Course Name: Digital System Design Lab

Course Number and Section: **14:332:437:03**

Experiment: 2 – Seven-Segment Display

Lab Instructor: Sumit Maheshwari and Chen Wang

Date Performed: 11/8/2017

Date Submitted: 11/8/2017

Submitted by: Pawel Derkacz - 151 00 5994

----------For Lab Instructor Use ONLY----------

GRADE: _____

COMMENTS:

Electrical and Computer Engineering Department
School of Engineering
Rutgers University, Piscataway, NJ 08854

PURPOSE

To create a circuit that will act as a Moore Finite-State Machine. In this case, it is to create a seven-segment display. Note: registers are not included in design, but in reality there would be a D flip-flop right after all output. Note: Total time spent on this lab was approximately 12 hours (due to checking all possibilities in regards to a combinational circuit).

Circuit Equations and Diagram:

Note: Diagram attached separately, to preserve all data in photo (not shrinking, cropping, or otherwise modifying the original). Also, I am using the Quartus circuit diagram also as a computer drawn schematic. Lastly, if you are having difficulty seeing the gate output, each output should directly follow my equations (thus making things slightly easier to decipher).

Equations and Explanations:

For low-output (used in Quartus):

$$\overline{S_a} = D_2\overline{D_1 D_0} + D_3 D_2 \overline{D_1} + \overline{D_3 D_2 D_1} D_0 + D_3 \overline{D_2} D_1 D_0$$
$$\overline{S_b} = \overline{D_3} D_2 \overline{D_1} D_0 + D_3 D_2 \overline{D_0} + D_3 D_1 D_0 + D_2 D_1 \overline{D_0}$$
$$\overline{S_c} = \overline{D_3 D_2} D_1 \overline{D_0} + D_3 D_2 D_1 + D_3 D_2 \overline{D_0}$$
$$\overline{S_d} = \overline{D_3} D_2 \overline{D_1 D_0} + D_3 \overline{D_2} D_1 \overline{D_0} + D_2 D_1 D_0 + \overline{D_2 D_1} D_0$$
$$\overline{S_e} = \overline{D_3} D_2 \overline{D_1} + \overline{D_2 D_1} D_0 + \overline{D_3} D_0$$
$$\overline{S_f} = D_3 D_2 \overline{D_1} + \overline{D_3 D_2} D_0 + \overline{D_3} D_1 D_0 + \overline{D_3 D_2} D_1$$
$$\overline{S_g} = \overline{D_3 D_2 D_1} + \overline{D_3} D_2 D_1 D_0$$

Low-output would use 21 ANDs, 7 ORs, and 4 INVs. Could use 25 Two-Input ANDs instead of a mix of 21 Three- and Four-Input ANDs. Longest chain for the first circuit would be INV → AND → OR → INV (if needing to invert the output to get a '1'). As for the second, it would be INV → AND → AND → OR → INV. Thus, the second circuit is worse in all ways, when compared to the first (other than using only 2-input ANDs).

For high-output:

$$S_a = \overline{D_2 D_0} + \overline{D_3} D_1 + D_2 D_1 + \overline{D_3} D_2 D_0 + D_3 \overline{D_2 D_1}$$
$$S_b = \overline{D_2 D_1} + \overline{D_2 D_0} + \overline{D_3 D_1 D_0} + \overline{D_3} D_1 D_0 + D_3 \overline{D_1} D_0$$
$$S_c = \overline{D_2 D_1} + \overline{D_3} D_0 + \overline{D_1} D_0 + D_3 \overline{D_2} + \overline{D_3} D_2$$
$$S_d = D_2 \overline{D_1} D_0 + \overline{D_2} D_1 D_0 + D_3 \overline{D_1 D_0} + \overline{D_3 D_2 D_0} + D_2 D_1 \overline{D_0}$$
$$S_e = \overline{D_2 D_0} + D_1 \overline{D_0} + D_3 D_2 + D_3 D_1$$
$$S_f = D_3 D_1 + D_2 D_1 \overline{D_0} + D_3 \overline{D_2} + \overline{D_3} D_2 \overline{D_1} + \overline{D_3 D_1 D_0}$$
$$S_g = D_3 + D_2 \overline{D_1} + D_1 \overline{D_0} + \overline{D_2} D_1$$

High-output would require 24 ANDs, 7 ORs, and 4 INVs. If using only 2-Input ANDs, only one more would be needed, resulting in 25 total ANDs. However, this requires a sequence of INV → AND → AND → OR, resulting in a slightly longer delay time. If using only 2-input ORs, then the low-input would require 17, and high-output would require 27. The difference in time would only be 1 * T_OR for low-output, and 2 * T_OR for high-output. Therefore, the total component number for the fastest circuit

(low-output, with varying input lines) is 32, while the component number for the 2-input only circuit is 56 (high-output). In terms of cost, the faster circuit comes out ahead by $0.50 (if considering protoboard chips), but has many more unused slots on the chips. Since the circuit will be built on an FPGA, I choose to use the fastest low-output circuit, with an inverter on the output to make said output a logic high. Note, since Quartus doesn't like to use lpm_inv, I used NANDs instead.

THEORY

For a **low-output** circuit, the following should be the output table:

Hex	Inputs				Outputs							Out put
Digit	D_3	D_2	D_1	D_0	S_g	S_f	S_e	S_d	S_c	S_b	S_a	Hex
0	0	0	0	0	1	0	0	0	0	0	0	40
1	0	0	0	1	1	1	1	1	0	0	1	79
2	0	0	1	0	0	1	0	0	1	0	0	24
3	0	0	1	1	0	1	1	0	0	0	0	30
4	0	1	0	0	0	0	1	1	0	0	1	19
5	0	1	0	1	0	0	1	0	0	1	0	12
6	0	1	1	0	0	0	0	0	0	1	0	02
7	0	1	1	1	1	1	1	1	0	0	0	78
8	1	0	0	0	0	0	0	0	0	0	0	00
9	1	0	0	1	0	0	1	1	0	0	0	18
A	1	0	1	0	0	0	0	1	0	0	0	08
B	1	0	1	1	0	0	0	0	0	1	1	03
C	1	1	0	0	0	1	0	0	1	1	1	27
D	1	1	0	1	0	1	0	0	0	0	1	21
E	1	1	1	0	0	0	0	0	1	1	0	06
F	1	1	1	1	0	0	0	1	1	1	0	0E

DATA SECTION

Waveforms from ModelSim are attached separately, for the same reason as the circuit schematic. There are two pictures, one that has a binary radix, and the other is in hex.

CONCLUSIONS

This lab gave me some major headaches, trying to find out which gates could be reused, and if it would be better to have high redundancy vs. speed vs. cost. All in all, I believe there is a simpler way to make this circuit, without normal combinational logic. I am certain MUXs could be used in some way, and simplify the circuit dramatically. However, at this point I have done the combinational circuit, and no longer wish to experience this headache. Perhaps some other day I will find a more elegant solution to this lab.