Project 2 Report

INTRODUCTION

Sequential circuits represent an important component of many machines that we interact with on a daily basis. One example that immediately comes to mind is a microwave oven. The sequential circuit in a microwave is demonstrated by the timer. While the timer is greater than zero, the microwave will emit its waves that hit the food being heated up. Once the timer hits zero, the microwave turns off and a sound is emitted, letting the user know that their food is ready for consumption. This simple timer mechanism can be represented with a state diagram. Each second that the timer is active represents an individual state in the state diagram. The final state would be when the timer hits zero. The final state would look something like this:

FINAL\_STATE:

1. Stop the emission of microwaves and…
2. Emit an indicator sound, letting the user know that the food is ready.
3. Set the timer to be controlled by the user’s input, n, and wait for the “start” button to initialize the internal state to the first state, which is represented by the nth second, defined by the user.

For every other state, the microwaves will still be emitted, and the food will continue to get cooked. Finally, we have a clock that decrements a counter with a frequency of 1Hz. Each time the counter gets decremented we move from one state to the next.

For my project, I will demonstrate a sequential circuit in the form of a password protected safe. If the user inputs the right password, a light will turn on and a sound will be emitted from a speaker. The light and sound will remain on until the system gets reset with a simple switch. Further, each state of this machine is represented by a specific light turning on, indicating that the correct button has been pressed.

SOFTWARE

Below is the source code of my project:

module p2(clk,seq,rst,led,st1,st2,st3,speaker);

input rst;

input clk;

input [3:0] seq;

output reg led;

output reg st1; // The light corresponding to the first correct digit of the password being // entered.

output reg st2;

output reg st3;

output speaker;

reg[2:0] curstate,nxtstate;

parameter s0 = 3'b000,

s1 = 3'b001,

s2 = 3'b010,

s3 = 3'b011,

s4 = 3'b100;

// Allow the user to reset the password entry system using a switch

always @ (posedge clk or posedge rst)

begin

if (rst == 1)

curstate <= s0;

else

curstate <= nxtstate;

end

// Check that password has been entered correctly

always @ (\*)

begin

case (curstate)

s0: if (seq == 4'b0001) nxtstate = s1;

else nxtstate = s0;

s1: if (seq == 4'b0100) nxtstate = s2;

else nxtstate = s1;

s2: if (seq == 4'b0010) nxtstate = s3;

else nxtstate = s2;

s3: if (seq == 4'b1000) nxtstate = s4;

else nxtstate = s3;

s4: nxtstate = s4;

default: nxtstate = s0;

endcase

end

always @ (\*) // The light turns on when the password has been entered correctly.

begin

if (curstate == s4)

led = 1;

else

led = 0;

end

always @ (\*) // Set up indicator lights for each state, excluding the final state.

begin

if (curstate == s1) st1 = 1;

else st1 = 0;

if (curstate ==s2) st2 = 1;

else st2 = 0;

if (curstate == s3) st3 = 1;

else st3 = 0;

end

reg[27:0] tone;

always @ (posedge clk)

tone <= tone+1;

wire[6:0] fastsweep = (tone[22] ? tone[21:15]: ~tone[21:15]);

wire[6:0] slowsweep = (tone[25] ? tone[24:18]: ~tone[24:18]);

wire[14:0] clkdivider = {2’b01, (tone[27] ? slowsweep : fastsweep), 6’b000000};

reg[14:0] counter;

always @ (posedge clk)

if (counter == 0) counter <= clkdivider;

else counter <= counter-1;

reg speaker;

always @ (posedge clk)

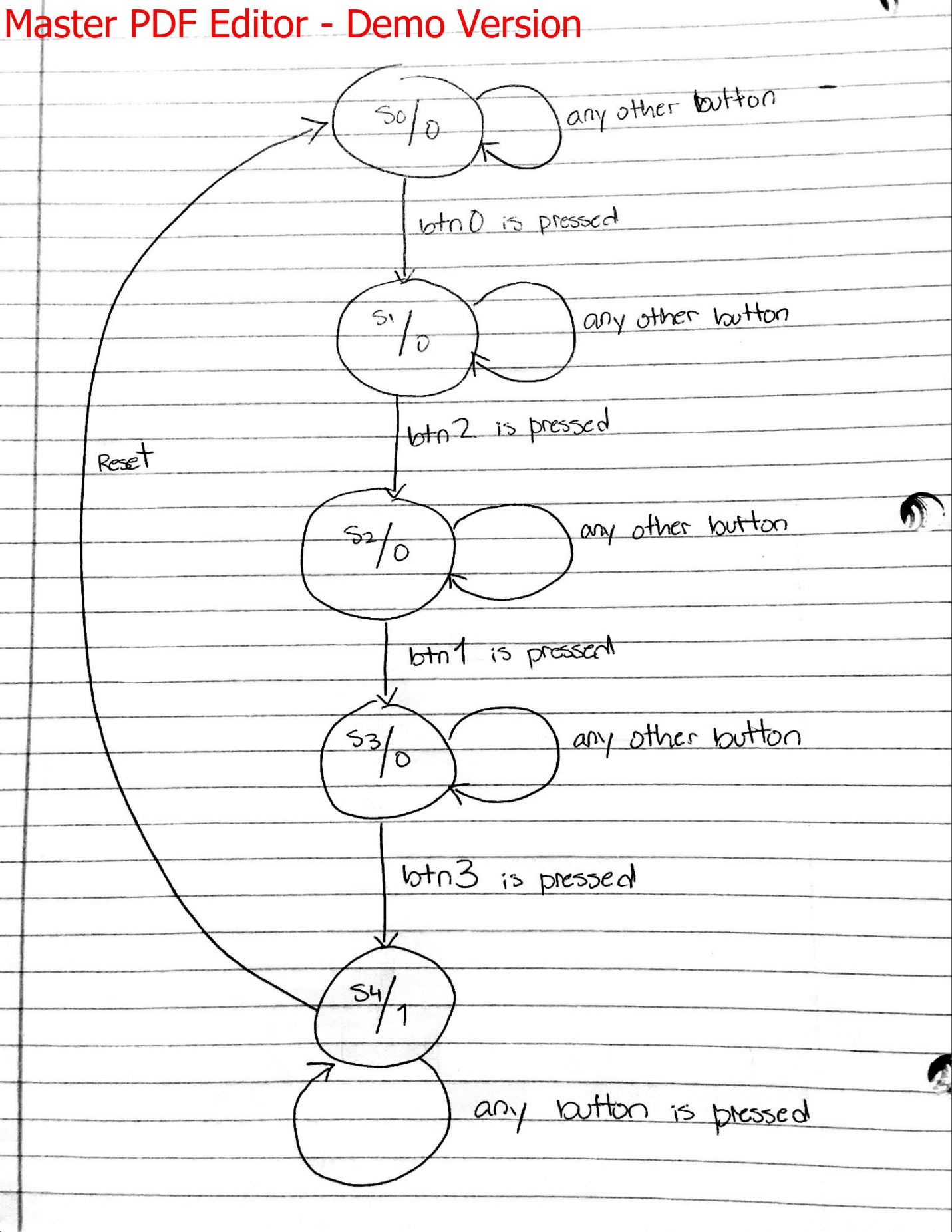
if (counter == 0 && curstate == s4) speaker <= ~speaker;

endmodule

ANALYSIS

Much of the code seen above is to output sound when the password has been entered correctly, but this is not the focus of this project. For an explanation of outputting a siren sound to a speaker, please see my first project.

The finite state machine inherent in this password checker is represented by the case statement involving s0, s1, s2, s3, and s4. Once a certain state has been reached, the system will wait for input from the push buttons on the Basys2 board. The user input is represented by “seq[3:0]” where each bit is mapped to a push button. We see that a specific sequence of button presses is required to reach the final state, s4. Once s4 is reached, the system will remain in that state until power is lost or the reset switch is activated, moving us back to the initial state where the user can try to input the password again. The following page contains a state diagram of this machine. Finally, in order to indicate to the user that a correct button has been pressed, specific lights will light up with “led” corresponding to the final state and st1, st2, and st3 corresponding the other states respectively. The “st**x**” variables are mapped to leds that occur contiguously to indicate that a new state has been reached.



SOURCES

<http://www.fpga4fun.com/MusicBox2.html> - I used this source for the Verilog source code for outputting a siren sound to a speaker.