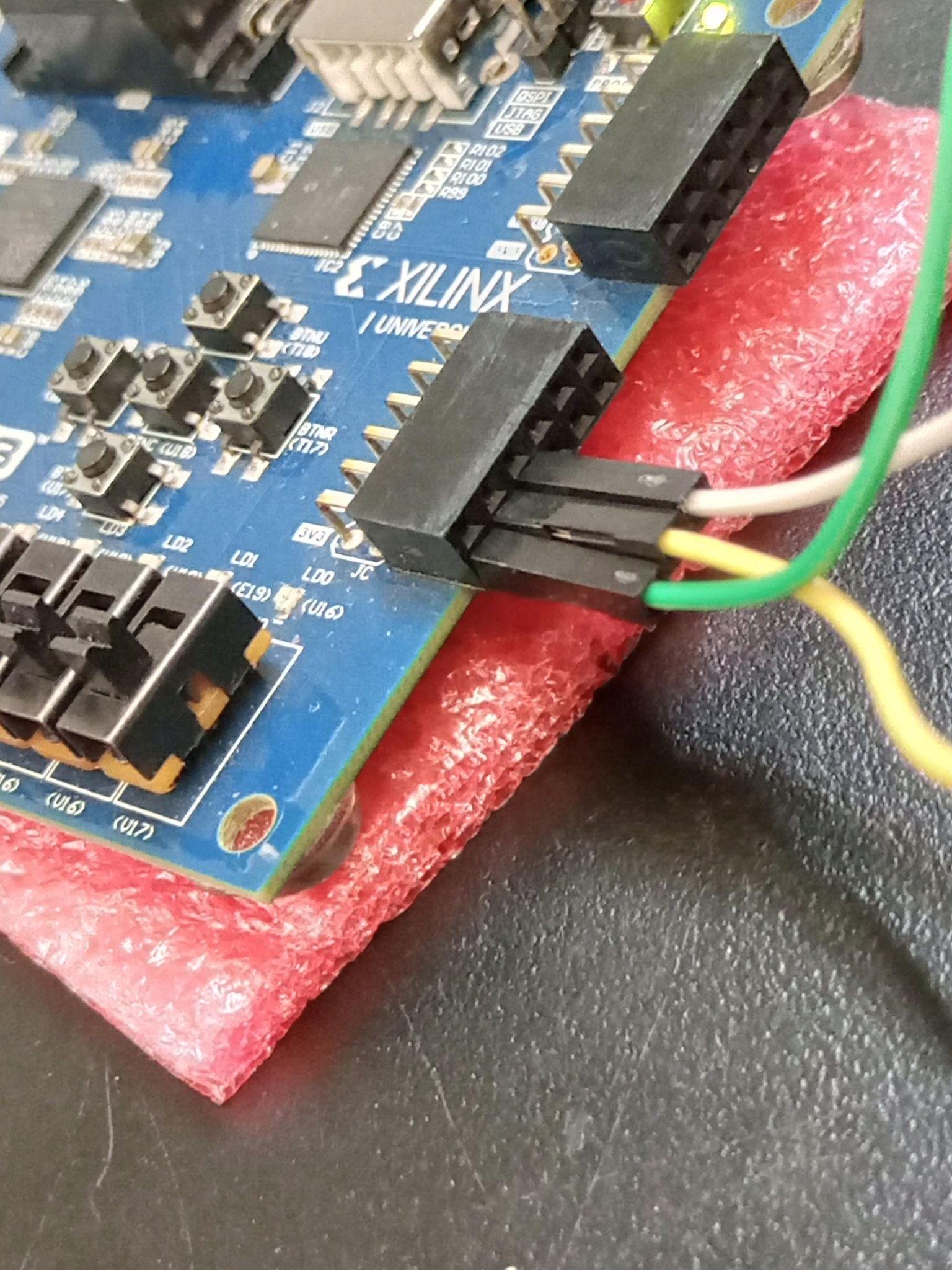
**Verilog Design Code:**  

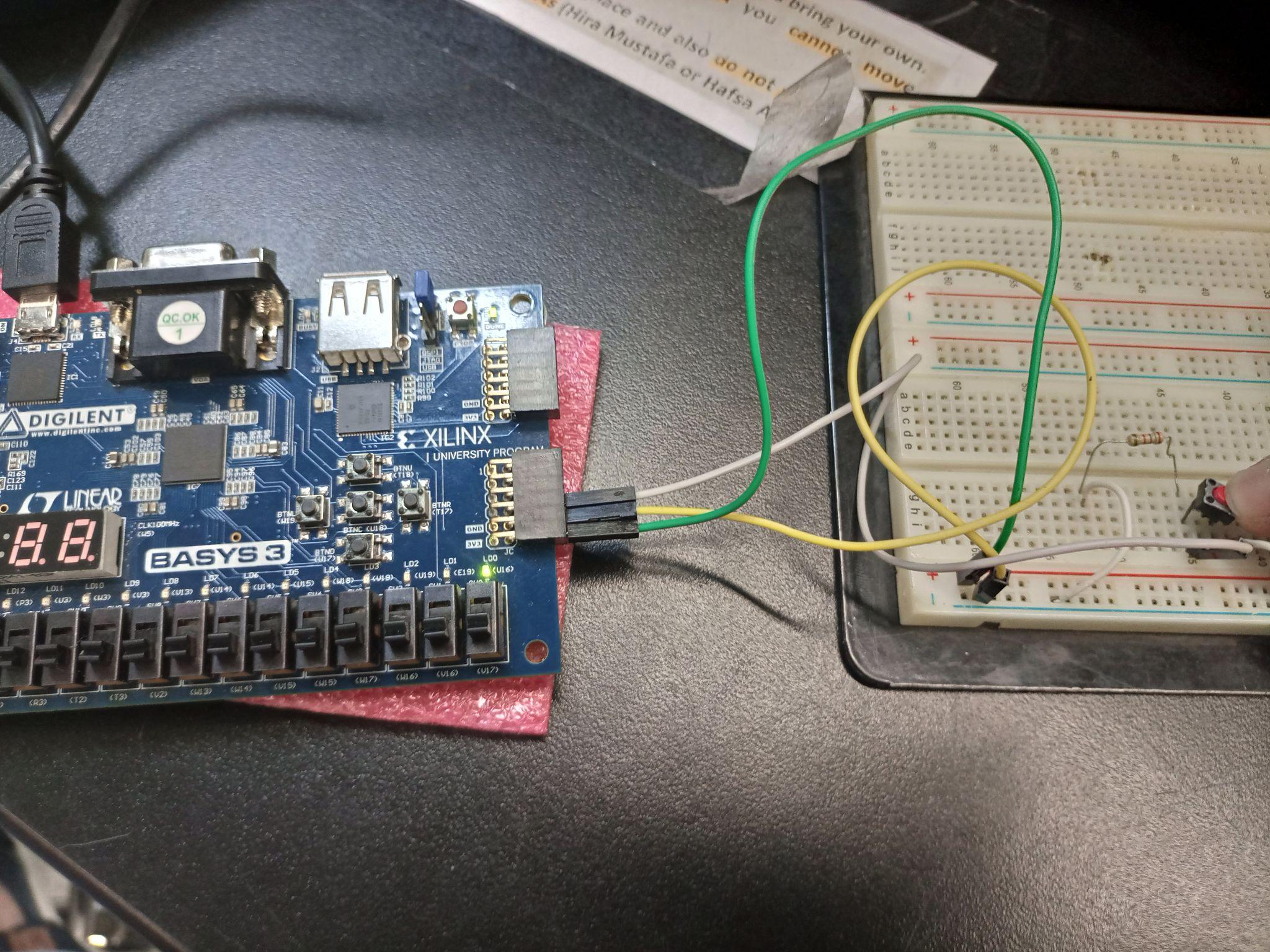

**Constraint File:**



**Pictures attached as proof:**





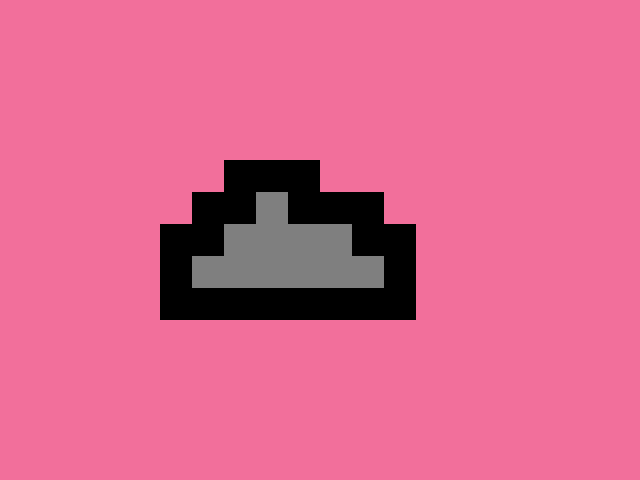
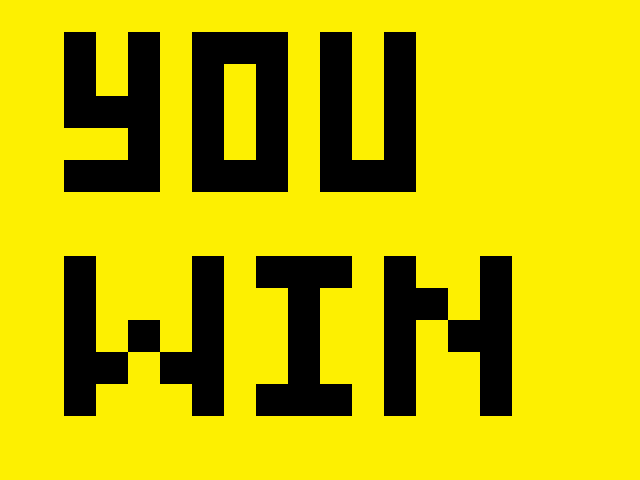
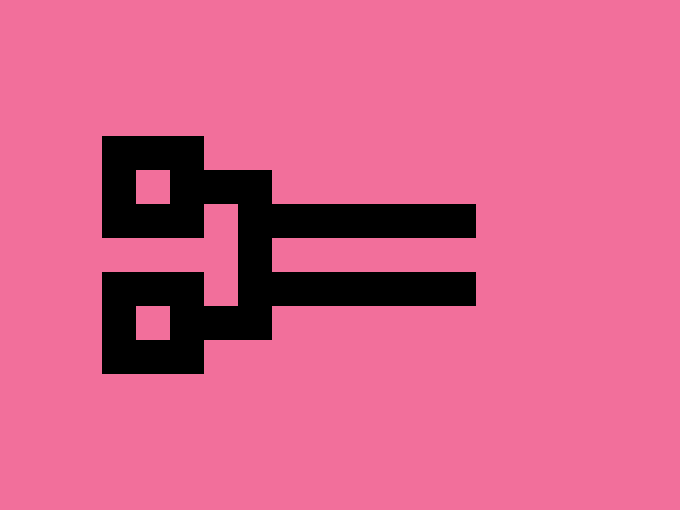


We plan to connect every input circuit in parallel and give an input of Start, Rock, Paper, Scissor and Reset through the port JC3: N17, JC2: M18, JC1: K17, JC11: R18 respectively to our FPGA.

**Output Block:**

**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:

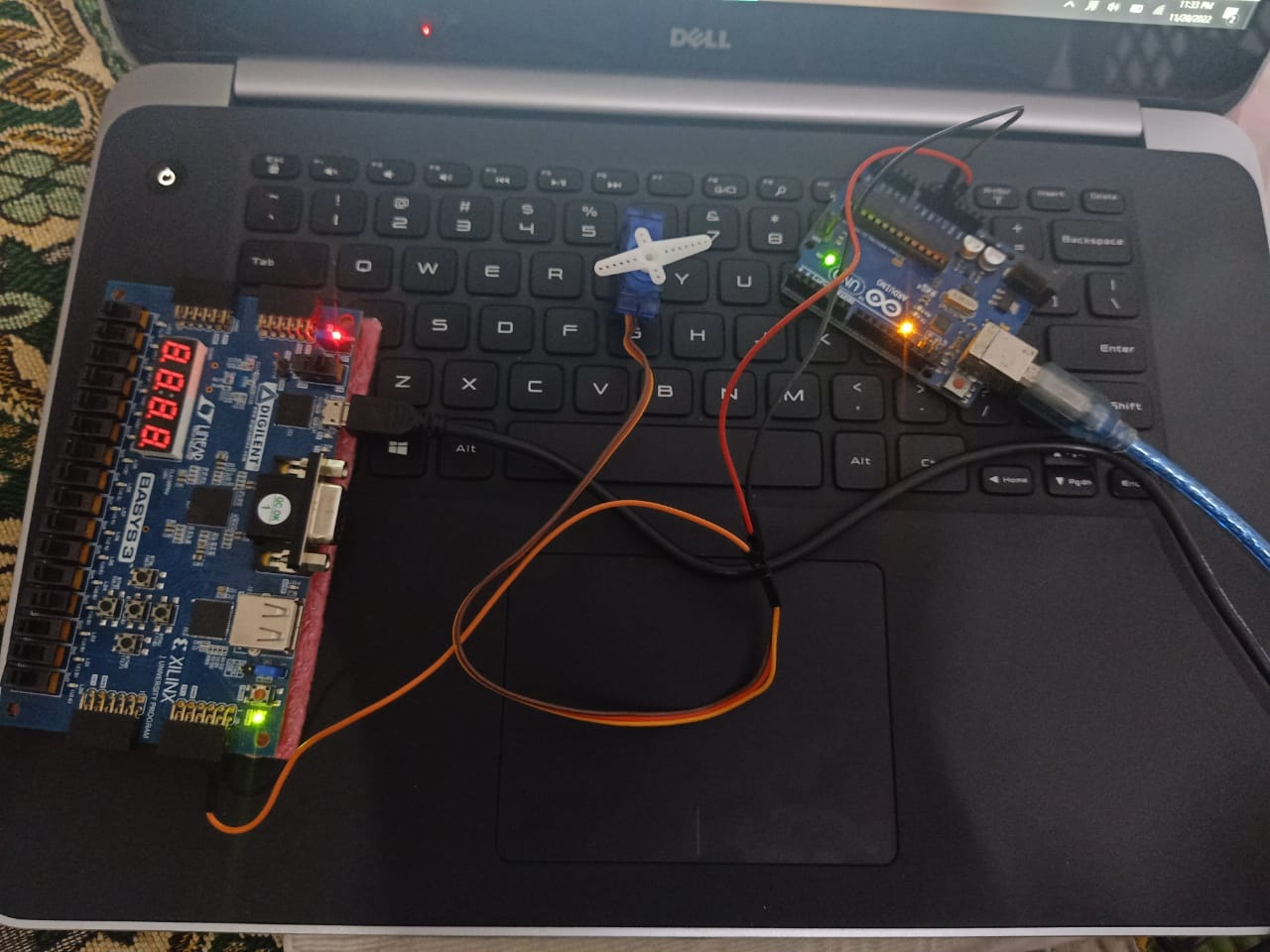


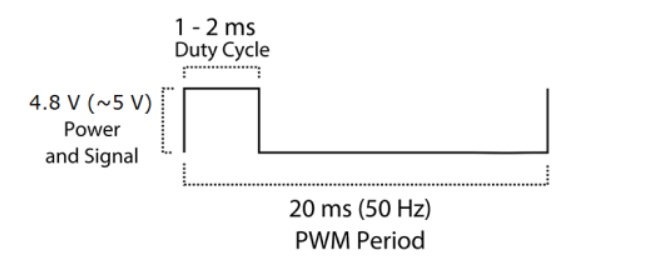
Currently, work is being done to implement one of the states.

The plan is 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 will check pixel by pixel, and will display the right colours at the dedicated spots accordingly.

**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.

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. Servos are useful because they are specifically designed electronic motors that can sustain some specific angle of rotation. 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.



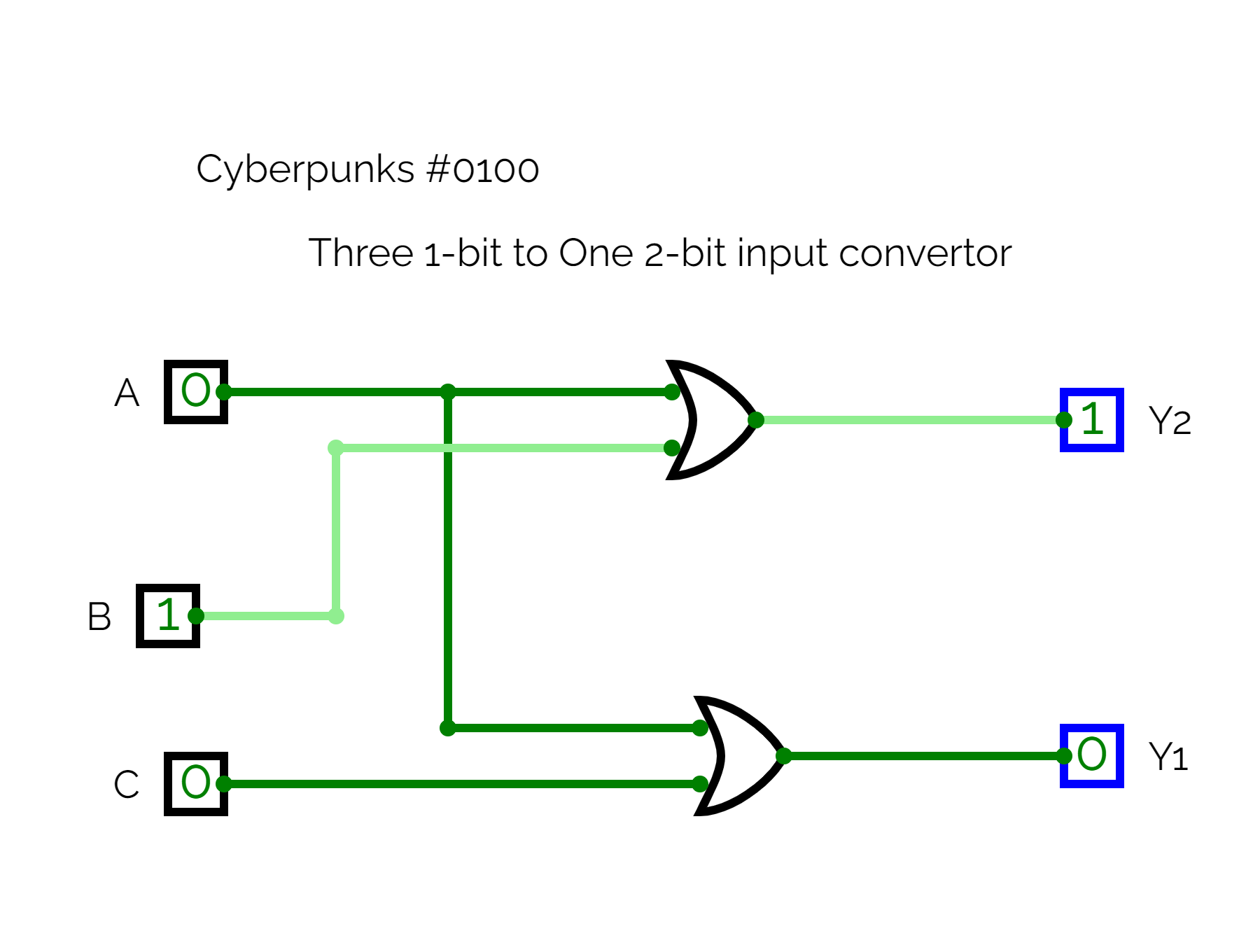
**Control Block:**

We will convert the Three 1-bit input into One 2-bit input to simplify the calculations. 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 |

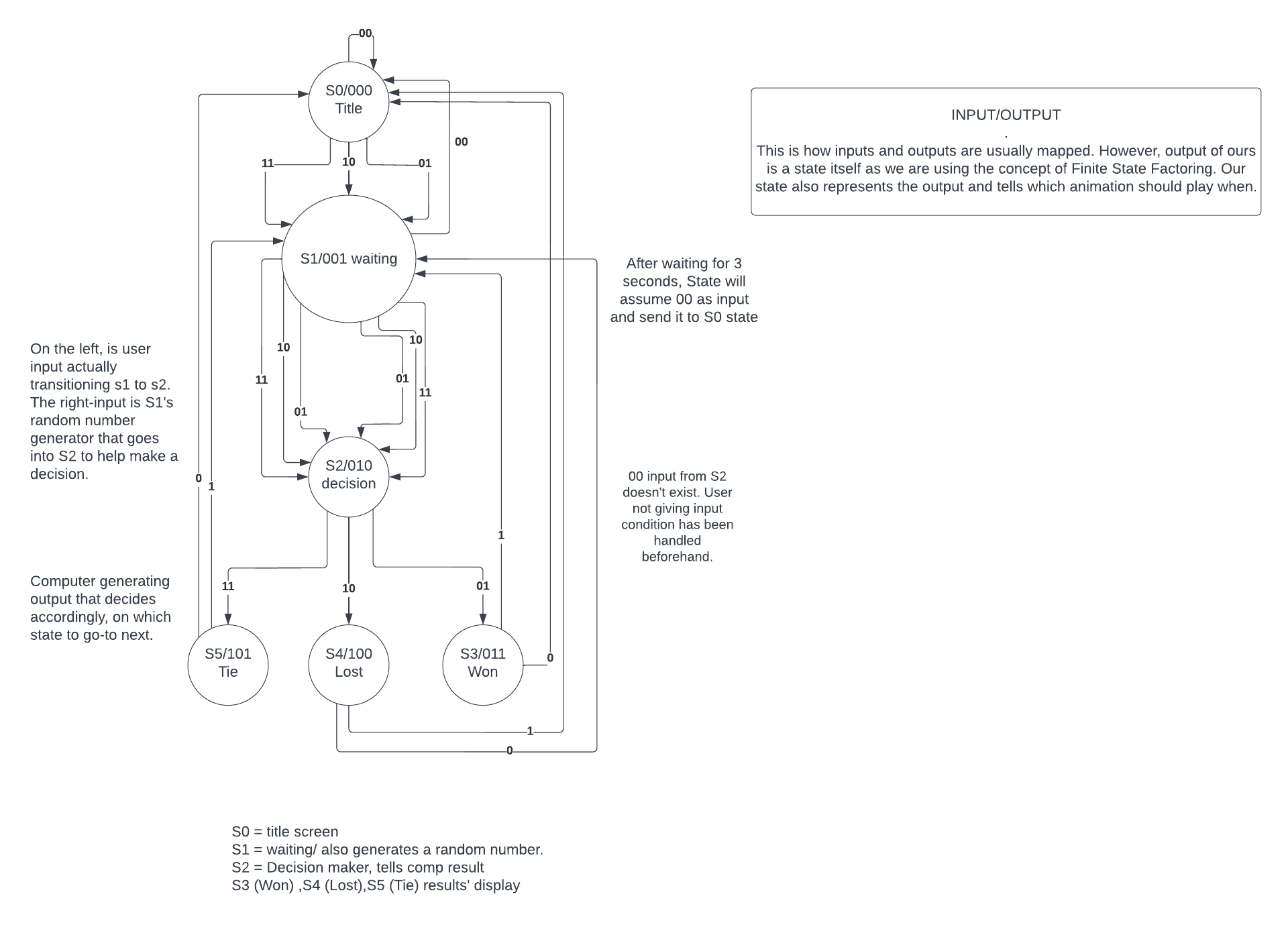
Y2 = A + B

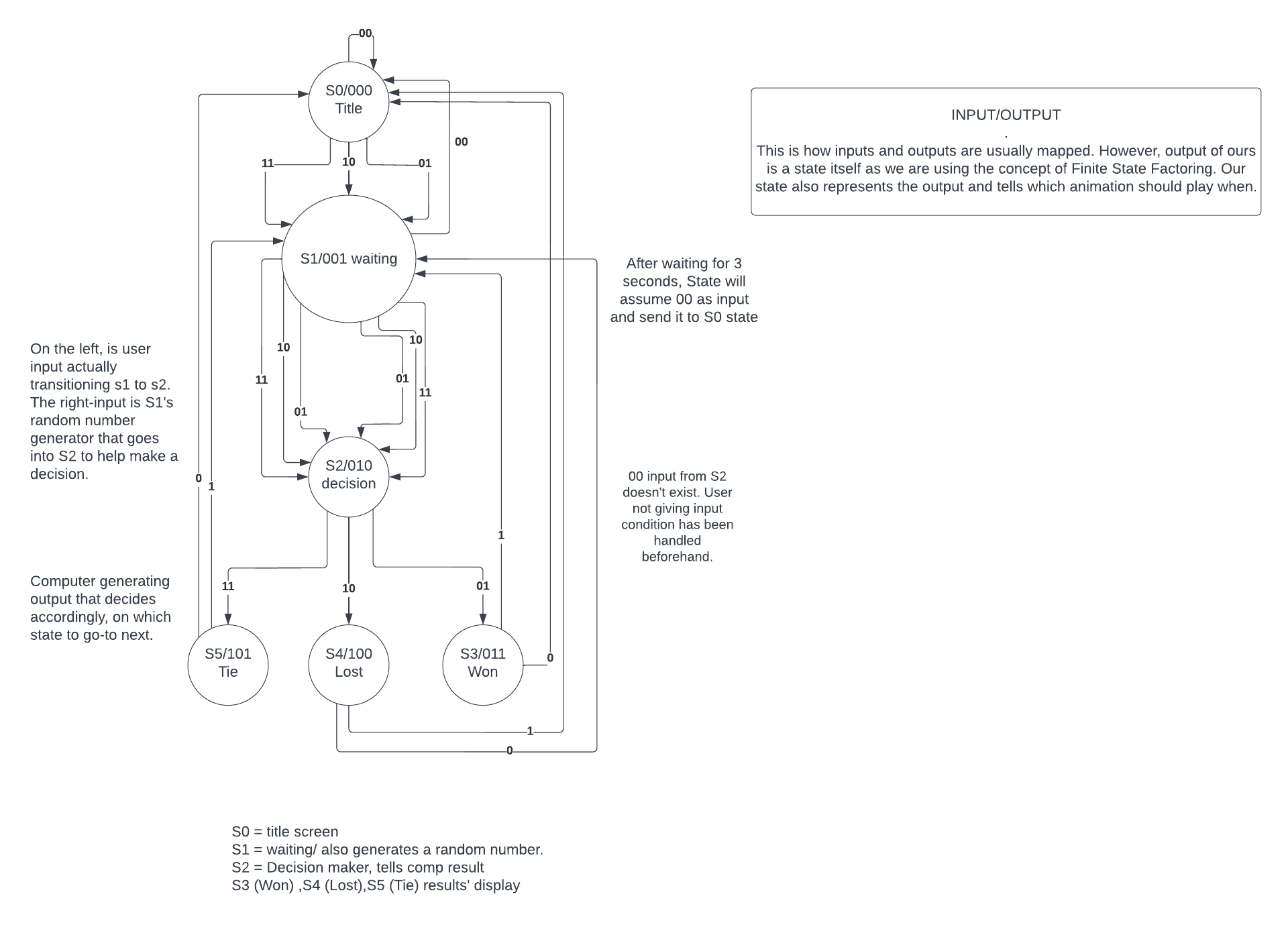
Y1 = C + A

[4]

**State Transition Diagram: [5]**

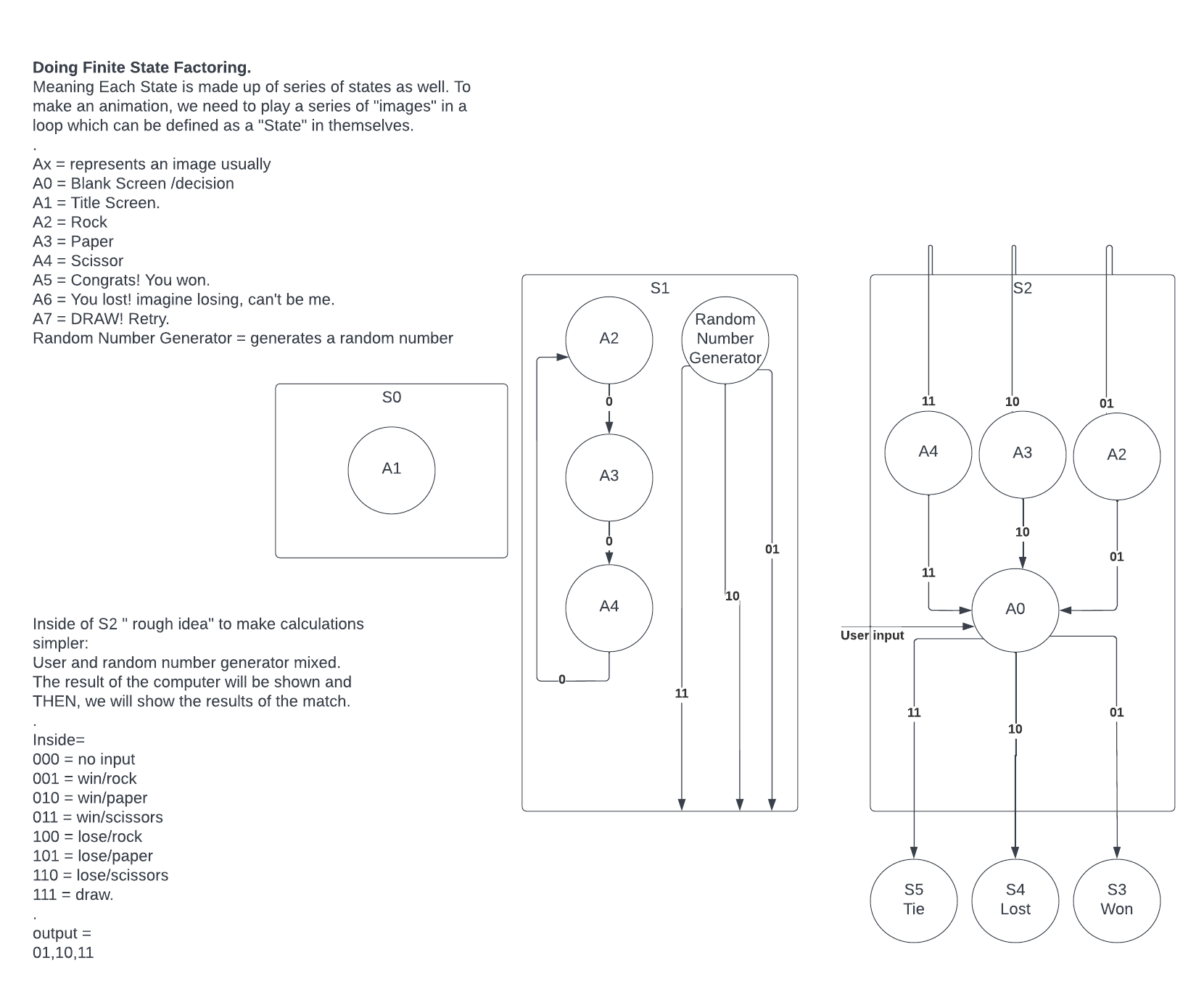
The shareable link of these state transition diagrams are attached on the reference page!!

****

****

**Assumption:** “State transition diagram for VGA”

As you have seen in the “output” block, the team is also working their hardest to implement an actual doll dancing with the use of Servo Motors. But as we were told that VGA output is necessary, and that has been our primary concern first and foremost. And as such, the one shown below is regarding how the state transition diagram will look for VGA output.

****

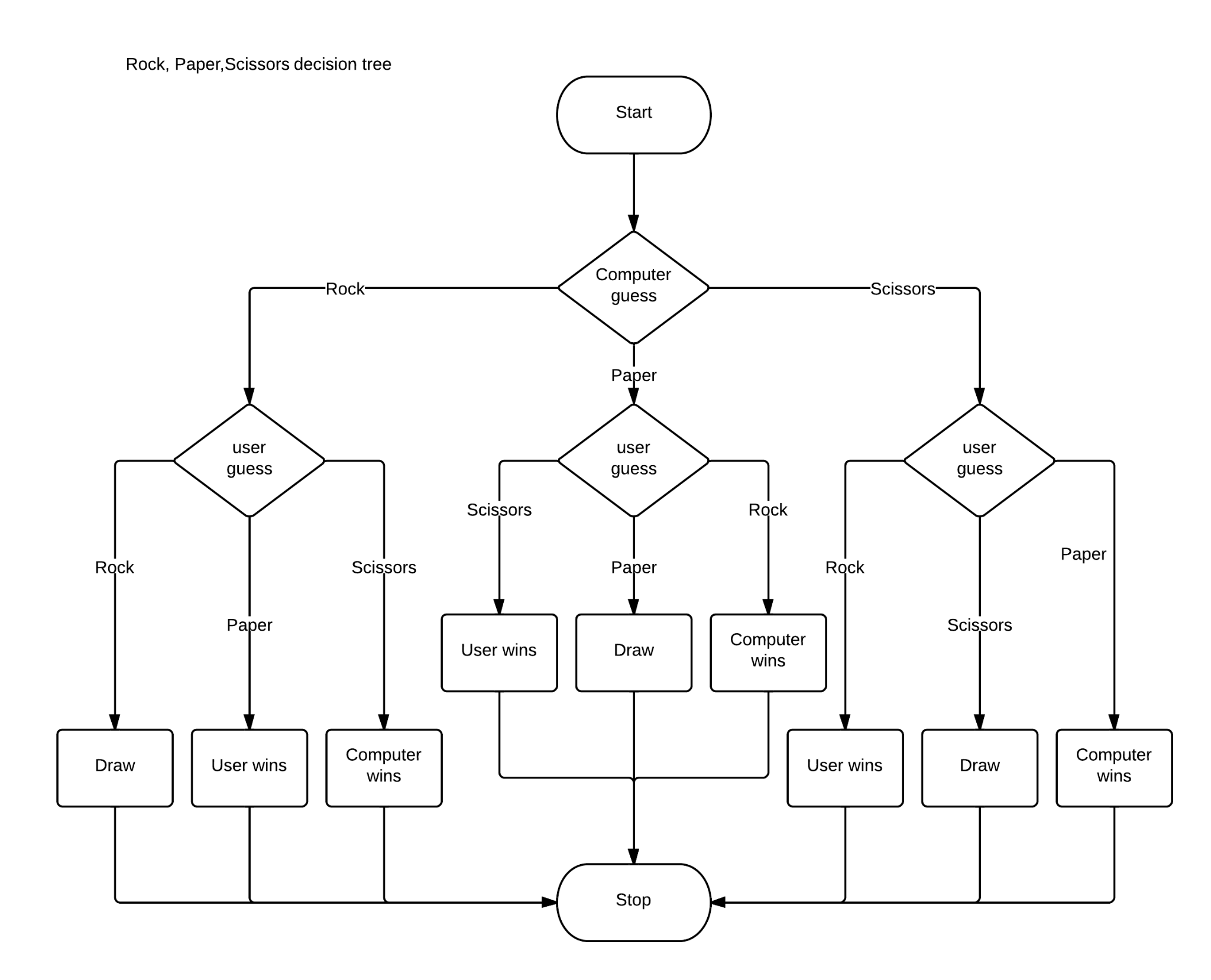
**Random Number Generator Priority Selector from 4-bits:**

|  | 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 |

O1 = D' + A'B'C + AB'C' + ABC

O2 = D + A'BC + ABC'

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



**Extra:**  “But how will the doll be shown?”

For the sake of clarity and explaining all of our plans, attached below is our idea on how the dancing doll will be implemented. Idea on how this design will be implemented is talked about in the “output section”.



**Basic Sketch:**

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.

[S0] Title Screen =By default, the doll will stay still (Animation #4) and won’t do anything.  
[S1] 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

[S3] Win = Doll Dances (#4 -> #5 -> #4 -> #5 -> #4 -> #5 -> #4).   
[S4] Lost = Doll Dances (#1 -> #2 -> #3 -> #2 -> #1 -> #2 -> #3 -> #4)

[S5] TIE =: (#4 -> #2 -> #3 -> #2 -> #3) [feels like the doll is saying hi].

**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>  
  
*\*Any other thing used in making of this report can be provided if asked*  
  
***Thanks for Reading all the way through!***