

BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS 223: DIGITAL DESIGN

FINAL REPORT

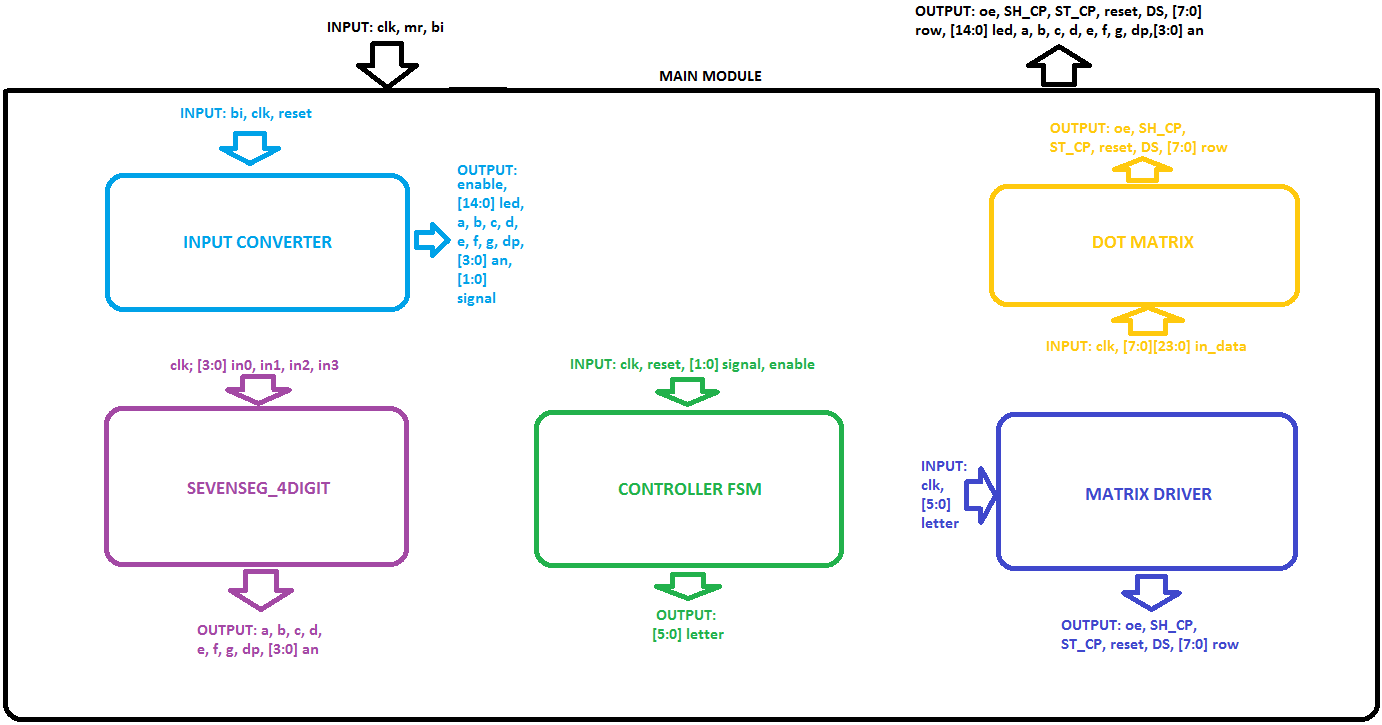
**MORSECODER**

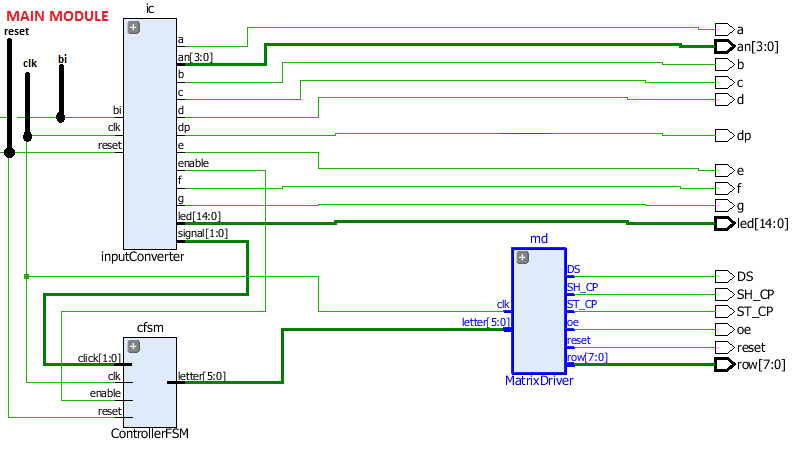
BERAT BİÇER - 21503050

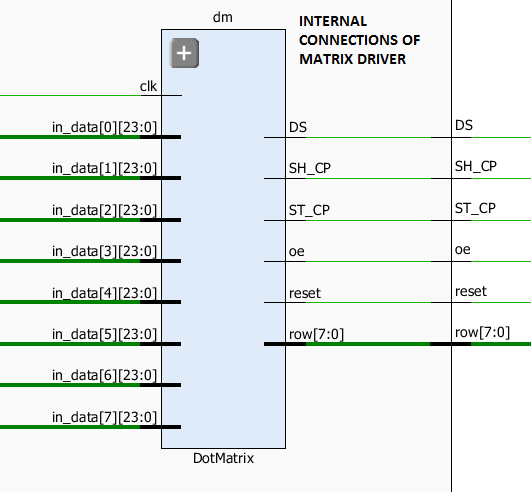
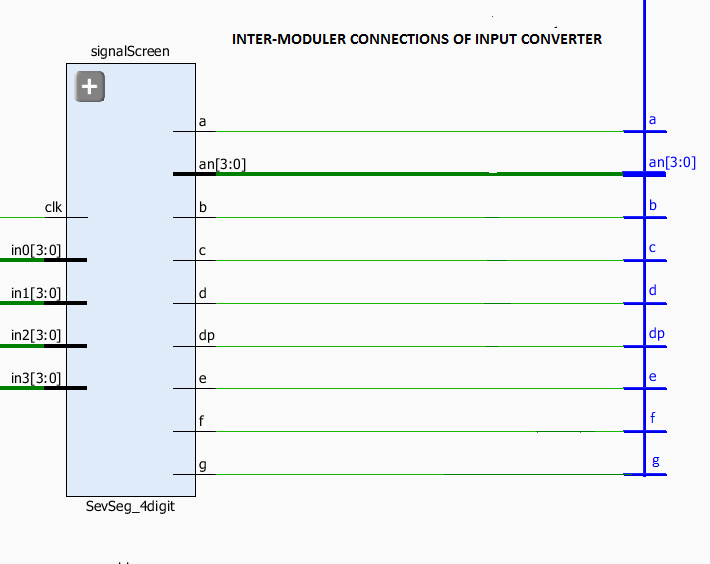
MEHMETALİ SEMİ YENİMOL – 21401565

SECTION 3

25.12.2016

GENERAL APPEARENCE

DATA FLOW



MODULE DESCRIPTIONS

MAIN MODULE

Inputs

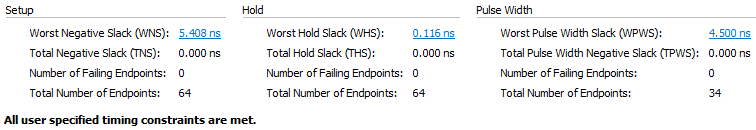
* clk: Main clock signal of the system, taken via BASYS3, 1 bit
* mr: Master Reset signal to reset the entire system, taken via switches on BASYS3, 1 bit
* bi: Input which can be either a dot, a dash or a space; taken via the middle push button of BASYS3, 1 bit

Outputs

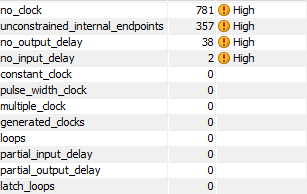
* oe: Output Enable signal of 8x8 Display Module, 1 bit
* SH\_CP: Shift Register Clock Pulse, used to enable 8x8 Display Module, 1 bit
* ST\_CP: Store Register Clock Pulse, used to enable 8x8 Display Module, 1 bit
* reset: Reset Signal, used to enable 8x8 Display Module, 1 bit
* DS: Data signal, used to enable 8x8 Display Module, 1 bit
* [7:0] row: Row signal, used to enable 8x8 Display Module, 8 bit
* [14:0] led: Used to light the leds of BASYS3, 15 bit
* a, b, c, d, e, f, g, dp: Used to light 7 Segment Display, each one is 1 bit
* [3:0] an: Used to light 7 Segment Display, 4 bit

Description of the Module

* Main driver of the system
* Controls data transfer between modules
* Inputs are given to here and outputs are taken from this module
* Main module has the main data flow within the project: It takes inputs through BASYS3, sends them to appropriate modules, takes outputs from them and sends data taken from previous modules to either another module or outputs them.
* It has 3 instances of other modules: InputConverter, ControllerFSM, MatrixDriver. InputConverter is used to translate the data which is in Morse code to binary. ControllerFSM is used to interpret the Morse code in binary form as letters or numbers. MatrixDriver takes those letters and numbers, then determines an appropriate representation so that they can be shown in 8x8 Display Module.

Timing

Clock pulse for the system is slowed down to 50 MHz, because clock pulse was unable to reach the entire system in one cycle in the base case.



INPUT CONVERTER

Inputs

* clk: Clock signal of the module, transfered from Main module, 1 bit
* mr: Reset signal of the module, transfered from Main module, 1 bit
* bi: Input which can be either a dot, a dash or a space; transfered from Main module, 1 bit

Outputs

* enable: Enable signal to signal the system that input is interpretted and data can be transfered to ControllerFSM.
* led: Used to light the leds of BASYS3, 15 bit, sent back to Main module
* a, b, c, d, e, f, g, dp: individual LED output for the 7-segment along with the digital point, sent back to Main module
* an: Data to be shown on 7 Segment Display, 4 bit, sent back to Main module
* signal: Interpreted version of the input of Main module; it is now either a dash, a dot or a space, 2 bit, sent back to Main module

Description of the Module

* Converts input which is taken from push buttons into dot-dash-space form and then to binary
* It works as a High-Level State Machine: Basicly when the button is pressed, it starts a counter. If this counter is less than a sentinel value which is 3\*107, it keeps incrementing it. After this sentinel value if button is lifted, input is interpreted as a dot signal and corresponds to 2’b01 in binary. If the button is pressed for longer periods up until another sentinel, input becomes a dash and corresponds to 2’b10 in binary. If the counter is larger than the second sentinel which is 9\*107, input becomes space and corresponds to 2’b00 in binary. When input is interpretted, it sends an enable signal back to main module so that ControllerFSM can use this input to determine the corresponding character. Thus, the customer needs to press the middle push button for approximately 1 seconds to get a dot and 3 seconds for a dash. After input type is determined, enable signal is sent to ControllerFSM module to activate it.
* This module also create outputs for using 7 Segment Display module with respect to the current state of the HLSM and input that is interpreted. Moreover, leds of the BASYS3 are used to help the customer understand how long the buton is pressed therefore what the input to the system will be interpretted.

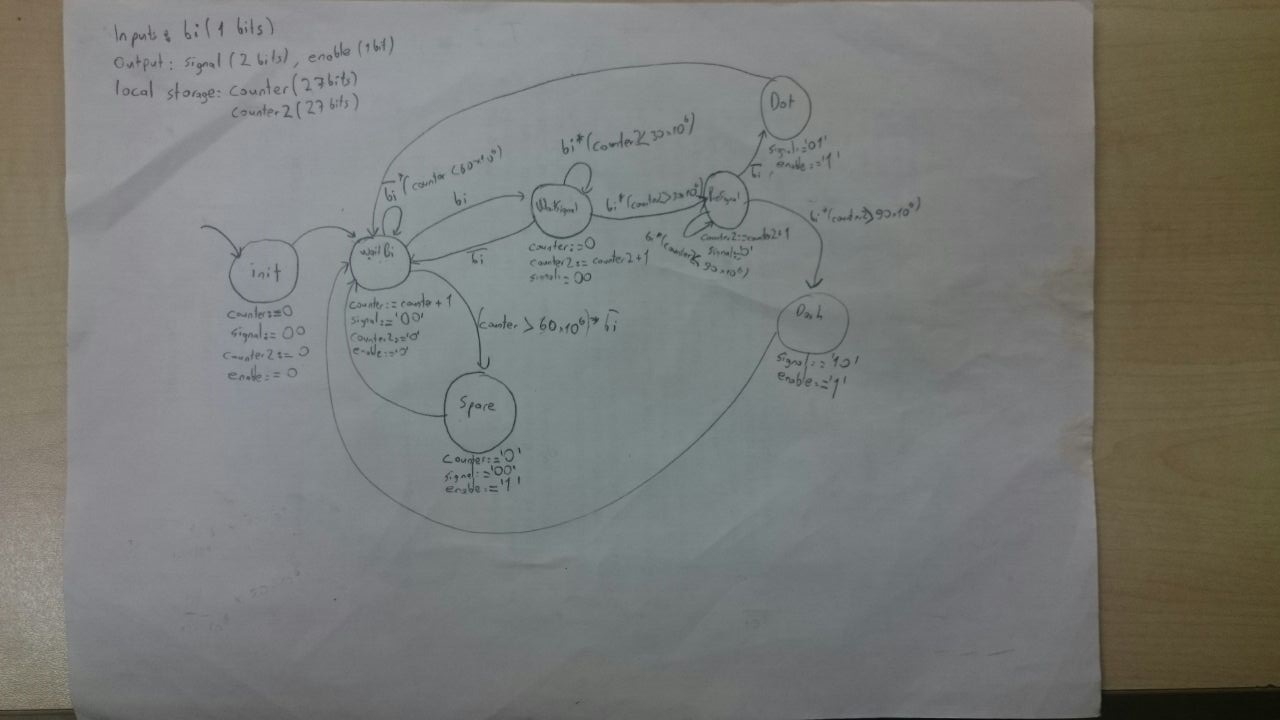
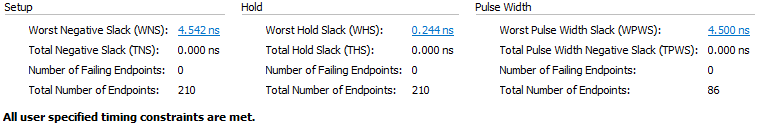
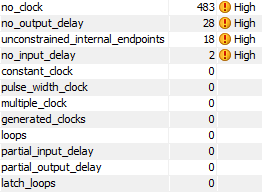


Figure 1: HLS Diagram of Input Converter

Sentinel values and therefore counters are specifically chosen so that customer can clearly see what’s happening within the system. These values can be larger or smaller according to the funtionality and requirements of various situations.

Timing





SEVENSEGMENT\_4BIT

Enables the 7 Segment display module on BASYS3. Shows the current state of the HLSM of InputConverter module and inputs that are interpreted. Also, used for debugging purposes. Check references for more information since this module does not belong to us.

CONTROLLER FSM

Inputs

* clk: Clock signal of the module, transfered from Main module, 1 bit
* reset: Reset signal of the module, transfered from Main module, 1 bit
* click: Representation of input taken from push button in binary form, 2 bit, taken from Input Converter
* enable: Enable signal taken from Input Converter so that Controller FSM can start interpretting the data in binary into English alphabet and digits

Outputs

* letter: Interpretted version of the data, sent back to Main module, 6 bit

Description of the Module

* Converts input which is taken from push buttons into dot-dash-space form and then to binary
* Works as a Finite State Machine: at each positive edge of the clock, if enable signal is 1’b1 in binary, FSM moves into the next state. States are determined with respect to the Morse code.

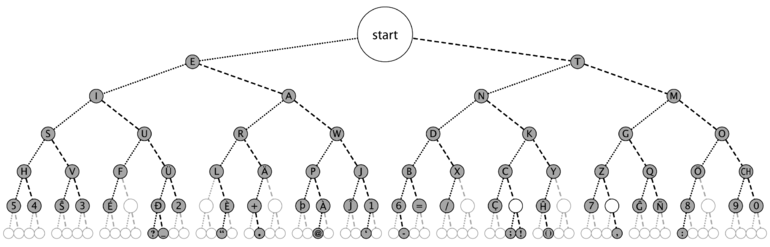
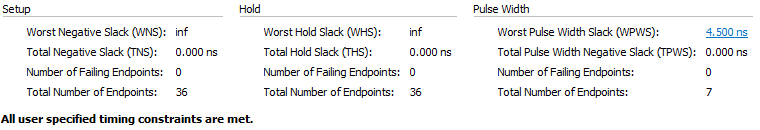


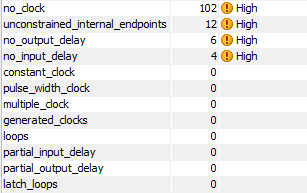
Figure 1: States of FSM with respect to Morse Code

When the input is taken if it’s a 2’b01 (dot), it goes to the left side of the current state. If it’s a 2’b10(dash), in binary, it moves to the left side of the current state. If it’s a 2’b00(space) or 2’b11(invalid), it goes to INVALID state and returns back to the initial state in the next positive edge of the clock. However, special characters and non-English symbols in the map above are not available.

When FSM moves on to the next state, it determines an output, 6-bit letter, to indicate which symbol input currently is. The reason is that when transmitting data via Morse code if you get a dot for example, it is ‘E’ unless another data is given. So, if we want to send a message of ‘P’, we need to give dot-dash-dash-dot to the FSM. However, the system does not know if we wanted to give a signal of ‘E’ until we give another input. If we give another input, only then the system moves on the the next state and waits for another input. If no input is given, it recognizes that we do not want to move further in the map and returns to the initial state.

Timing





MATRIX DRIVER

Inputs

* clk: Clock signal, taken from Main module, 1 bit
* letter: Binary representation of the characters, taken from Controller FSM, 6 bit

Outputs

* oe: Output Enable signal of 8x8 Display Module, 1 bit, sent back to Main module
* SH\_CP: Shift Register Clock Pulse, used to enable 8x8 Display Module, 1 bit, sent back to Main module
* ST\_CP: Store Register Clock Pulse, used to enable 8x8 Display Module, 1 bit, sent back to Main module
* reset: Reset Signal, used to enable 8x8 Display Module, 1 bit, sent back to Main module
* DS: Data signal, used to enable 8x8 Display Module, 1 bit , sent back to Main module
* [7:0] row: Row signal, used to enable 8x8 Display Module, 8 bit, sent back to Main module

Description of the Module

* Determines the layout of the 8x8 Display Module with respect to the character taken as input, ”letter”.
* It determines the required inputs to be able to use 8x8 Display Module. It first determines which rows will be online, then defines a 24 bit data for each row(namely a message). For example, if we want to show a ‘E’ on the display module each row must be active and messages for each row are as follows:

24’b11111111\_00000000\_00000000, 24'b11111111\_00000000\_00000000, 24'b11000000\_00000000\_00000000, 24'b11111111\_00000000\_00000000, 24'b11111111\_00000000\_00000000, 24'b11000000\_00000000\_00000000, 24'b11111111\_00000000\_00000000, 24'b11111111\_00000000\_00000000.

After this process is complete, it stores this data and sends it to the DotMatrix module so that 8x8 Display Module becomes online. By defining the data above for each input “letter”, we choose in which color should a cell be lightened as we wish.

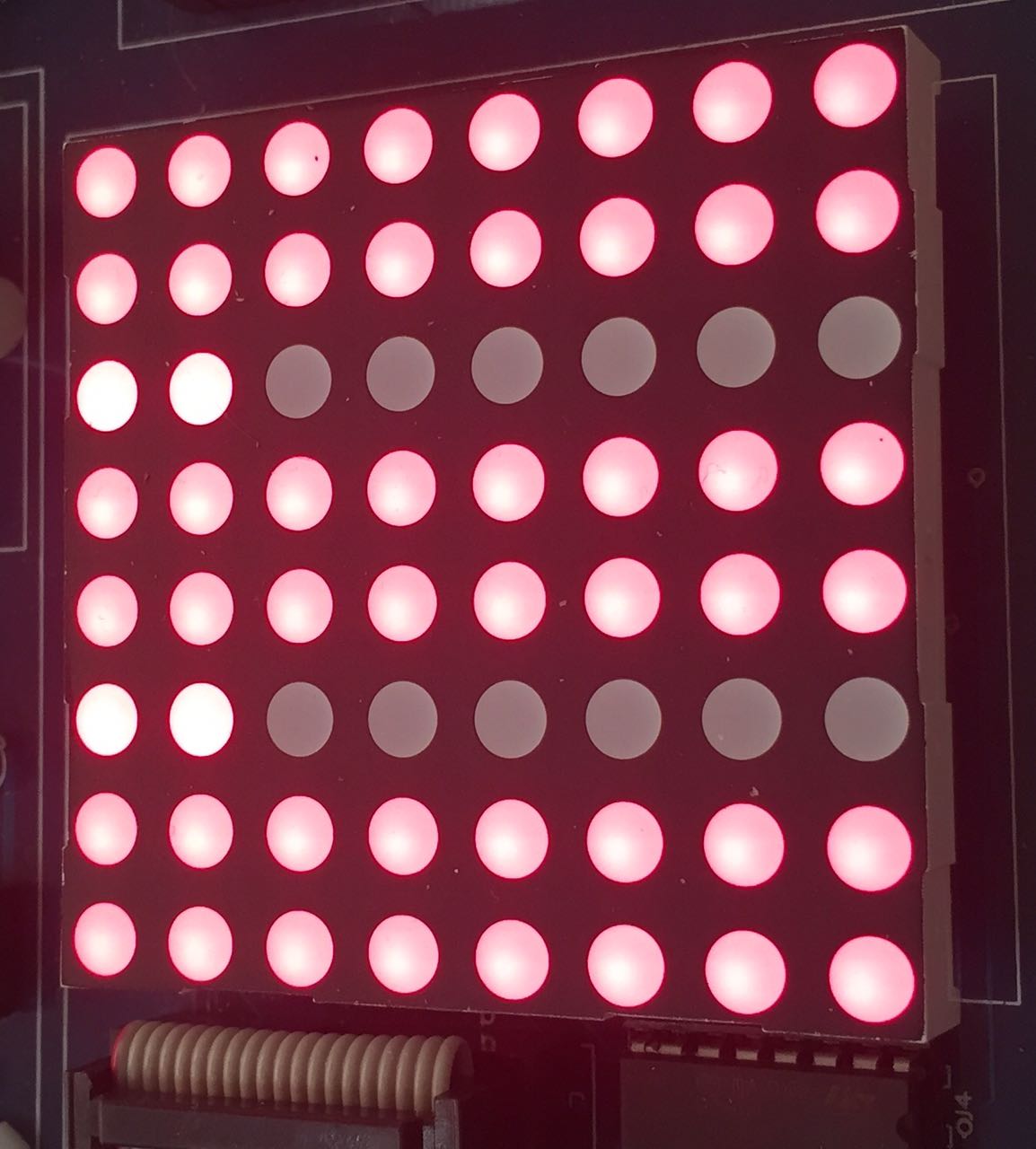
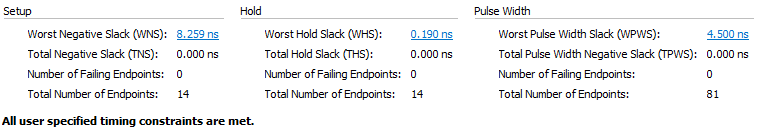
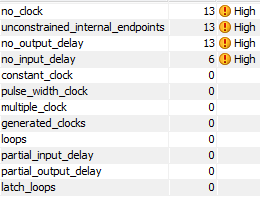


Figure 2: Letter 'E' on 8x8 Display

When the outputs are taken from module DotMatrix, this data is sent back to Main module and given to the 8x8 Display Module via cables.

Timing





DOT MATRIX

Inputs

* clk: Clock signal, taken from Main module, 1 bit
* in\_data: Stores data taken from MatrixDriver module, 8x24 bit array

Outputs

* oe: Output Enable signal of 8x8 Display Module, 1 bit, sent back to MatrixDriver module
* SH\_CP: Shift Register Clock Pulse, used to enable 8x8 Display Module, 1 bit, sent back to MatrixDriver module
* ST\_CP: Store Register Clock Pulse, used to enable 8x8 Display Module, 1 bit, sent back to MatrixDriver module
* reset: Reset Signal, used to enable 8x8 Display Module, 1 bit, sent back to MatrixDriver module
* DS: Data signal, used to enable 8x8 Display Module, 1 bit , sent back to MatrixDriver module
* [7:0] row: Row signal, used to enable 8x8 Display Module, 8 bit, sent back to MatrixDriver module

Description of the Module

* Sends appropriate signals to 8x8 Display module to make it online.
* Since the 8x8 Display Module has 8 rows and each row has a 24 bit data, for each frame, we need 192 bit data in total. This data is taken from MatrixDriver module.

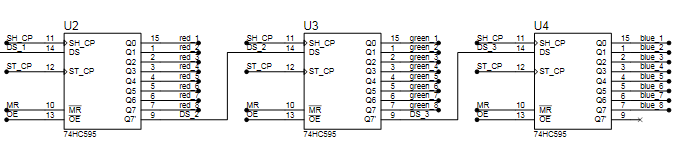
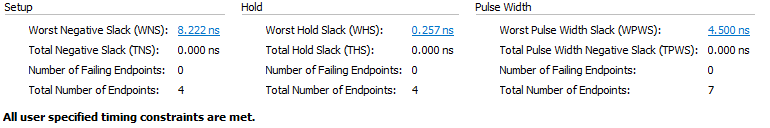
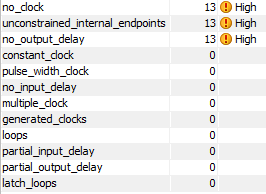


Figure 3: Internal System of 8x8 Display Module

For each row, there are 3 8-bit color shift registers: red, green and blue respectly, which at each clock cycle, they shift one bit of the data in serial in-parallel out format. To be able to light the 8x8 Display module, in first 24 cycles, we need to send data to the first register so that at each cycle, data can be shifted through the last register. Therefore, for the first 24 clock cycles, we send a clock signal, SH\_CP, and a 1 bit data, DS, at each cycle: shiftRed, shiftGreen, shiftBlue respectly. In the meantime, we need the store registers and output registers within color registers be offline so that we can see a full frame. Thus, in the first 24 clock cycle, clock for the store registers, ST\_CP, is 1’b0 and enable signal, OE, for the output registers are 1’b1 since its an active-low signal. When the data is taken and shifted, we need to send them to store registers with a clock signal, ST\_CP. For the next 24 cycles, we set SH\_CP to 1’b0 since we don’t need anymore data, ST\_CP to 1’b1 and OE to 1’b1 so that we delay the output until all the data is stored. When the store process is complete, we enable the output registers by setting OE signal to 1’b0. When we are done with a row, we go to the next row and repeat this process again. Finally, when all the rows are done, a frame appears on the 8x8 Display Module.

Timing



REFERENCES

1. **Lines 0-76 of** **DotMatrix.sv**; taken from Mert Aytöre’s CS223 Project; originally in Verilog, re-written in SystemVerilog and added some other functionality with respect to the needs of our project. Source: **github.com/mertaytore/cs223project**
2. **File** **SevSeg\_4digit.sv**; taken from CS223 Fall 2016 Lab Assignment 4; no changes made. Source: **dl.dropboxusercontent.com/u/11084576/CS223/SevSeg\_4digit.sv**

APPENDICE 1: SYSTEM VERILOG CODE OF THE PROJECT

**MAIN MODULE**

`timescale 1ns / 1ps

module Main(

input logic clk,

input logic mr,

input logic bi,

output logic oe, //output enable

output logic SH\_CP, // shift register clk pulse

output logic ST\_CP, // store register clk pulse

output logic reset, // reset for the shift register

output logic DS, // digital signal

output logic [7:0] row,

output logic [14:0] led,

output logic a, b, c, d, e, f, g, dp, //individual LED output for the 7-segment along with the digital point

output logic [3:0] an

);

logic [5:0] letter;

logic enable;

logic [1:0] click;

//Slowing down the clock in order to prevent delays

int count = 0;

logic clk\_en;

always@ (posedge clk)

begin

count <= count + 1;

if (count== 2) //D: last value for counter

count <= 27'b0; //N: length of counter

if (count==27'b0)

clk\_en <= 1'b1;

else

clk\_en <= 1'b0;

end

//inputConverterDriver ic(bi, clk, mr,enable, led, a, b, c, d, e, f, g, dp, an, click);

inputConverterV1 ic( bi, clk\_en, mr, enable, led, a, b, c, d, e, f, g, dp, an, click);

ControllerFSM cfsm( clk\_en, mr, click, enable, letter);

MatrixDriver md(clk\_en, letter, oe, SH\_CP, ST\_CP, reset, DS, row);

Endmodule

**INPUT CONVERTER**

`timescale 1ns / 1ps

/\*

This module converts button signals to the dash, dot, or space

according to the morse code.

\*/

module inputConverterV1(

input logic bi, clk, reset,

output logic enable,

output logic [14:0] led,

output logic a, b, c, d, e, f, g, dp, //individual LED output for the 7-segment along with the digital point

output logic [3:0] an,

output logic [1:0] signal

);

logic [3:0] in0, in1, in2, in3;

SevSeg\_4digit signalScreen(clk, in0, in1, in2, in3,a, b, c, d, e, f, g, dp,an );

int counter = 0, counter2 = 0;

typedef enum logic [2:0] { init, waitBi, space, waitSignal, preSignal, dot, dash } statetype;

statetype state, nextstate;

always@ (posedge clk)

begin

if( state == init) // Initializing the counters

begin

counter <= 0;

counter2 <= 0;

end

if( state == waitSignal || state == preSignal)// Dealing with counter 2

begin

counter <= 0;

counter2 <= counter2 + 1;

if (counter2== 100000000) //D: last value for counter

counter2 <= 27'b0; //N: length of counter

end

if( state == waitBi)

begin

counter2 <= 0;

counter <= counter + 1;

if (counter== 100000000) //D: last value for counter

counter <= 27'b0; //N: length of counter

end

if( state == space)

begin

counter <= 0;

end

end

//state register

always\_ff @(posedge clk, posedge reset)

if(reset) state <= init;

else state <= nextstate;

//Next State Logic

always\_comb

case( state)

init: nextstate = waitBi;

waitBi: begin

if(bi) nextstate = waitSignal;

else begin

if(counter > 27'b011100100111000011100000000 )

nextstate = space;

else

nextstate = waitBi;

end

end

space: nextstate = waitBi;

waitSignal: begin

if(bi && counter2 < 27'b001110010011100001110000000 )

nextstate = waitSignal;

if( bi && counter2 >= 27'b001110010011100001110000000)

nextstate = preSignal;

if( !bi)

nextstate = waitBi;

end

preSignal:begin

if( bi && counter >= 27'b100001011000001110110000000)

nextstate = dash;

if(bi && counter2 < 27'b101010111010100101010000000 )

nextstate = preSignal;

if( !bi)

nextstate = dot;

end

dot: nextstate = waitBi;

dash: nextstate = waitBi;

default: nextstate = init;

endcase

//Output Logic and Local Variables

always\_comb

case( state)

init: begin signal = 2'b00; enable = 0; in3 = 4'b0000; end

waitBi: begin signal = 2'b00; enable = 0; in3 = 4'b0001;end

space: begin signal = 2'b00; enable = 1; in3 = 4'b0010;end

waitSignal: begin signal= 2'b00; in3 = 4'b0011;end

preSignal: begin signal = 2'b00;in3 = 4'b0100; end

dot: begin signal = 2'b01; enable= 1; in3 = 4'b0101;end

dash: begin signal = 2'b10; enable= 1; in3 = 4'b0110; end

default: begin signal = 2'b00; enable = 0; in3 = 4'b1111; end

endcase

//Showing the counter on leds of Basys 3

always\_comb

begin

if( counter2 < 27'b000011001001011010101000000 )

led = 15'b100000000000000;

if( counter2 >= 27'b000011001001011010101000000 &&

counter2 < 27'b001010001100101110101000000)

led = 15'b110000000000000;

if( counter2 >= 27'b001010001100101110101000000 &&

counter2 < 27'b001001100010010110100000000)

led = 15'b111000000000000;

if( counter2 >= 27'b001001100010010110100000000 &&

counter2 < 27'b001100011001011101010000000)

led = 15'b111100000000000;

if( counter2 >= 27'b001111101111000101001000000 &&

counter2 < 27'b001100011001011101010000000)

led = 15'b111110000000000;

if( counter2 >= 27'b001100011001011101010000000 &&

counter2 < 27'b010011000100101101000000000)

led = 15'b111111000000000;

if( counter2 >= 27'b010011000100101101000000000 &&

counter2 < 27'b010101111011110011110000000)

led = 15'b111111100000000;

if( counter2 >= 27'b010101111011110011110000000 &&

counter2 < 27'b011001010001011011101000000)

led = 15'b111111110000000;

if( counter2 >= 27'b011001010001011011101000000 &&

counter2 < 27'b011100100111000011100000000)

led = 15'b111111111000000;

if( counter2 >= 27'b011100100111000011100000000 &&

counter2 < 27'b011111011110001010010000000)

led = 15'b111111111100000;

if( counter2 >= 27'b011111011110001010010000000 &&

counter2 < 27'b100010110011110010001000000)

led = 15'b111111111110000;

if( counter2 >= 27'b100010110011110010001000000 &&

counter2 < 27'b100110001001011010000000000)

led = 15'b111111111111000;

if( counter2 >= 27'b100110001001011010000000000 &&

counter2 < 27'b101001000000100000110000000)

led = 15'b111111111111100;

if( counter2 >= 27'b101001000000100000110000000 &&

counter2 < 27'b101100010110001000101000000)

led = 15'b111111111111110;

if( counter2 >= 27'b101100010110001000101000000 &&

counter2 < 27'b101111101011110000100000000)

led = 15'b111111111111111;

end

//Displaying the output signals on seven segment

always\_comb

begin

if( signal == 2'b00)

begin in0 = 4'b0000; in1 = 4'b0000; in2 = 4'b0000; end

if( signal == 2'b01)

begin in0 = 4'b0001; in1 = 4'b0000; in2 = 4'b0000;end

if( signal == 2'b10)

begin in0 = 4'b0000; in1 = 4'b0001; in2 = 4'b0000; end

end

endmodule

**CONTROLLER FSM**

`timescale 1ns / 1ps

module ControllerFSM(

input logic clk,

input logic reset,

input logic [1:0] click,

input logic enable,

output logic [5:0] letter

);

typedef enum logic [5:0] { D\_START, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S,T, U, V, W, X, Y, Z,U\_DASH, R\_DASH, O\_DOT, O\_DASH, INVALID, D0, D1, D2, D3, D4, D5, D6, D7, D8, D9 } statetype;

statetype state, nextstate;

//state register

always\_ff @(posedge clk, posedge reset)

begin

if(reset) state <= D\_START;

else begin if(enable) state <= nextstate; end

end

//next state logic

always\_comb

case(state)

D\_START:

begin

if( click == 2'b01 ) nextstate = E;

if( click == 2'b10 ) nextstate = T;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

E:

begin

if( click == 2'b01 ) nextstate = I;

if( click == 'b10 ) nextstate = A;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

T:

begin

if( click == 2'b01 ) nextstate = N;

if( click == 2'b10 ) nextstate = M;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

I:

begin

if( click == 2'b01 ) nextstate = S;

if( click == 2'b10 ) nextstate = U;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

A:

begin

if( click == 01 ) nextstate = R;

if( click == 10 ) nextstate = W;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

N:

begin

if( click == 01 ) nextstate = D;

if( click == 10 ) nextstate = K;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

M:

begin

if( click == 01 ) nextstate = G;

if( click == 10 ) nextstate = O;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

S:

begin

if( click == 01 ) nextstate = H;

if( click == 10 ) nextstate = V;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

U:

begin

if( click == 01 ) nextstate = F;

if( click == 10 ) nextstate = U\_DASH;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

R:

begin

if( click == 01 ) nextstate = L;

if( click == 10 ) nextstate = R\_DASH;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

W:

begin

if( click == 01 ) nextstate = P;

if( click == 10 ) nextstate = J;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

D:

begin

if( click == 01 ) nextstate = B;

if( click == 10 ) nextstate = X;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

K:

begin

if( click == 01 ) nextstate = C;

if( click == 10 ) nextstate = Y;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

G:

begin

if( click == 01 ) nextstate = Z;

if( click == 10 ) nextstate = Q;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

O:

begin

if( click == 01 ) nextstate = O\_DOT;

if( click == 10 ) nextstate = O\_DASH;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

H:

begin

if( click == 01 ) nextstate = D5;

if( click == 10 ) nextstate = D4;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

V:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = D3;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

F:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

U\_DASH:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = D2;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

L:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

R\_DASH:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

P:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

J:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = D1;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

B:

begin

if( click == 01 ) nextstate = D6;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

X:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

C:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

Y:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

Z:

begin

if( click == 01 ) nextstate = D7;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

Q:

begin

if( click == 01 ) nextstate = INVALID;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

O\_DOT:

begin

if( click == 01 ) nextstate = D8;

if( click == 10 ) nextstate = INVALID;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

O\_DASH:

begin

if( click == 01 ) nextstate = D9;

if( click == 10 ) nextstate = D0;

if( click != 2'b01 && click != 2'b10 ) nextstate = D\_START;

end

D5:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D4:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D3:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D2:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D1:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D6:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D7:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D8:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D9:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

D0:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

INVALID:

if( click == 00) nextstate = D\_START;

else nextstate = INVALID;

endcase

//output logic

always\_comb

if( click == 00 && enable)//If there is space

case(state)

D\_START: letter = 6'b000\_000;

A: letter = 6'b000\_001;

B: letter = 6'b000\_010;

C: letter = 6'b000\_011;

D: letter = 6'b000\_100;

E: letter = 6'b000\_101;

F: letter = 6'b000\_110;

G: letter = 6'b000\_111;

H: letter = 6'b001\_000;

I: letter = 6'b001\_001;

J: letter = 6'b001\_010;

K: letter = 6'b001\_011;

L: letter = 6'b001\_100;

M: letter = 6'b001\_101;

N: letter = 6'b001\_110;

O: letter = 6'b001\_111;

P: letter = 6'b010\_000;

Q: letter = 6'b010\_001;

R: letter = 6'b010\_010;

S: letter = 6'b010\_011;

T: letter = 6'b010\_100;

U: letter = 6'b010\_101;

V: letter = 6'b010\_110;

W: letter = 6'b010\_111;

X: letter = 6'b011\_000;

Y: letter = 6'b011\_001;

Z: letter = 6'b011\_010;

D0: letter = 6'b100\_000;

D1: letter = 6'b100\_001;

D2: letter = 6'b100\_010;

D3: letter = 6'b100\_011;

D4: letter = 6'b100\_100;

D5: letter = 6'b100\_100;

D6: letter = 6'b100\_101;

D7: letter = 6'b100\_110;

D8: letter = 6'b100\_111;

D9: letter = 6'b101\_000;

INVALID: letter = 6'b111\_111;

endcase

endmodule

**MATRIX DRIVER**

`timescale 1ns / 1ps

module MatrixDriver(

input logic clk,

input logic [5:0] letter,

output logic oe, //output enable

output logic SH\_CP, // shift register clk pulse

output logic ST\_CP, // store register clk pulse

output logic reset, // reset for the shift register

output logic DS, // digital signal

output logic [7:0] row);

// message ve row buradan matrix modulune verilecek

logic [7:0][23:0] data;

always\_ff@(posedge clk)

case(letter)

6'b000\_000: begin // D\_START

data[0] <= 24'b00000000\_00000000\_11111111;

data[1] <= 24'b00000000\_00000000\_11111111;

data[2] <= 24'b00000000\_00000000\_11111111;

data[3] <= 24'b00000000\_00000000\_11111111;

data[4] <= 24'b00000000\_00000000\_11111111;

data[5] <= 24'b00000000\_00000000\_11111111;

data[6] <= 24'b00000000\_00000000\_11111111;

data[7] <= 24'b00000000\_00000000\_11111111;

end

6'b000\_001: begin // A

data[0] <= 24'b00111100\_00000000\_00000000;

data[1] <= 24'b01111110\_00000000\_00000000;

data[2] <= 24'b11000011\_00000000\_00000000;

data[3] <= 24'b11000011\_00000000\_00000000;

data[4] <= 24'b11111111\_00000000\_00000000;

data[5] <= 24'b11111111\_00000000\_00000000;

data[6] <= 24'b11000011\_00000000\_00000000;

data[7] <= 24'b11000011\_00000000\_00000000;

end

6'b000\_010: begin // B

data[0] <= 24'b11111110\_00000000\_00000000;

data[1] <= 24'b11000001\_00000000\_00000000;

data[2] <= 24'b11000001\_00000000\_00000000;

data[3] <= 24'b11111110\_00000000\_00000000;

data[4] <= 24'b11111110\_00000000\_00000000;

data[5] <= 24'b11000001\_00000000\_00000000;

data[6] <= 24'b11000001\_00000000\_00000000;

data[7] <= 24'b11111110\_00000000\_00000000;

end

6'b000\_011: begin // C

data[0] <= 24'b01111100\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b11000011\_00000000\_00000000;

data[3] <= 24'b11000000\_00000000\_00000000;

data[4] <= 24'b11000000\_00000000\_00000000;

data[5] <= 24'b11000011\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b01111100\_00000000\_00000000;

end

6'b000\_100: begin // D

data[0] <= 24'b11111100\_00000000\_00000000;

data[1] <= 24'b11111110\_00000000\_00000000;

data[2] <= 24'b11000011\_00000000\_00000000;

data[3] <= 24'b11000011\_00000000\_00000000;

data[4] <= 24'b11000011\_00000000\_00000000;

data[5] <= 24'b11000011\_00000000\_00000000;

data[6] <= 24'b11111110\_00000000\_00000000;

data[7] <= 24'b01111100\_00000000\_00000000;

end

6'b000\_101: begin // E

data[0] <= 24'b11111111\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b11000000\_00000000\_00000000;

data[3] <= 24'b11111111\_00000000\_00000000;

data[4] <= 24'b11111111\_00000000\_00000000;

data[5] <= 24'b11000000\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b11111111\_00000000\_00000000;

end

6'b000\_110: begin // F

data[0] <= 24'b11111111\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b11000000\_00000000\_00000000;

data[3] <= 24'b11111000\_00000000\_00000000;

data[4] <= 24'b11111000\_00000000\_00000000;

data[5] <= 24'b11000000\_00000000\_00000000;

data[6] <= 24'b11000000\_00000000\_00000000;

data[7] <= 24'b11000000\_00000000\_00000000;

end

6'b000\_111: begin // G

data[0] <= 24'b11111111\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b11000000\_00000000\_00000000;

data[3] <= 24'b11011111\_00000000\_00000000;

data[4] <= 24'b11011111\_00000000\_00000000;

data[5] <= 24'b11000011\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b11111111\_00000000\_00000000;

end

6'b001\_000: begin // H

data[0] <= 24'b11000011\_00000000\_00000000;

data[1] <= 24'b11000011\_00000000\_00000000;

data[2] <= 24'b11000011\_00000000\_00000000;

data[3] <= 24'b11111111\_00000000\_00000000;

data[4] <= 24'b11111111\_00000000\_00000000;

data[5] <= 24'b11000011\_00000000\_00000000;

data[6] <= 24'b11000011\_00000000\_00000000;

data[7] <= 24'b11000011\_00000000\_00000000;

end

6'b001\_001: begin // I

data[0] <= 24'b01111110\_00000000\_00000000;

data[1] <= 24'b01111110\_00000000\_00000000;

data[2] <= 24'b00011000\_00000000\_00000000;

data[3] <= 24'b00011000\_00000000\_00000000;

data[4] <= 24'b00011000\_00000000\_00000000;

data[5] <= 24'b00011000\_00000000\_00000000;

data[6] <= 24'b01111110\_00000000\_00000000;

data[7] <= 24'b01111110\_00000000\_00000000;

end

6'b001\_010: begin // J

data[0] <= 24'b01111110\_00000000\_00000000;

data[1] <= 24'b01111110\_00000000\_00000000;

data[2] <= 24'b00011000\_00000000\_00000000;

data[3] <= 24'b00011000\_00000000\_00000000;

data[4] <= 24'b00011000\_00000000\_00000000;

data[5] <= 24'b00011000\_00000000\_00000000;

data[6] <= 24'b01111000\_00000000\_00000000;

data[7] <= 24'b01111000\_00000000\_00000000;

end

6'b001\_011: begin // K

data[0] <= 24'b0011\_0001\_0000000000000000;

data[1] <= 24'b0011\_0011\_0000000000000000;

data[2] <= 24'b0011\_0110\_0000000000000000;

data[3] <= 24'b0011\_1100\_0000000000000000;

data[4] <= 24'b0011\_1100\_0000000000000000;

data[5] <= 24'b0011\_0110\_0000000000000000;

data[6] <= 24'b0011\_0011\_0000000000000000;

data[7] <= 24'b0011\_0001\_0000000000000000;

end

6'b001\_100: begin // L

data[0] <= 24'b11000000\_00000000\_00000000;

data[1] <= 24'b11000000\_00000000\_00000000;

data[2] <= 24'b11000000\_00000000\_00000000;

data[3] <= 24'b11000000\_00000000\_00000000;

data[4] <= 24'b11000000\_00000000\_00000000;

data[5] <= 24'b11000000\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b11111111\_00000000\_00000000;

end

6'b001\_101: begin // M

data[0] <= 24'b11000011\_00000000\_00000000;

data[1] <= 24'b11100111\_00000000\_00000000;

data[2] <= 24'b11111111\_00000000\_00000000;

data[3] <= 24'b10011001\_00000000\_00000000;

data[4] <= 24'b10000001\_00000000\_00000000;

data[5] <= 24'b10000001\_00000000\_00000000;

data[6] <= 24'b10000001\_00000000\_00000000;

data[7] <= 24'b10000001\_00000000\_00000000;

end

6'b001\_110: begin // N

data[0] <= 24'b11000001\_00000000\_00000000;

data[1] <= 24'b11100001\_00000000\_00000000;

data[2] <= 24'b10110001\_00000000\_00000000;

data[3] <= 24'b10011001\_00000000\_00000000;

data[4] <= 24'b10001101\_00000000\_00000000;

data[5] <= 24'b10000111\_00000000\_00000000;

data[6] <= 24'b10000011\_00000000\_00000000;

data[7] <= 24'b10000001\_00000000\_00000000;

end

6'b001\_111: begin // O

data[0] <= 24'b00111100\_00000000\_00000000;

data[1] <= 24'b01111110\_00000000\_00000000;

data[2] <= 24'b11000011\_00000000\_00000000;

data[3] <= 24'b11000011\_00000000\_00000000;

data[4] <= 24'b11000011\_00000000\_00000000;

data[5] <= 24'b11000011\_00000000\_00000000;

data[6] <= 24'b01111110\_00000000\_00000000;

data[7] <= 24'b00111100\_00000000\_00000000;

end

6'b010\_000: begin // P

data[0] <= 24'b11111000\_00000000\_00000000;

data[1] <= 24'b11111100\_00000000\_00000000;

data[2] <= 24'b11000110\_00000000\_00000000;

data[3] <= 24'b11000110\_00000000\_00000000;

data[4] <= 24'b11111100\_00000000\_00000000;

data[5] <= 24'b11111000\_00000000\_00000000;

data[6] <= 24'b11000000\_00000000\_00000000;

data[7] <= 24'b11000000\_00000000\_00000000;

end

6'b010\_001: begin // Q

data[0] <= 24'b00111100\_00000000\_00000000;

data[1] <= 24'b01111110\_00000000\_00000000;

data[2] <= 24'b11000011\_00000000\_00000000;

data[3] <= 24'b11000011\_00000000\_00000000;

data[4] <= 24'b11000011\_00000000\_00000000;

data[5] <= 24'b11000011\_00000000\_00000000;

data[6] <= 24'b01111100\_00000000\_00000000;

data[7] <= 24'b00111111\_00000000\_00000000;

end

6'b010\_010: begin // R

data[0] <= 24'b11111000\_00000000\_00000000;

data[1] <= 24'b10001100\_00000000\_00000000;

data[2] <= 24'b10001100\_00000000\_00000000;

data[3] <= 24'b10001100\_00000000\_00000000;

data[4] <= 24'b11111000\_00000000\_00000000;

data[5] <= 24'b10001000\_00000000\_00000000;

data[6] <= 24'b10000100\_00000000\_00000000;

data[7] <= 24'b10000010\_00000000\_00000000;

end

6'b010\_011: begin // S

data[0] <= 24'b01111110\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b11000000\_00000000\_00000000;

data[3] <= 24'b11111110\_00000000\_00000000;

data[4] <= 24'b01111111\_00000000\_00000000;

data[5] <= 24'b00000011\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b01111110\_00000000\_00000000;

end

6'b010\_100: begin // T

data[0] <= 24'b11111111\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b00011000\_00000000\_00000000;

data[3] <= 24'b00011000\_00000000\_00000000;

data[4] <= 24'b00011000\_00000000\_00000000;

data[5] <= 24'b00011000\_00000000\_00000000;

data[6] <= 24'b00011000\_00000000\_00000000;

data[7] <= 24'b00011000\_00000000\_00000000;

end

6'b010\_101: begin // U

data[0] <= 24'b11000011\_00000000\_00000000;

data[1] <= 24'b11000011\_00000000\_00000000;

data[2] <= 24'b11000011\_00000000\_00000000;

data[3] <= 24'b11000011\_00000000\_00000000;

data[4] <= 24'b11000011\_00000000\_00000000;

data[5] <= 24'b11000011\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b11111111\_00000000\_00000000;

end

6'b010\_110: begin // V

data[0] <= 24'b11000011\_00000000\_00000000;

data[1] <= 24'b11000011\_00000000\_00000000;

data[2] <= 24'b11000011\_00000000\_00000000;

data[3] <= 24'b11000011\_00000000\_00000000;

data[4] <= 24'b11000011\_00000000\_00000000;

data[5] <= 24'b01111110\_00000000\_00000000;

data[6] <= 24'b00111100\_00000000\_00000000;

data[7] <= 24'b00011000\_00000000\_00000000;

end

6'b010\_111: begin // W

data[0] <= 24'b11000011\_00000000\_00000000;

data[1] <= 24'b11000011\_00000000\_00000000;

data[2] <= 24'b11011011\_00000000\_00000000;

data[3] <= 24'b11011011\_00000000\_00000000;

data[4] <= 24'b11111111\_00000000\_00000000;

data[5] <= 24'b11100111\_00000000\_00000000;

data[6] <= 24'b11000011\_00000000\_00000000;

data[7] <= 24'b11000011\_00000000\_00000000;

end

6'b011\_000: begin // X

data[0] <= 24'b11000011\_00000000\_00000000;

data[1] <= 24'b11100111\_00000000\_00000000;

data[2] <= 24'b01111110\_00000000\_00000000;

data[3] <= 24'b00111100\_00000000\_00000000;

data[4] <= 24'b00111100\_00000000\_00000000;

data[5] <= 24'b01111110\_00000000\_00000000;

data[6] <= 24'b11100111\_00000000\_00000000;

data[7] <= 24'b11000011\_00000000\_00000000;

end

6'b011\_001: begin // Y

data[0] <= 24'b11100111\_00000000\_00000000;

data[1] <= 24'b11100111\_00000000\_00000000;

data[2] <= 24'b11100111\_00000000\_00000000;

data[3] <= 24'b11100111\_00000000\_00000000;

data[4] <= 24'b00111100\_00000000\_00000000;

data[5] <= 24'b00111100\_00000000\_00000000;

data[6] <= 24'b00111100\_00000000\_00000000;

data[7] <= 24'b00111100\_00000000\_00000000;

end

6'b011\_010: begin // Z

data[0] <= 24'b11111111\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b00001110\_00000000\_00000000;

data[3] <= 24'b00011100\_00000000\_00000000;

data[4] <= 24'b00111000\_00000000\_00000000;

data[5] <= 24'b01110000\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b11111111\_00000000\_00000000;

end

6'b100\_000: begin // D0

data[0] <= 24'b00111100\_00000000\_00000000;

data[1] <= 24'b01111110\_00000000\_00000000;

data[2] <= 24'b01100110\_00000000\_00000000;

data[3] <= 24'b01100110\_00000000\_00000000;

data[4] <= 24'b01100110\_00000000\_00000000;

data[5] <= 24'b01100110\_00000000\_00000000;

data[6] <= 24'b01111110\_00000000\_00000000;

data[7] <= 24'b00111100\_00000000\_00000000;

end

6'b100\_001: begin // D1

data[0] <= 24'b00111000\_00000000\_00000000;

data[1] <= 24'b01111000\_00000000\_00000000;

data[2] <= 24'b01011000\_00000000\_00000000;

data[3] <= 24'b00011000\_00000000\_00000000;

data[4] <= 24'b00011000\_00000000\_00000000;

data[5] <= 24'b00011000\_00000000\_00000000;

data[6] <= 24'b01111110\_00000000\_00000000;

data[7] <= 24'b01111110\_00000000\_00000000;

end

6'b100\_010: begin // D2

data[0] <= 24'b00111100\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b11000111\_00000000\_00000000;

data[3] <= 24'b00001110\_00000000\_00000000;

data[4] <= 24'b00011100\_00000000\_00000000;

data[5] <= 24'b00111000\_00000000\_00000000;

data[6] <= 24'b01110000\_00000000\_00000000;

data[7] <= 24'b11111111\_00000000\_00000000;

end

6'b100\_011: begin // D3

data[0] <= 24'b01111110\_00000000\_00000000;

data[1] <= 24'b01111110\_00000000\_00000000;

data[2] <= 24'b00000110\_00000000\_00000000;

data[3] <= 24'b00011110\_00000000\_00000000;

data[4] <= 24'b00011110\_00000000\_00000000;

data[5] <= 24'b00000110\_00000000\_00000000;

data[6] <= 24'b01111110\_00000000\_00000000;

data[7] <= 24'b01111110\_00000000\_00000000;

end

6'b100\_100: begin // D4

data[0] <= 24'b11000000\_00000000\_00000000;

data[1] <= 24'b11000000\_00000000\_00000000;

data[2] <= 24'b11000000\_00000000\_00000000;

data[3] <= 24'b11000011\_00000000\_00000000;

data[4] <= 24'b11111111\_00000000\_00000000;

data[5] <= 24'b11111111\_00000000\_00000000;

data[6] <= 24'b00000011\_00000000\_00000000;

data[7] <= 24'b00000011\_00000000\_00000000;

end

6'b100\_101: begin // D5

data[0] <= 24'b11111111\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b11000000\_00000000\_00000000;

data[3] <= 24'b11111111\_00000000\_00000000;

data[4] <= 24'b11111111\_00000000\_00000000;

data[5] <= 24'b00000011\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b11111111\_00000000\_00000000;

end

6'b100\_110: begin // D6

data[0] <= 24'b01111000\_00000000\_00000000;

data[1] <= 24'b01111000\_00000000\_00000000;

data[2] <= 24'b01100000\_00000000\_00000000;

data[3] <= 24'b01100000\_00000000\_00000000;

data[4] <= 24'b01111110\_00000000\_00000000;

data[5] <= 24'b01100110\_00000000\_00000000;

data[6] <= 24'b01100110\_00000000\_00000000;

data[7] <= 24'b01111110\_00000000\_00000000;

end

6'b100\_111: begin // D7

data[0] <= 24'b11111111\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b00000110\_00000000\_00000000;

data[3] <= 24'b00001100\_00000000\_00000000;

data[4] <= 24'b00011000\_00000000\_00000000;

data[5] <= 24'b00110000\_00000000\_00000000;

data[6] <= 24'b01100000\_00000000\_00000000;

data[7] <= 24'b11000000\_00000000\_00000000;

end

6'b101\_000: begin // D8

data[0] <= 24'b01111110\_00000000\_00000000;

data[1] <= 24'b01000010\_00000000\_00000000;

data[2] <= 24'b01111110\_00000000\_00000000;

data[3] <= 24'b00111100\_00000000\_00000000;

data[4] <= 24'b00111100\_00000000\_00000000;

data[5] <= 24'b01111110\_00000000\_00000000;

data[6] <= 24'b01000010\_00000000\_00000000;

data[7] <= 24'b01111110\_00000000\_00000000;

end

6'b101\_001: begin // D9

data[0] <= 24'b01111110\_00000000\_00000000;

data[1] <= 24'b01000010\_00000000\_00000000;

data[2] <= 24'b01000010\_00000000\_00000000;

data[3] <= 24'b01000010\_00000000\_00000000;

data[4] <= 24'b01111110\_00000000\_00000000;

data[5] <= 24'b00000010\_00000000\_00000000;

data[6] <= 24'b00000010\_00000000\_00000000;

data[7] <= 24'b01111110\_00000000\_00000000;

end

6'b111\_111: begin // INVALID

data[0] <= 24'b00000000\_11111111\_00000000;

data[1] <= 24'b00000000\_11111111\_00000000;

data[2] <= 24'b00000000\_11111111\_00000000;

data[3] <= 24'b00000000\_11111111\_00000000;

data[4] <= 24'b00000000\_11111111\_00000000;

data[5] <= 24'b00000000\_11111111\_00000000;

data[6] <= 24'b00000000\_11111111\_00000000;

data[7] <= 24'b00000000\_11111111\_00000000;

end

default: begin // Default

data[0] <= 24'b11111111\_00000000\_00000000;

data[1] <= 24'b11111111\_00000000\_00000000;

data[2] <= 24'b11111111\_00000000\_00000000;

data[3] <= 24'b11111111\_00000000\_00000000;

data[4] <= 24'b11111111\_00000000\_00000000;

data[5] <= 24'b11111111\_00000000\_00000000;

data[6] <= 24'b11111111\_00000000\_00000000;

data[7] <= 24'b11111111\_00000000\_00000000;

end

endcase

DotMatrix dm(clk, data, oe, SH\_CP, ST\_CP, reset, DS, row);

Endmodule

**DOT MATRIX**

`timescale 1ns / 1ps

module DotMatrix( input logic clk,

input logic [7:0][23:0] in\_data,

output logic oe, //output enable

output logic SH\_CP, // shift register clk pulse

output logic ST\_CP, // store register clk pulse

output logic reset, // reset for the shift register

output logic DS, // digital signal

output logic [7:0] row);

logic f,e;

logic [1:0] counter;

logic [8:0] i = 1; // data sinyalinin ileticisi

logic [2:0] a = 0;

logic [9:0] d = 0;

logic [7:0][23:0] data = in\_data;

logic [24:1] message;

always@(posedge clk)

begin

counter <= counter + 1;

f <= counter[1]; // SHCP

e <= ~f;

end

always@(posedge e)

i = i + 9'b000000001;

always@(\*)

begin

if (i < 9'b000000100) // baslangıçta i 4'e gelene kadar sisteme reset atıyor.

reset<=0;

else

reset<=1;

if (i>9'b000000011 && i<9'b000011100) //4'le 27 arasında data akışı seri olarak devam ediyor

DS<=message[i-9'b000000011];

else

DS<=0;

if (i<9'b000011100) //i 28'e geldiğinde data akış tamamlanıyor. 24 bit data alınmı oluyor. bu sureden sonra clk durduluyor yeni data akışına kadar.

begin

SH\_CP<=f;

ST\_CP<=e;

end

else

begin

SH\_CP<=0;

ST\_CP<=1;

end

end

always @(posedge f)//clk un durduğu surede OE=0 yani output registerin çıktığında aktif durumda.

begin

if (i>9'b000011100 && i<9'b110011101)

oe<=0;

else

oe<=1;

end

always @(posedge f) //bir satır tamamlandığında a bir arttırılıyor 2. satıra geçmek için.

begin

if (i==9'b110011110)

begin

a = a+1;

//i <=9'b0;

end

end

always\_ff@(posedge clk)

message <= data[a];

always\_comb

case(a)

3'b000: row = 8'b0000\_0001;

3'b001: row = 8'b0000\_0010;

3'b010: row = 8'b0000\_0100;

3'b011: row = 8'b0000\_1000;

3'b100: row = 8'b0001\_0000;

3'b101: row = 8'b0010\_0000;

3'b110: row = 8'b0100\_0000;

3'b111: row = 8'b1000\_0000;

default: row = 8'b0000\_0000;

endcase

endmodule

**SEVSEG\_4DIGIT**

`timescale 1ns / 1ps

module SevSeg\_4digit(

input clk,

input [3:0] in0, in1, in2, in3, // 4 values for 4 digits (decimal value)

output a, b, c, d, e, f, g, dp, //individual LED output for the 7-segment along with the digital point

output [3:0] an // anode: 4-bit enable signal (active low)

);

// divide system clock (100Mhz for Basys3) by 2^N using a counter, which allows us to multiplex at lower speed

localparam N = 18;

logic [N-1:0] count = {N{1'b0}}; //initial value

always@ (posedge clk)

count <= count + 1;

logic [3:0]digit\_val; // 7-bit register to hold the current data on output

logic [3:0]digit\_en; //register for enable vector

always\_comb

begin

digit\_en = 4'b1111; //default

digit\_val = in0; //default

case(count[N-1:N-2]) //using only the 2 MSB's of the counter

2'b00 : //select first 7Seg.

begin

digit\_val = in0;

digit\_en = 4'b1110;

end

2'b01: //select second 7Seg.

begin

digit\_val = in1;

digit\_en = 4'b1101;

end

2'b10: //select third 7Seg.

begin

digit\_val = in2;

digit\_en = 4'b1011;

end

2'b11: //select forth 7Seg.

begin

digit\_val = in3;

digit\_en = 4'b0111;

end

endcase

end

//Convert digit number to LED vector. LEDs are active low.

logic [6:0] sseg\_LEDs;

always\_comb

begin

sseg\_LEDs = 7'b1111111; //default

case(digit\_val)

4'd0 : sseg\_LEDs = 7'b1000000; //to display 0

4'd1 : sseg\_LEDs = 7'b1111001; //to display 1

4'd2 : sseg\_LEDs = 7'b0100100; //to display 2

4'd3 : sseg\_LEDs = 7'b0110000; //to display 3

4'd4 : sseg\_LEDs = 7'b0011001; //to display 4

4'd5 : sseg\_LEDs = 7'b0010010; //to display 5

4'd6 : sseg\_LEDs = 7'b0000010; //to display 6

4'd7 : sseg\_LEDs = 7'b1111000; //to display 7

4'd8 : sseg\_LEDs = 7'b0000000; //to display 8

4'd9 : sseg\_LEDs = 7'b0010000; //to display 9

default : sseg\_LEDs = 7'b0111111; //dash

endcase

end

assign an = digit\_en;

assign {g, f, e, d, c, b, a} = sseg\_LEDs;

assign dp = 1'b1; //turn dp off

endmodule