Habib University

EE/CS 172/130

Digital Logic and Design

Final Project Progress Report

**Project**

Interactive Piano

**Group Members**

Simal Anjum, Zainab Raza, Ammar Ahmed, Rida Fatima

**Home Section**

T4



**Introduction:**

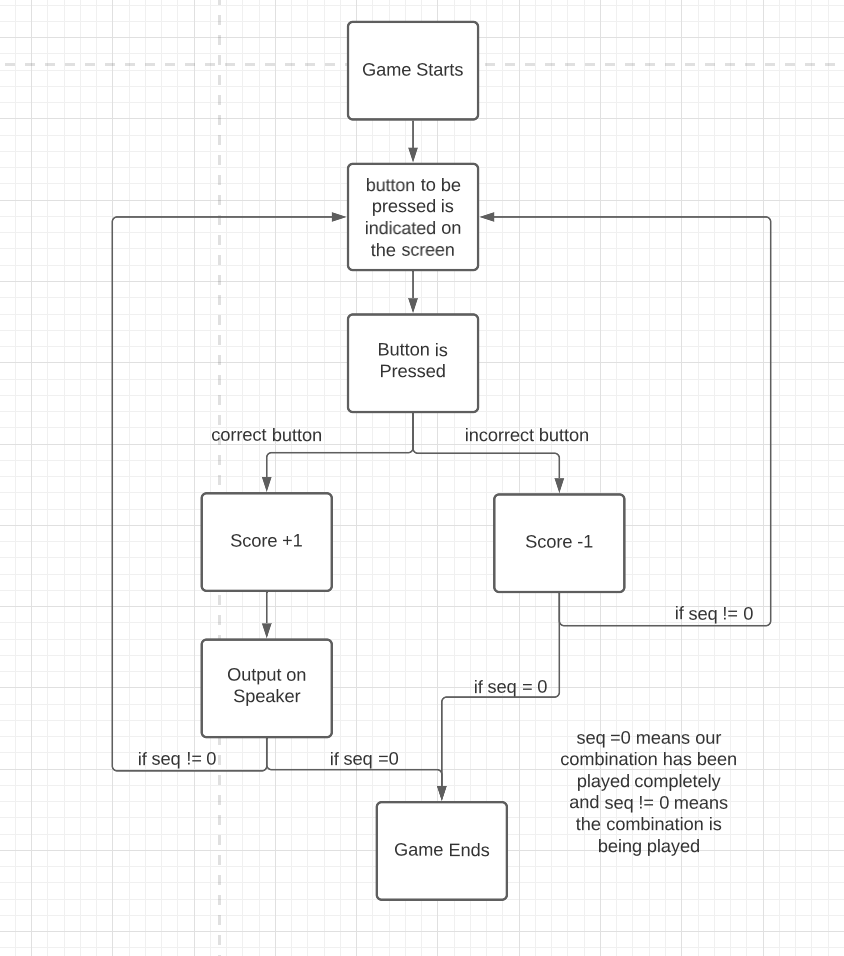
Our final project for this course was inspired by the very popular mobile game “Piano Tiles”.

Not only is this a fun game to play to pass time but it is also a good method that encourages the “practice makes perfect” ideology.

Our implementation of this game, even though it is on a smaller scale, had the same basic ideas and concepts. The player will try to play the stored sequence in the order that is indicated on the screen. Our implementation of this game is a 4 push button input with audio and visual feedback on the Digilent Basys 3 FPGA. We made use of digital logic and design concepts and tested our skills in Verilog.

Link to the GitHub repository: <https://github.com/rida-fatima03/PianoTiles.git>

**User Flow Diagram:**



How to play?

A combination is stored and the user has to play that combination in the order. The button to be pressed will be indicated on the screen and the user will press one of the four buttons whichever corresponds to the indicated block on the screen. If the correct button is pressed a score is added and the corresponding frequency is played on the speaker.

**Module Overview:**

1. Input Module: we used 4 push buttons to take input from the user.
2. Sound Module: we used a speaker that we connected to our FPGA for this. Corresponding sound to the input will be played.
3. Score Module: if the correct button is pressed score is added.
4. VGA Module: A small piano is shown on the screen. Indication for which key to be pressed will be shown.

**Module Block Diagram:**

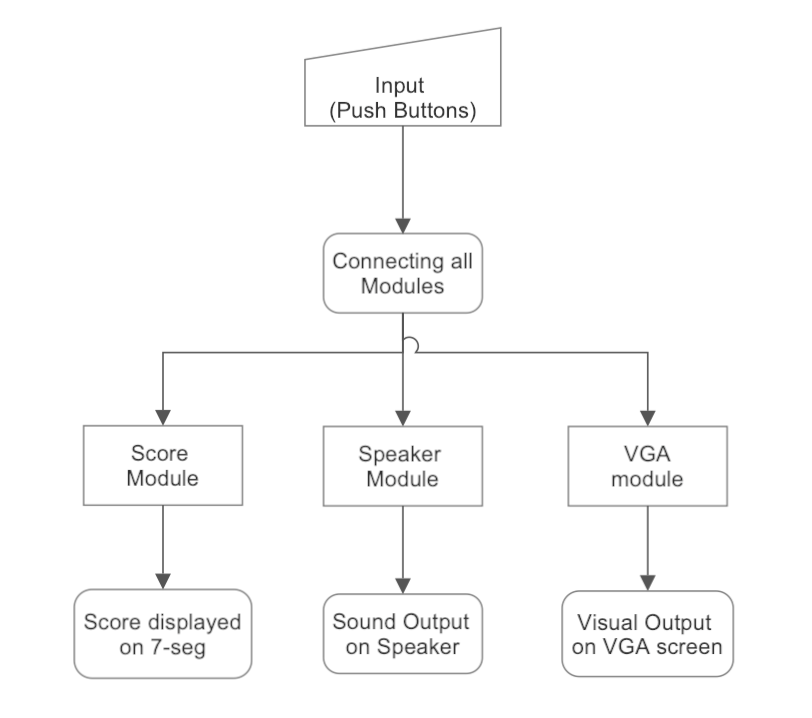


Fig.1.1: Block Diagram for the Modules.

**Progress:**

The first thing we did was write the Verilog code to integrate the push buttons with our FPGA.

We did however encounter a small problem that we are currently trying to find the solution for. Pressing the bush buttons switches off the LED on the FPGA but we need the opposite of that to happen.

Our second step was to write the Verilog code for our VGA display to show 4 keys of the keyboard. We have accomplished this.

Our third step was to write the Verilog code for our Speaker Module that outputs frequencies specific to the push buttons on the FPGA. We have also accomplished this.

Our final step at this stage was to integrate the pushbuttons and the speaker module. But due to the issue we are facing with the LED being switched on when the buttons are not pressed our speaker outputs the frequencies of all the buttons. But we have tested and we know that the correct frequency is being generated.

Once we fix this small issue we can move onto integrating the VGA screen with our speaker module and then the final step will be to integrate the score module.

We have the code from the lab for the seven segment display. We just need to modify it to cater to the needs of our project.

Code for everything we have accomplished till now can be found in the github repository.

**Input Block:**

Since we are using external push buttons we designed a small external circuit using a breadboard, four push buttons, 4 resistors and some connecting wires to connect those push buttons to the FPGA.

The pin configuration is as follows:

Since we have 4 push buttons that we need to connect to the FPGA we are using one of the four PMOD ports. On our breadboard One end of the first button is connected to JC4:P18, the second button to JC3:N17, the third button to JC2:M18 and the fourth button to JC1:K17. The other end of all four buttons is connected to a one kOhm resistor each and that is connected to the JC5:GND pin. For the reset button we are utilising one of the pushbuttons on the FPGA.

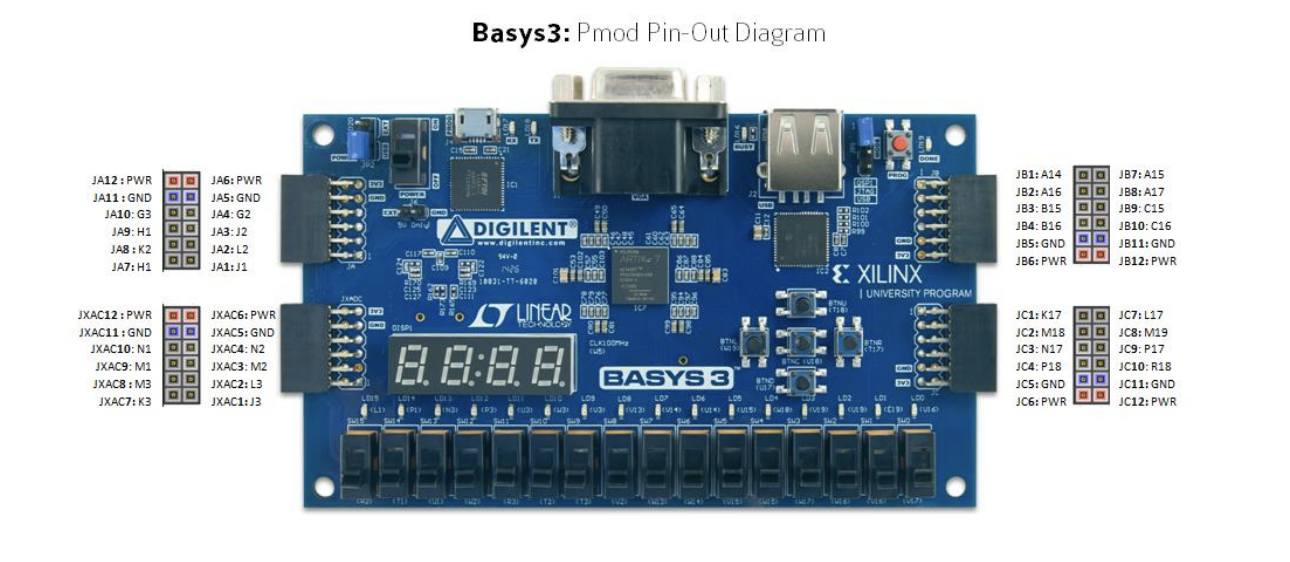


Fig 2.1: Basys3 Pmod Pin-Out Diagram

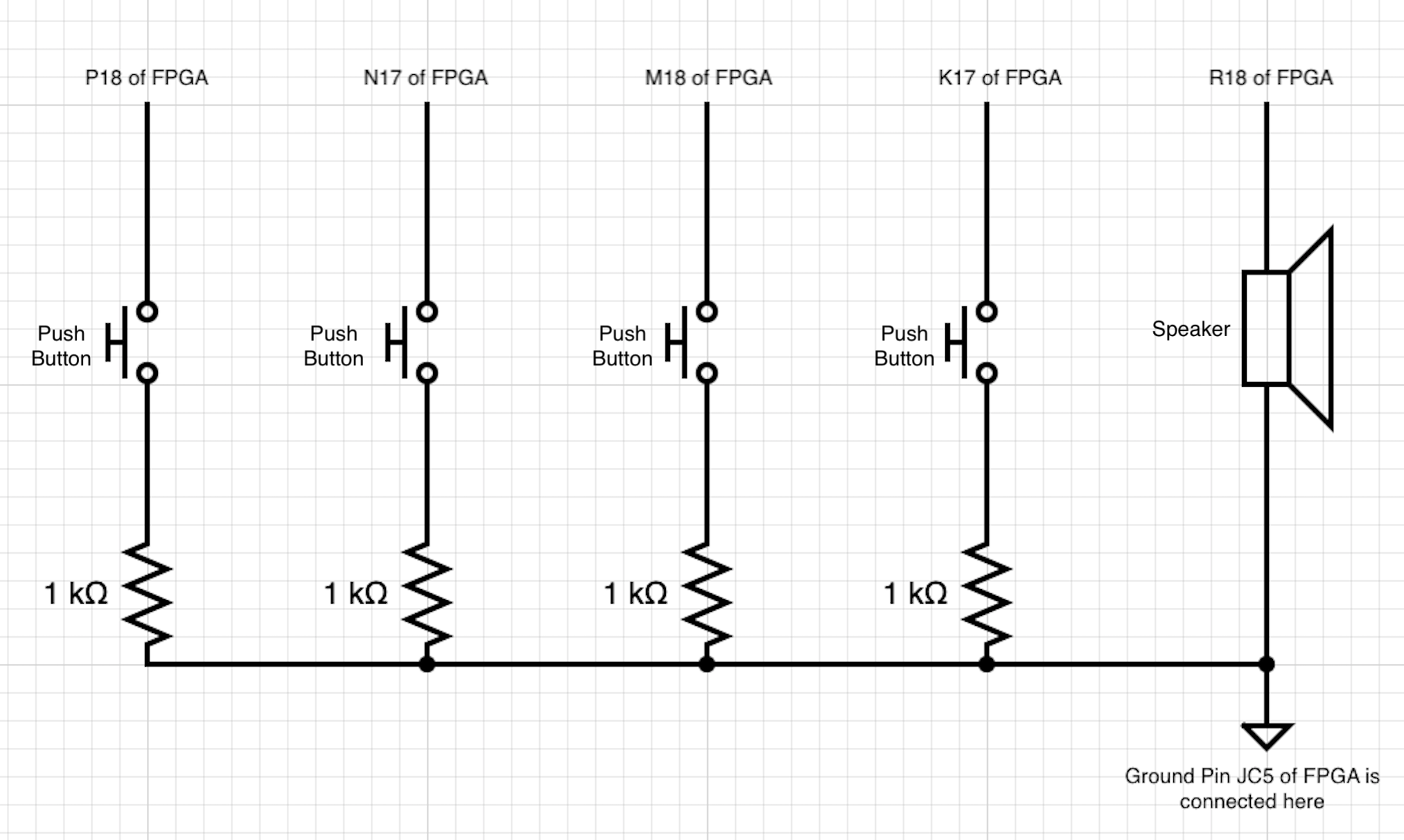


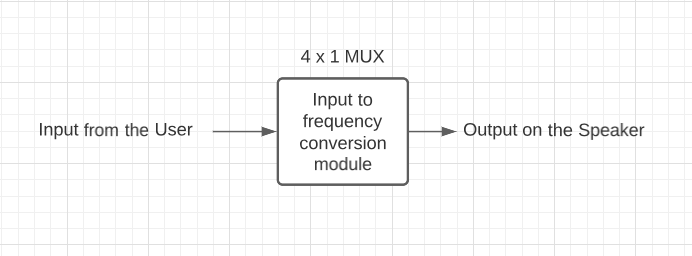
Fig 2.1: circuit diagram for our input circuit. This circuit is made on a breadboard and the pins are connected to the FPGA as shown in the circuit.

**Output Block:**

We will be dividing our Output block into 3 parts because we have 3 types of outputs.

1. Sound Module:

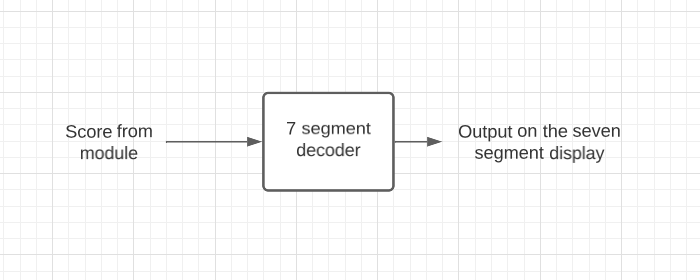
Each button has a specific frequency assigned to it. When the input is high the sound is played on the speaker. We have 4 timers that work on clock frequencies and will then generate the frequencies assigned to them. If no button is pressed the frequency output will be 0. We have 2 modules for this part of the implementation. The Timer module and the Piano Module. The timer module deals with the frequency whereas the piano module deals with the output and input. We tested our speaker and we found out that if we pass a square wave it will generate frequencies hence we do not need an amplifier or any other external circuit to make our speaker work. We can simply connect our speaker to the fpga using connecting wires and use our verilog code to generate frequencies that will be the output.



2. Score Module:

We will display the score of the player on the 7 segment display. We have not started working on this for now as this is going to be the last step of our implementation right after the integration of everything else.

The player will have a point added for every correct button press. This will be implemented using a binary adder code which we have done in the labs.



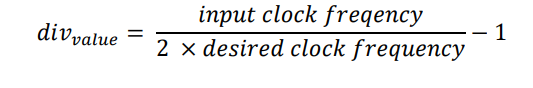
3. VGA Module:

For this part we modified the checkerboard exercise from lab 11. We modified the pixel\_gen code to get four keys shown on the display.

The modules used are as follows:

**clk\_div:**

This module has one input clk, and one output clk\_d. In this experiment, the clk input will take the default clock (of 100 MHz) and we will slow it down using the formula below.



**Reason for slowing the clock down:**

Basys3 board uses 100MHz clock (𝑇𝑖𝑚𝑒 𝑝𝑒𝑟𝑖𝑜𝑑 = 1/(100 × 10^6) = 10𝑛𝑠) and the counter output is supposed to be updated at every positive edge of clock. While driving VGA output, we require a clock frequency of 25MHz. Therefore, we need to slow down the process by dividing the clock.

**h\_counter**: The **h\_counter** module has the following features

* Input: 1-bit **clk** signal
* Outputs: 10-bit **h\_count** signal and 1-bit **trig\_v** signal.
* Increment the counter output (**h\_count**) at every positive edge of clock signal (**clk**) and counts up to **799**.
* The counter resets to **0** if the value exceeds **799**.
* **trig\_v** is set to **1** when counter resets.
* **trig\_v** would be **zero** if count is greater than **1** after reset.

We used the code given on LMS for this module.

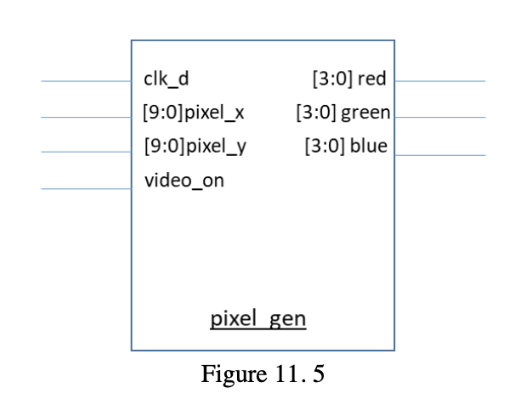
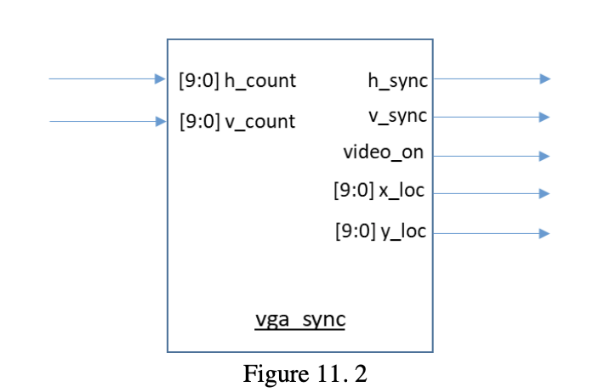
**v\_counter:** The **v\_counter** module has the following features

* Inputs: 1-bit **clk** and **enable\_v** signal
* Outputs: 10-bit **v\_count** signal
* Increment the counter output (**v\_count**) at every positive edge of the clock signal (**clk**) and count up to **524**, only if the enable\_vsignal is **1**.
* If the enable\_vsignal is **low**, the counter does not increment, but only maintains its current state.

We used the code given on LMS for this module.

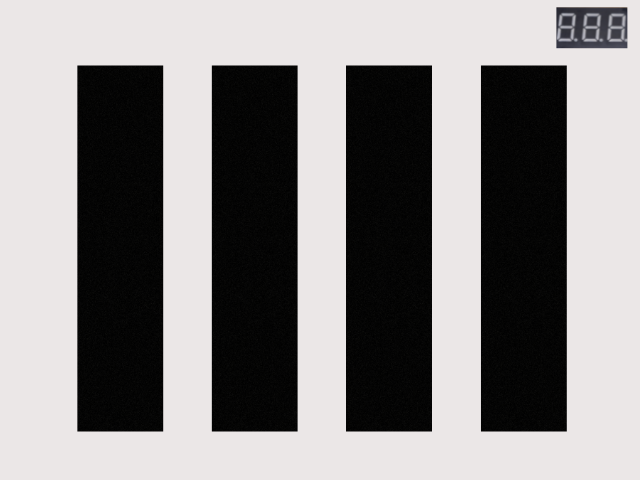
**vga\_sync**: The **vga\_sync** circuit generates timing and synchronisation signals. It takes **h\_count** and **v\_count** as input and generates five outputs **h\_sync**, **v\_sync**, **x\_loc**, **y\_loc** and **video\_on**. We used the code given on LMS for this module.

* To control the monitor's transverse horizontal and vertical scans, the **h\_sync** and **v\_sync** signals are connected to the VGA connector.
* The **video\_on** signal indicates whether the current targeted pixel is in the displayable region. The **video\_on** signal is high only when the **h\_count** issmaller than **640** and **v\_count** is smaller than **480**.
* The **x\_loc** (the location of pixel x) **y\_loc** (the location of pixel y) signals. The **x\_loc** and **y\_loc** specify the location of the current pixel and can be obtained from **h\_count** and **v\_count**.

(figs from lab11)

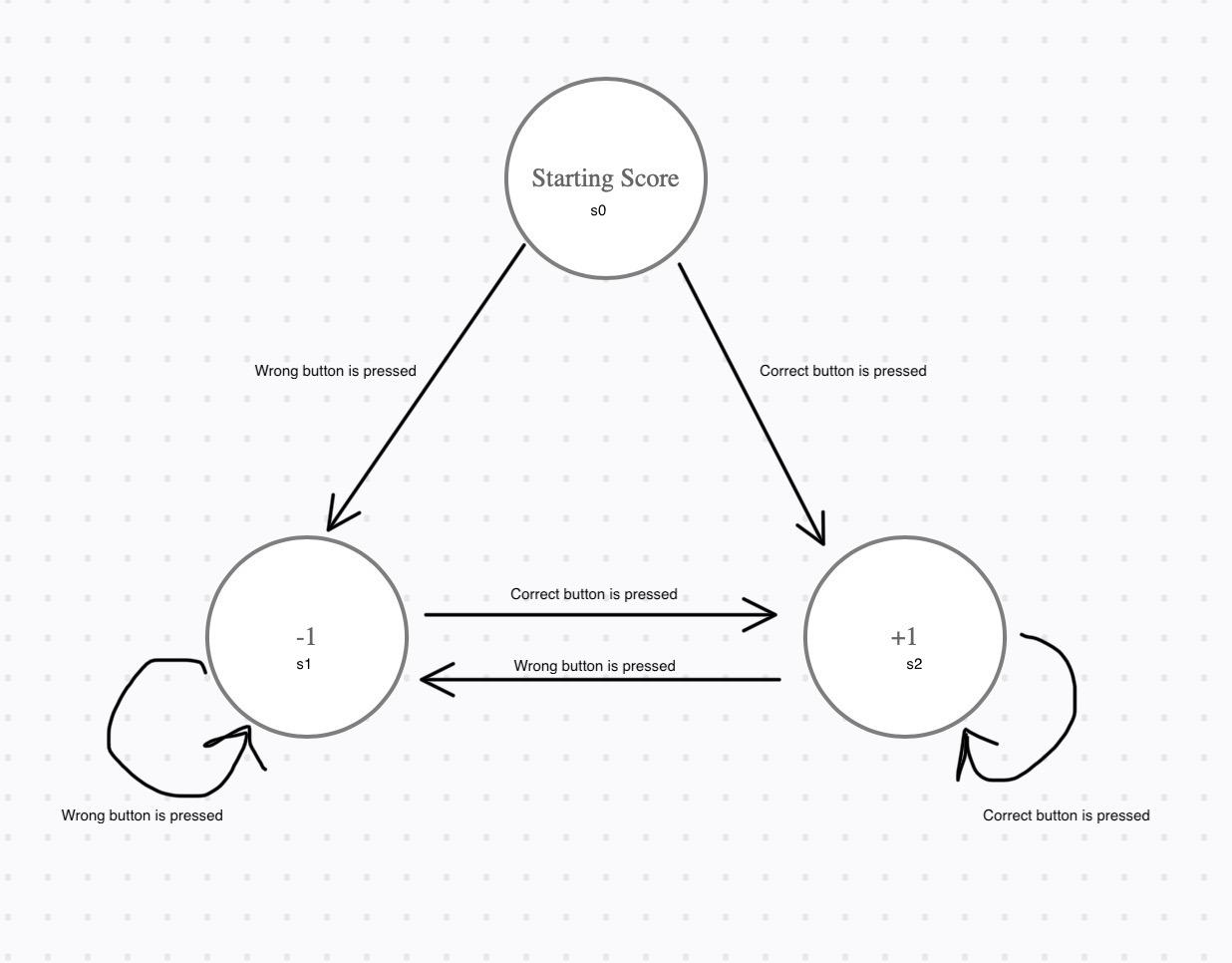
**pixel\_gen:**

This module has one input **clk\_d** (of 25MHz), two 10-bit input **x** and **y** and one input **video\_on**. This module generates three 4-bit signals of Red, Green and Blue color. We used multiplexer to create the black and white setting for our piano tiles (as shown in the figure below). However, we are still working on the seven segment display to display the score on the screen.

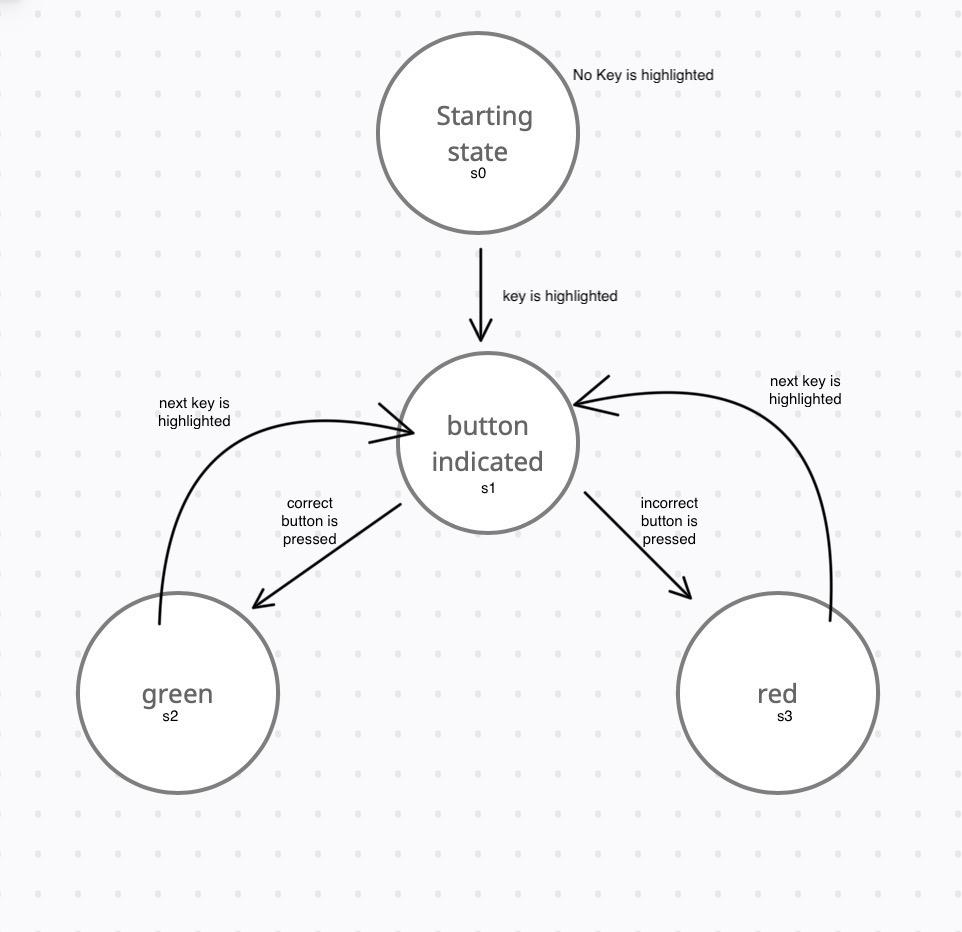


**Control Block:**

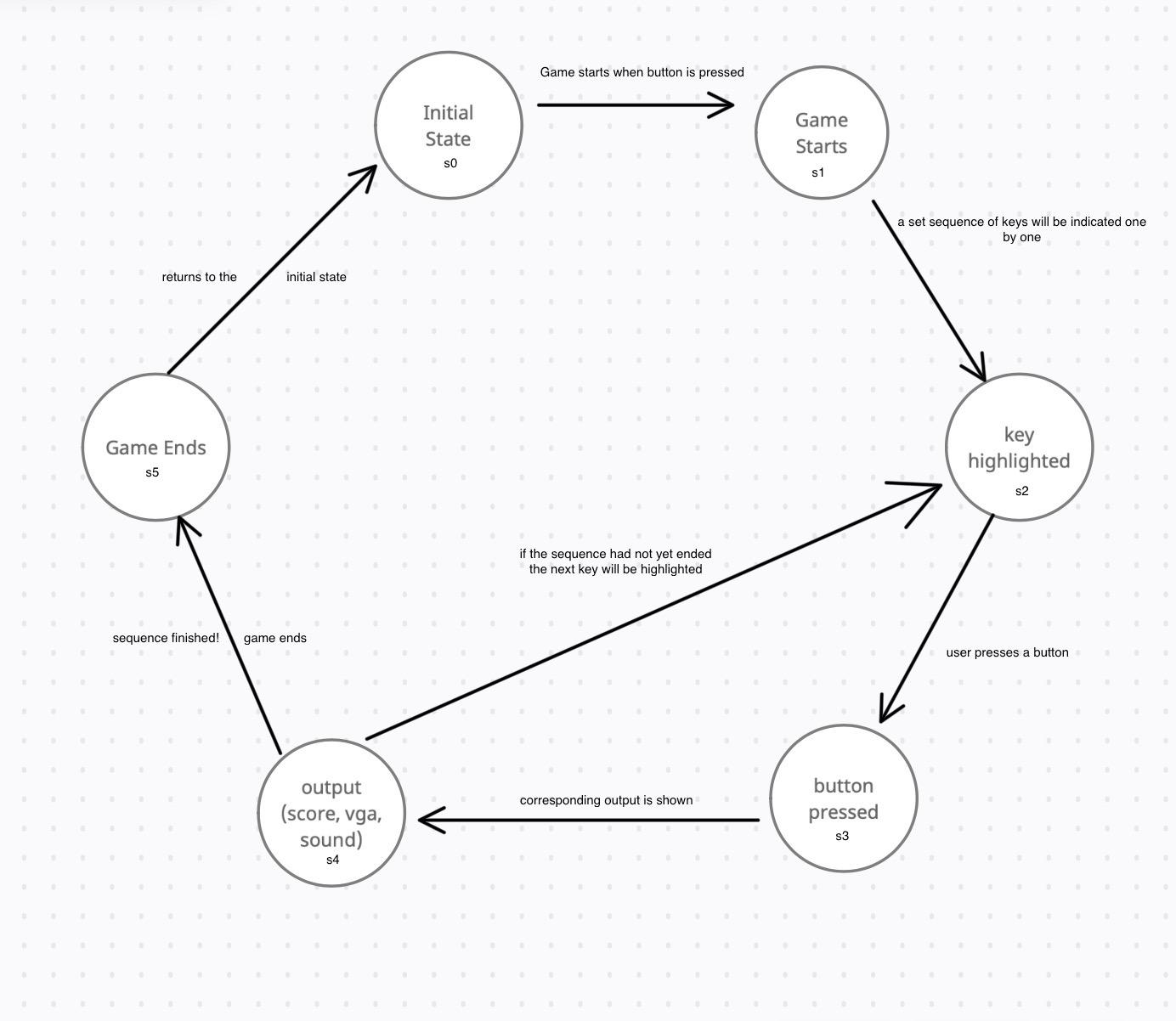
**FSM for Score:**

****

**FSM for VGA:**

****

**MAIN FSM:**

****

**Point to be noted: if at any point the reset button is pressed the game resets i.e. returns to state s0 (the initial state).**

References:

1- <https://digilent.com/reference/_media/basys3:basys3_rm.pdf>

2-**“Projects:Fpga\_piano [ca Wiki].” *Home | Olin College of Engineering*, https://wikis.olin.edu/ca/doku.php?id=projects:fpga\_piano. Accessed 20 Nov. 2022.**

<https://wikis.olin.edu/ca/doku.php?id=projects:fpga_piano>

3- **Kizheppatt, Vipin. *Design of a Simple Piano | Declaring Constants in Verilog | Module Parameterization | Part - 20*. YouTube, 23 Oct. 2020,** [**https://www.youtube.com/watch?v=ydDgKsfWNgI**](https://www.youtube.com/watch?v=ydDgKsfWNgI)**.**

4- **“Fpga4fun.Com - Music Box.” *Fpga4fun.Com - Where FPGAs Are Fun*, https://www.fpga4fun.com/MusicBox.html. Accessed 20 Nov. 2022.**

<https://www.fpga4fun.com/MusicBox.html>

5- DLD Labs