Brain Drain

How far can you make it?

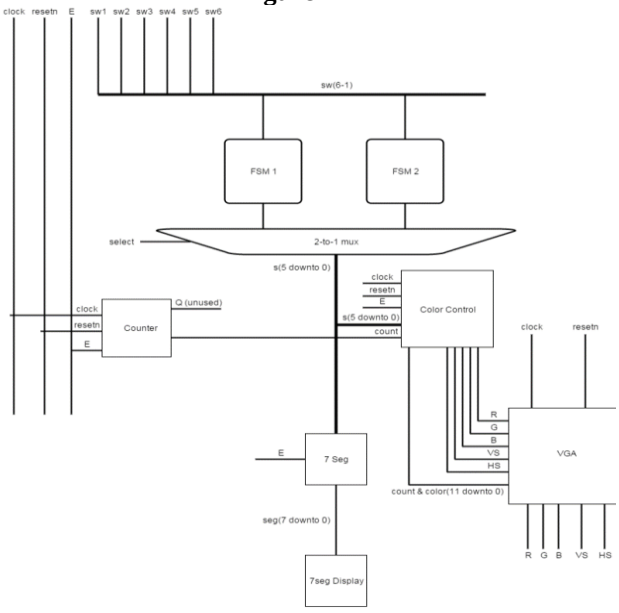
Robert McInerney, Max Manley, Zarif Ghazi
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
ramciner@oakland.edu, mmmarley@oakland.edu, zrghazi@oakland.edu

***Abstract*——the purpose of our project is to test the memory of the participants. While crafting and concocting the different parts of our project, our group had realized that we were able to separate the two different games in our project by creating two finite state machines, and the way to access these different FSMs was to dedicate it to a switch on the board. The combinations of colors that were displayed on the monitor differed based on which game was being played. If we were to perform the project again, we would definitely have considered adding more games (FSMs) in our program to make the game even more challenging.**

I. Introduction

The motivation behind the game Brain Drain was to implement all the different components of digital circuitry that we had learned about in class and utilize them in an effective and creative manner. Our Project utilizes finite state machines, a counter, a 2-1 multiplexer, a color control module, and a VGA module. Each of these modules were talked about during class, other than the color control module which was constructed ourselves.

Figure 1



II. Methodology

A. Methodology for Finite State Machines

One of the main aspects of the project, which is also responsible for creating the "meat" of the game, is the finite state machine. At its core, the game is quite linear. The game will enter the first level, and then progress through a finite amount of levels in order to reach the endgame state. Given the linear, progressive nature of finite state machines, they were an obvious choice to handle this part of the game. The first problem that was tackled was the amount of levels the game has. This is accomplished by the amount of states that were put into the finite state machine.

Figure 2

```
Transitions: process (resetn, clock, switch1,switch2,switch3,switch4,switch5,switch6)
begin
   if resetn = '0' then
      x <= S1;
   elsif (clock'event and clock = '1') then     -- Sequence = 1,1-2,1-3,1-6, 4-5-6-2,2-3-6
      case x is
         when S1 =>
            if switch1 = '1' then x <= S2; else x <= S1; end if;
         when S2 =>
            if switch5 = '1' then if switch2='1' then x <= S3; else x <= S2; end if;end if;
         when S3 =>
            if switch1 = '1' then if switch3='1' then x <= S4; else x <= S3; end if;end if;
```

As shown in figure 2. Each state in the FSM corresponds to a level in the game. For example, the game we created had eight levels, thus our FSM had eight states. If we wanted to increase or decrease the amount of levels, we would only need to change the amount of states that the FSM has. The next problem that the FSM takes care of is the sequence of colors. The colors themselves are generated by the color changer module, which develops a twelve bit number and sends the number to the VGA to display a color, but the FSM is how the game knows that the correct color or sequence of colors was inputted by the player. The FSM was built with six inputs, each input corresponding to one of six unique colors being displayed on the VGA. The FSM then begins in state one and requires the player to match the single color being shown on the VGA. If the player flips the correct switch up, the FSM will recognize this and move on to the next state, or level. This is the basis behind how the game knows when the player has passed the level. The FSM is checking for a certain input or string of inputs, which correspond to the colors on the level. Once the FSM detects the appropriate inputs, it enters the next state of level. The final problem that the FSM solves is one relating to the

color changing module. In order for the color changing module to know which color or sequence of colors to display, it requires some sort of input. This input comes from the output of the FSM. The output of the FSM is state dependent. This is necessary because the game needs to leave the FSM, display on the VGA, then come back into the FSM at the correct state. This is achieved through the output signal "s" in the FSM.

Figure 3

```
Outputs: process (x)
begin
   case x is
      when S1 => s <= "000000";
      when S2 => s <= "000010";
      when S3 => s <= "000011";
```

As shown in figure 3, each state in the two FSMs has a unique, six bit output called "s". Usually, when using two or more FSMs, the output signal "s" would be the same for all FSMs. This approach does not work in our game because the color change module needs to know not only which state the FSM is in, but also which FSM is passing the state. This is because if "s" was the same for FSM1 and FSM2, then the sequence of colors would always be the same for each state machine, but we wanted each state machine to have unique sequences. By assigning a unique value for "s" to state 1 in FSM1 and state 1 in FSM2, the color changing module can determine which FSM is being used, and therefore display the correct sequence for that specific game.

B. Methodology for The Color Control Module

A key component of Brain Drain is displaying the colors that the user needs to input. The finite state machine will require a series of switches to be activated in order to move on to the next level. These switches are given colors and flashed at the player consecutively. The variable "s" mentioned in the methodology of the finite state machines is given as in input to the color control. Each "s" signal is specific not only to the state but the finite state machine as well. This number is concatenated with the input from the counter. (See figure 3) The counter was needed in order to show multiple colors to add complexity to the game. The reason for concatenating the two values is to change the color every second according to the state and the counter value. A 12-bit number is then given to the VGA module to display various colors.

Figure 4

```
game_color <= count & s_hold;
   with game_color select
      color <=  "101000001100" when "00000000",--fsm1 s1
                "101000001100" when "01000000",
                "101000001100" when "10000000",
                "101000001100" when "11000000",

                "000001100000" when "00000010",--fsm1 s2
                "000001100000" when "01000010",
                "101000000000" when "10000010",
                "101000000000" when "11000010",

                "101000001100" when "00000011",--fsm1 s3
                "101000001100" when "01000011",
                "111111111111" when "10000011",
                "111111111111" when "11000011",

                "000000001100" when "00000100",--fsm1 s4
                "000000001100" when "01000100",
                "000000000000" when "10000100",
                "000000000000" when "11000100",
```

C. Methodology for The Seven Segment Display

An interesting detail to our game is how we utilized the seven segment display to tell the user which level he or she is in. The way this works is that it receives input in the formation of a six bit number (input "s") and then that number correlates to what digit is displayed as a result of bit –to-bit representation. The coding for this following process can be seen in figure 5.

Figure 5

```
architecture Behavioral of SevenSeg is

signal LED: STD_LOGIC_VECTOR(6 downto 0);

begin

with y select

   LED <=   "0110000" when "000000",
            "1101101" when "000010",
            "1111001" when "000011",
            "0110011" when "000100",
            "1011011" when "000101",
            "1011111" when "000110",
            "1110000" when "000111",
            "1111111" when "001000",
            "0110000" when "110001",
            "1101101" when "110010",
            "1111001" when "010011",
            "0110011" when "010100",
            "1011011" when "010101",
            "1011111" when "010110",
            "1110000" when "010111",
            "1111111" when "011000",
            "1001111" when "111100",
            "1001111" when "111110",
            "1111111" when others ;
```

D. Methodology for The VGA Module

To implement Brain Drain a VGA display was needed. We utilized the code given by the professor during class for the VGA display. All we needed to do was input a 12-bit number into the module to represent each color. The color control module did the rest of the work by changing the 12-bit number based on the counter.

III. Experimental Setup

The project was implemented using Xilinx 14.7, a VGA display and a Nexys 4 board. Using Xilinx we were able to run a test bench with simulated values in order to check functionality of the system prior to implementation on the Nexys board. Once the test bench had ben ran we used the Nexys 4 board and a VGA display to test the program. Each version of the game was ran through in order to make sure that no state of the game would represent a failure.

IV. Results

Our program gave us the exact results that we intended. When each switch was placed in the proper position based off of the colors displayed to the user the program would enter the next level. After completing every level the program would go into an end game state and display a grey screen. To restart the game you would need to click the reset button on the board. The program would then enter level one again and you could choose which game to play.

This project can be related to the material we learned in class in various ways. One of these ways is that we used a multiplexor to decide which game we were in. Finite state machines were used to implement our project as well that were gone over in length during class. Finally, a counter was also used during one of our lab sections in order to solve a problem so we were able to utilize the same methodology in our current project.

Conclusions

After completing this project, our group has three main take-away points heading into future projects and classes. The first of these is the sheer versatility of finite state machines. Finite state machines have a mind-boggling array of uses, whether it is creating a game as such, or using a FSM to model a multiplexer, or using a FSM to represent a set of complex rules and conditions, while choosing which inputs the user would like to use. If we wanted to, we could have used a FSM to model the two to one multiplexer in our project. The second main takeaway is that VGA displays use a twelve bit number to generate a color which is being output onto the screen. There are four pins on the VGA cord dedicated to red, four pins dedicated to blue, and four pins dedicated to green. Each one of these pins is assigned a Boolean value of “1” or “0”. For example, the color red is output to the VGA by the twelve bit number “101000000000” but a different shade of red could be sent with “110000000000”. The last main takeaway is that counters can be used in many different applications in order to modify an output over time. In our case, the counter was used to modify which colors were being displayed on the VGA every one second, but there are also other uses. Other uses include using a counter to tell a register when to accept new data or hold previous data, or using a counter to change the output of a seven segment display over time.

Our project has a couple of issues that remain unresolved. The first of which is removing the picture of the android from the VGA display. Another issue is that the game has no explicit game over state if the player fails the input the correct colors. If a game over state was added by adding several more “if” statements to each FSM, it would be a major improvement. Another possible improvement is adding more games by adding more finite state machines, as well as adding variable difficulty by making some FSMs have sixteen states or some FSMs accept more than six inputs and therefore the player is required to memorize more than six colors. Overall, the project was extremely rewarding and educational for all three members. We further cemented our knowledge and control over finite state machines, as well as acquiring deep knowledge about the inner workings of the VGA display.