EGCP 1010
Digital Logic Design (DLD)
Laboratory #3

Introduction to CedarLogic

Prepared By:
Alex Laird

on

September 24, 2007

Lab Partner:
Ryan Morehart

Objective:

The goal of this laboratory was to improve familiarity with CedarLogic, waveforms, and oscilloscopes. The student should gain a better understanding of Digital Logic Simulators and their abilities, and should successfully design a circuit to implement $F(A,B,C,D) = \sum m(0,1,4,8,12,13) + \sum d(14,15)$ using NAND-NANDX and NOR-NORX gates.

Procedure and Results:

For this lab, the class first implemented the following solution into a K-Map:

$F(A,B,C,D) = \sum m(0,1,4,8,12,13) + \sum d(14,15)$

The K-Map for this solution is shown in Table 1.

Table 1: K-Map for F

1	1	1	1
1	0	1	0
0	0	X	0
0	0	X	0

From these K-Maps, you can deduce the following Sum-Of-Product (SOP) solution by grouping the 1's.

F = A'B'C' + C'D' + AB <== (F SOP)

To get F', we grouped the 0's and received the following SOP solution.

F' = A'BD + AB'D + C <== (F' SOP)

To get the Product-Of-Sum (POS) of F, we complemented F' using DeMorgan's laws.

F = (A + B' + D')(A' + B + D')C' <== (F POS)

Before implementing F SOP and F POS, we created an Input Generation Circuit to generate inputs, counting in ascending numerical order from 0_{16} (0000_2) to F_{16} (1111_2). The CedarLogic circuit is shown in Figure 1. The Input Generation Circuit receives a four digit binary number in as an input and outputs those four digits. Each position is split to have two separate outputs; a normal output, and its inverse.

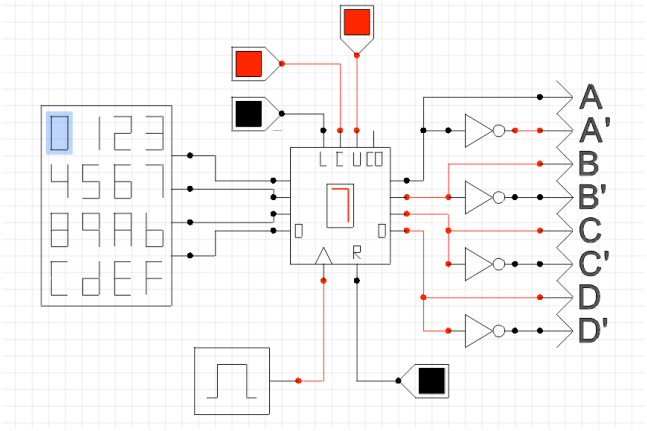

Figure 1: Input Generation Circuit

We implemented F SOP and F POS using the generated outputs from the Input Generation Circuit. The implementations of those are shown in Figure 2. F in SOP is the AND-OR (AO) and NAND-NANDX (NN) circuits. F in POS form is shown by the OR-AND (OA) and NOR-NORX (RR) circuits. However, all four circuits give the same solution everywhere except E_{16} and F_{16}. Notice that C has been inverted in the NOR-NORX circuit. This is because the inversions must cancel out, and since it is inverted on the front end of the NORX gate, the C must be inverted to make the circuit work.

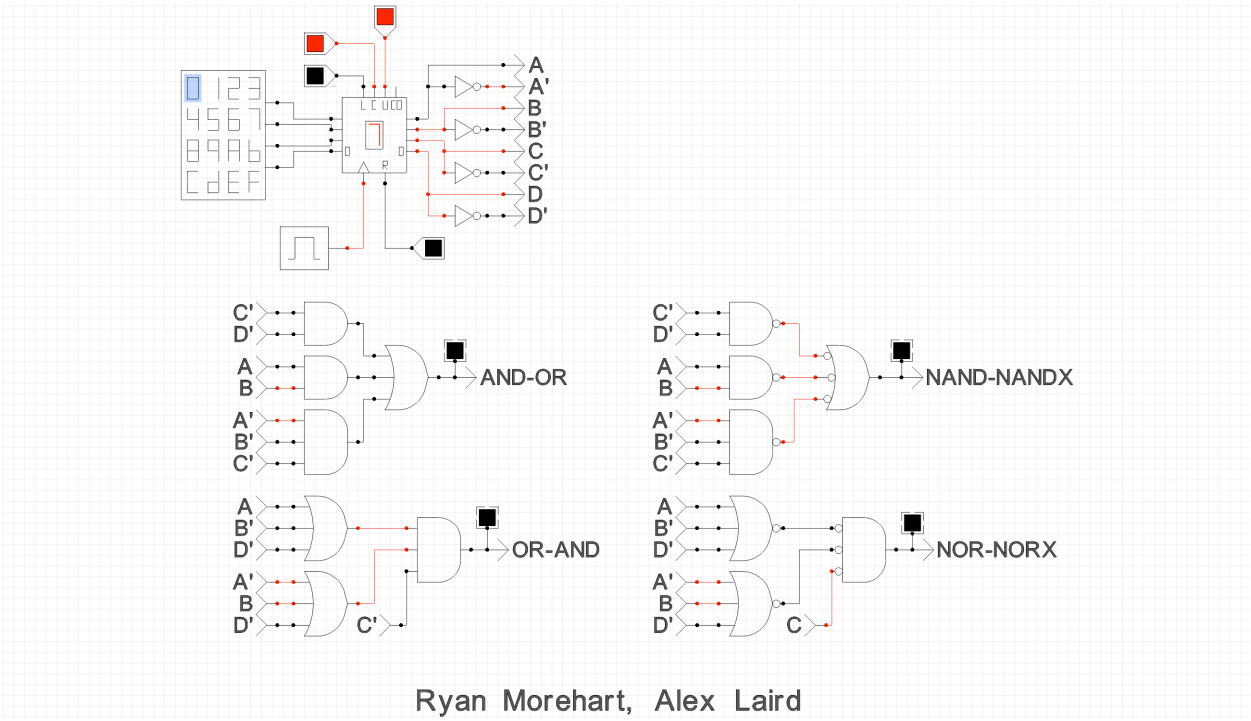

Figure 2: Full Circuit in CedarLogic

After verifying that the circuits performed correctly for every number from 0_{16} (0000_2) to F_{16} (1111_2), with the exception of E_{16} and F_{16}, we opened the Oscilloscope viewer in CedarLogic. We created the oscilloscope shown in Figure 3.

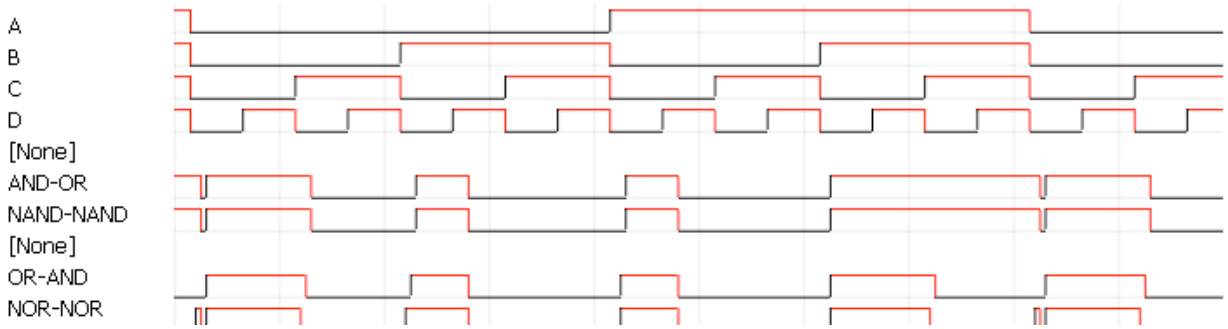

Figure 3: Oscilloscope for F in CedarLogic's Oscilloscope Viewer

From the Oscilloscope shown in Figure 3, the Truth Table shown in Table 2 was deduced.

Table 2: Truth Table for F

A	B	C	D	AO	OR	NN	RR
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	1	1	1	1
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0
1	1	0	0	1	1	1	1
1	1	0	1	1	1	1	1

A	B	C	D	AO	OR	NN	RR
1	1	1	0	1	0	1	0
1	1	1	1	1	0	1	0

As shown by Table 2, 1110_2 and 1111_2 vary in outputs between AO to OR and NN to RR. The reason for the difference in these last two numbers is because of the don't-care bits in the original F solution. This Truth Table and circuit is *still correct*. Since we don't care about those bits, we just ignore the fact that they differ.

Conclusion and Suggestions:

This lab gave me a good overview of CedarLogic and it's available options. The only real problems I ran into were with a few slight glitches in CedarLogic. Other then that, we didn't have any problems with this lab. Overall it was very insightful.

Questions:

I. Which Transistor-Transistor Logic (TTL) chips that are available in lab would be needed to implement each of the four implementations using the minimum number of chips per implementation (AO, OA, NN, RR). Explain which design you would implement in hardware if you were asked to and why.

One (1) 74LS11 (3-input AND) and one (1) 74LS32 (2-input OR) chips for the AND-OR circuit.
One (1) 74LS32 (2-input OR) and one (1) 74LS11 (3-input AND) chips for the OR-AND circuit.
Two (2) 74LS10 (3-input NAND) chips for the NAND-NANDX circuit.
One (1) 74LS27 (3-input NOR) chip for the NOR-NORX circuit.

I would implement the RR circuit because it has the lowest cost. Though it appears to have the same cost as the OA circuit, RR gates are faster, so this circuit will perform the best of the four.

II. If all four implementations did not give the same result for every possible input combination, and you believe your implementations are correct, explain why.

Because of the don't-care bits. On the K-Map, slots 1110_2 and 1111_2 both contain X's, which indicate that they can be either a 1 or 0, whichever is needed. Since they were used in a grouping for *both* the 1 grouping and the 0 grouping (F and F'), they show opposite answers when E_{16} or F_{16} are inputed to the circuits.

III. Since the CedarLogic program was student created and still under revision, we want your input to improve and fix any bugs. Write out at least two improvements you would like to see addressed.

The program almost always crashes when you try to delete a component that already has many lines connected to another component. If you detach the lines first, then delete it, *sometimes* it will allow you to delete it. Not always, however. This also tends to happen more regularly the longer you have been using the program, but, again, not always.

When creating the four circuits, we attempted to diagnose a problem for over ten minutes because, strangely, the NN circuit was different from the other three circuits a lot of the time. And obviously we were trying to get all four circuits to continuously have the same output (except at E_{16} and F_{16}). We couldn't find anything wrong with the circuit. We deleted the NAND gate that was attached to the inputs C' and D' and replaced this gate with *the exact same gate*. Ironically, the circuit worked from this point on. There was absolutely no logical reason for this to "fix" the problem. We deleted the gate and replaced it with an identical gate. The gate was a NAND gate, so perhaps there is a problem with something in the NAND programming.

It would be really nice if it were a little easier to connect the lines between two components. One way that would be significantly easier would be if you could select multiple components, let's say three for example, and then hold down some key, like Ctrl or Shift, and then click on a three input gate of some kind and the program would automatically connect all three of those lines to the gate and the three components outputs. Also, instead of having to drop a line directly to the input or output of some component, it would be nice if you could just drop a line "anywhere" on a component and it would automatically connect the line to the highest available input (or output if the line was coming from an input).