EGCP 1010
Digital Logic Design (DLD)
Laboratory #6

Four by Four (4 x 4) Sorting Stack

Prepared By:
Alex Laird

on

October 31st, 2007

Lab Partner:
Ryan Morehart

Objective:

The goal of this laboratory is to expose the student to a large-scale design problem, starting with a word description, causing the student to create Truth Tables, K-Maps, and finally Gate-Level Schematics. The student should learn that often times in circuit design, a single cell can be constructed and then simply copied and pasted for problems such as the Four by Four (4 x 4) Sorting Stack.

Abstract:

In this lab report, I will describe the procedure which was taken to come to the result of the Four by Four (4 x 4) Sorting Stack. First, we considered the behavior of the Stack. We also considered the implementation for the Interface. From the behavior, we deduced Truth Tables for both the Compare Logic and the Decision Logic. Karnaugh (K) Maps were created from the Truth Tables, and Gate-Level Schematics from the K-Maps. Since we realized the design for the Sorting Stack was repetitive for each bit, we realized we could simply make a one (1) bit cell and copy that multiple times. Using this method of copying, we created an entire Four by Four (4 x 4) Sorting Stack with an easy-to-use Input/ Output (I/O) Interface.

Procedure and Results:

Design Description:

To start, we created used the following design the I/O of our stack.

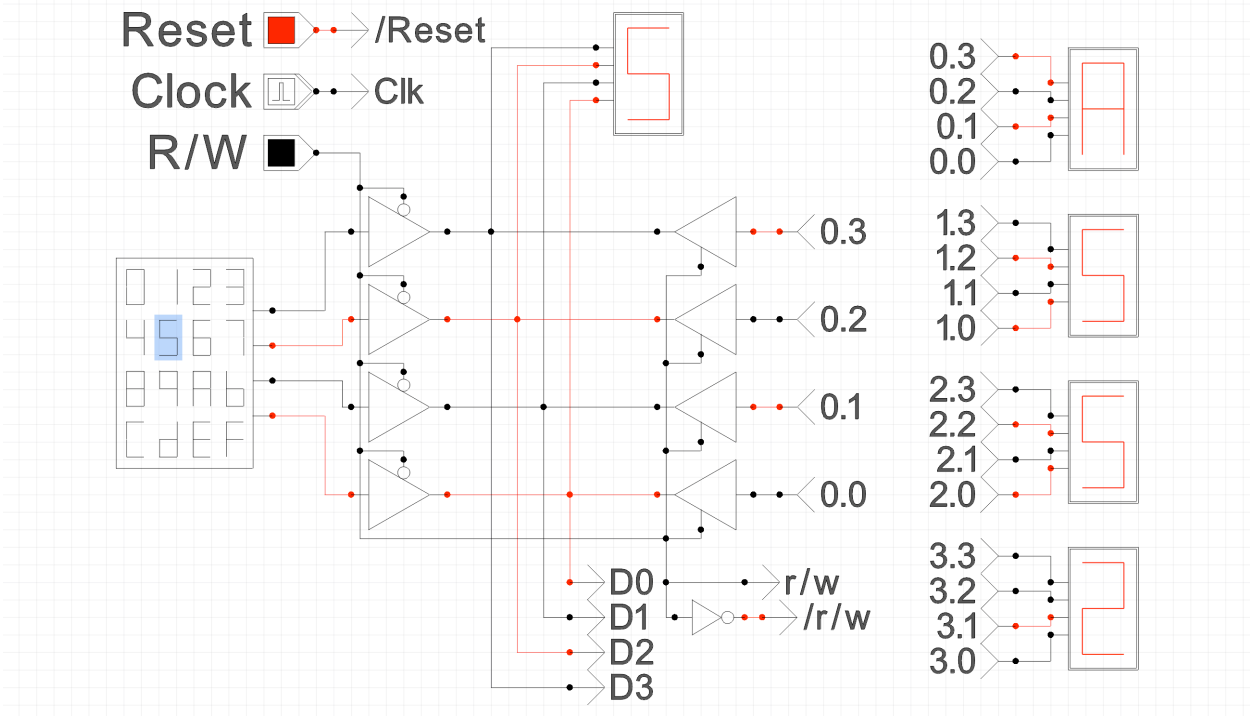

Figure 1: I/O Interface

The circuit would implement a Low-Active Reset, so assuming the Reset bit was set high (1), the user could select a number from the keypad, initiate a clock event, and the number would be added to the stack in a sorted manner. The number being added would be compared to *each individual* number that was already on the stack using the Compare Logic, which we would design, and the Decision Logic, also to be designed, would choose what position on the stack the new number would be placed at.

Figure 2 shows an example of the behavior of the stack with given inputs from the keypad, specified in the “Data” row. The “r/w” indicates whether the current state is Read (1) or Write (0). All events happen on the rising edge of the clock cycle, as indicated by the word “Rising” on the “Clk” row. Note that the data shown in “Data” row is the *current* input, and is stored on the *next* clock cycle.

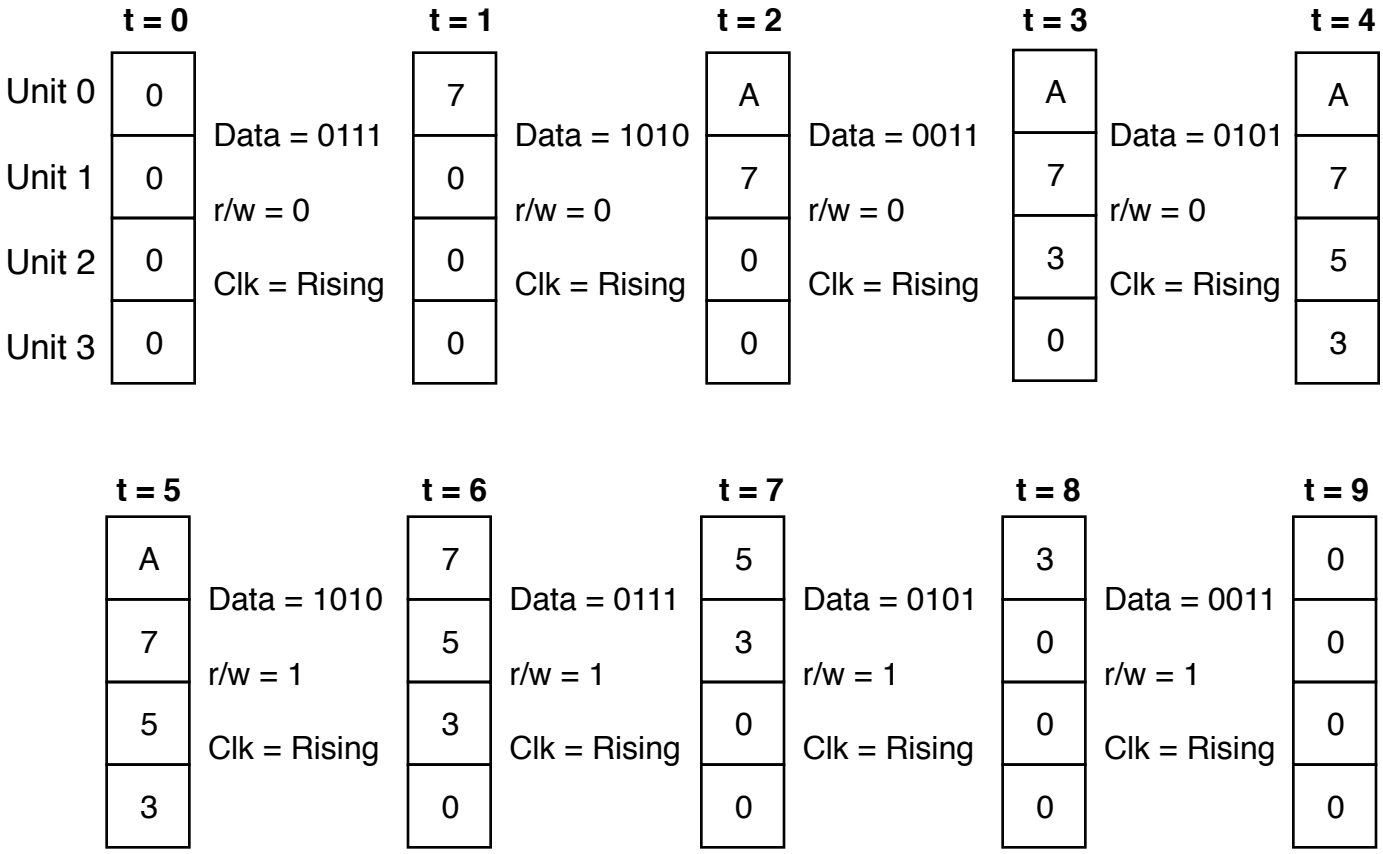

Figure 2: Functional Description of the Four by Four (4 x 4) Sorting Stack

As shown in the Figure above, if r/w = 0, when a data input is selected on the keypad, it is set in the queue. When the clock event occurs, the four (4) bit data is compared to each of the current values in Unit 0, Unit 1, Unit 2, and Unit 3. Each unit shown above is one of the 4-Bit Units, as shown in Figure 5 below of the hierarchical design. As shown in Figure 6, each 4-Bit Unit consists of a place for each bit unit, plus an additional place for the Decision Logic. When the incoming data is compared to each individual Unit from Figure 2, the Decision Logic is set to one (1) if the current bit value for that unit is larger than the incoming data, so the incoming data “loses” the location. It is set to zero (0) if the current bit value for that unit is smaller than the incoming data, so the incoming data “wins” the location. Each unit shown in Figure 2 is compared to the incoming data individually.

In the case that the incoming data is the same as the current unit locations bit data, the current current unit Decision Logic is set to zero, so the incoming data “wins” the location. The previous data will move down on unit on the stack.

If r/w is switched to one (1), the data will be read. On the next clock cycle, the top item from the Sorted Stack will pop off the stack and be read. All other items on the stack will shift up one unit position. The bottom unit will load a zero (0) because the farthest most bit is connected to ground.

An overall, top-down design style is shown below in Figure 3, Figure 4, and Figure 5.

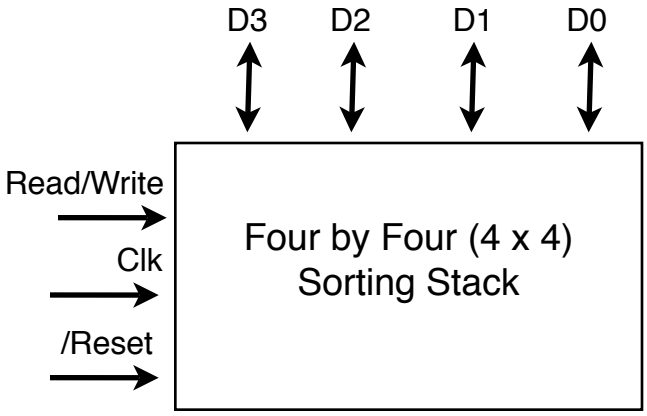

Figure 3: Four by Four (4 x 4) Sorting Stack Block Diagram

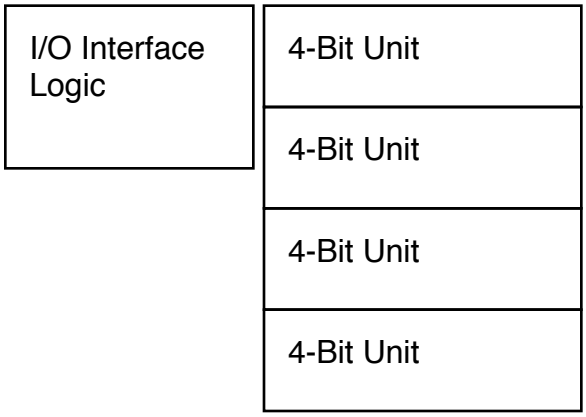

Figure 4: Hierarchical Decomposition of the Four by Four (4 x 4) Sorting Stack

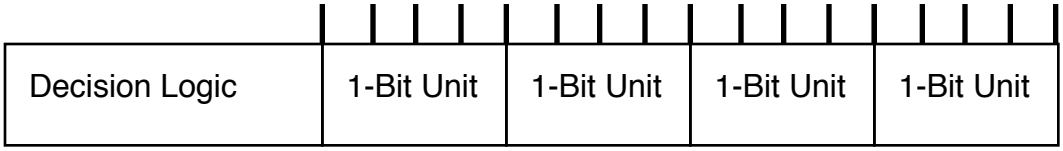

Figure 5: A Hierarchical Decomposition of the 4-Bit Unit

As shown in Figure 5, each 1-Bit unit is broken down into four (4) individual bits which make up the four (4) bit number on the stack.

Implementation Details:

From the design above, we deduced the Truth Tables for the Compare Logic and the Decision Logic. The Decision Logic has already been described as the logic which is attached to the end of each 4-Bit Unit and decides whether the current item on the stack or the incoming data “wins” that bit position. The Compare Logic is the actual logic to determine whether the current unit data or the incoming data is larger. The Compare Logic is made to compare from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). The Truth Tables for the Compare Logic and the Decision Logic are shown below in Table 1 and Table 2.

Table 1: Truth Table for Compare Logic

Cin	N.n	D	Cout	N.n (>, =, <) Data	Action
0	0	0	0	Equal	Pass Cin to Cout
0	0	1	0	N.n < Data	Send 0 to Cout
0	1	0	1	N.n > Data	Send 1 to Cout
0	1	1	0	Equal	Send 0 to Cout
1	0	0	1	N.n > Data	Send 1 to Cout
1	0	1	0	N.n < Data	Send 0 to Cout
1	1	0	1	N.N > Data	Send 1 to Cout
1	1	1	1	Equal	Send 1 to Cout

Table 2: Truth Table for Decision Logic

r/w	AC	CF	Mux 1	Mux 0	Above (>, =, <) Data	Self (<, =, >) Data	Action
0	0	0	1	0	Above is smaller	Self is smaller	Load above
0	0	1	X	X	Above is smaller	Self is greater	Don't care
0	1	0	0	1	Above is greater	Self is smaller	Load data
0	1	1	0	0	Above is greater	Self is greater	Load self
1	0	0	1	1	Read Mode	Read Mode	Load below
1	0	1	1	1	Read Mode	Read Mode	Load below
1	1	0	1	1	Read Mode	Read Mode	Load below
1	1	1	1	1	Read Mode	Read Mode	Load below

From these Truth Tables, K-Maps were very easily constructed for the Compare Logic and the Decision Logic, as shown in Figure 6 and Figure 7. N.n represents the current bit location being represented.

Within the 4-Bit Unit, the Compare In (Cin) is the compare from the previous one (1) bit cell. The Compare Out (Cout) will become the Cin of the next most significant one (1) bit cell. N.n represents the current value held by the unit on the stack, and D represents the data bit for N.

r/w simple indicates whether the current data is being read (1) or written (0). Above Compare (AC) is the bit carried from the above units comparison, telling the current unit whether the above unit was smaller or larger than the incoming data. The Comparison Final (CF) is the final compare of the current unit, which will then be passed on to become the AC of the next significant unit.

Cin	
0	1
0	1
0	1
0	1

N.n	
0	0
0	0
1	1
1	1

D	
0	0
1	1
1	1
0	0

Cout	
0	1
0	0
0	1
1	1

Figure 6: K-Maps for Compare Logic

r/w	
0	1
0	1
0	1
0	1

AC	
0	0
0	0
1	1
1	1

CF	
0	0
1	1
1	1
0	0

Mux 1	
1	1
X	1
0	1
0	1

Mux 0	
0	1
X	1
0	1
1	1

Figure 7: K-Maps for Decision Logic

Since Flip Flops will be used for the remainder of the logic, only the Cout, Mux 1, and Mux 0 logic need to be solved to become Gate-Level Schematics. The solutions for these three K-Maps is shown below.

Cout = $\underline{AC'} + \underline{AB} + \underline{BC'}$
Mux 1 = $\underline{A} + \underline{B'}$
Mux 0 = $\underline{BC'} + \underline{A}$

Mux 1 and Mux 0 are connected, and they both are results of the Decision Logic. The Gate-Level Schematic is shown below in Figure 8. We implemented it two different ways and ultimately chosen the second way, with NOR-XNOR gates, since these gates are faster logically. End connections were placed at every outgoing and incoming connection, and all connections were lined up vertically and horizontally so the units could easily be copied later.

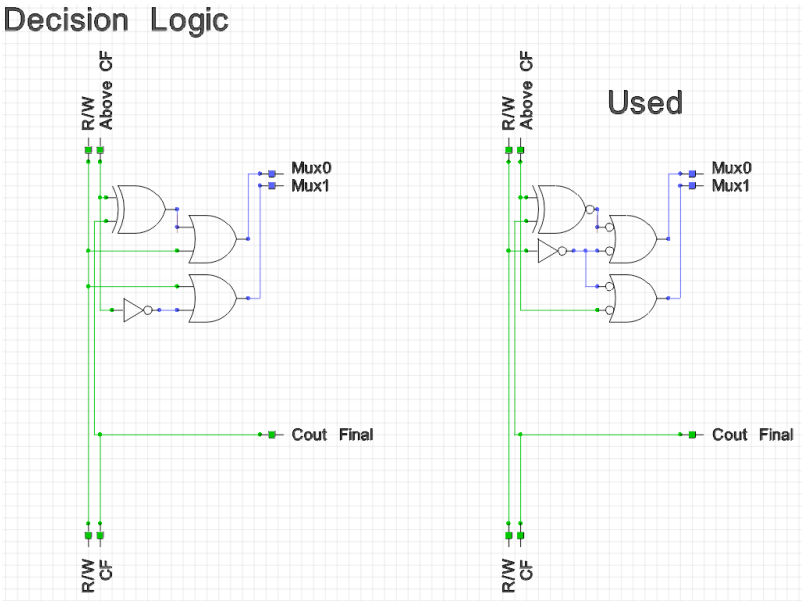

Figure 8: Two Implementations of Decision Logic

For the Compare Logic, which was simply the Cout, we implemented it three different ways: AND-OR gates, NAND-NANDX gates, and a Decoder. We ultimately chose the NAND-NANDX gates since they are the fastest gates, logically.

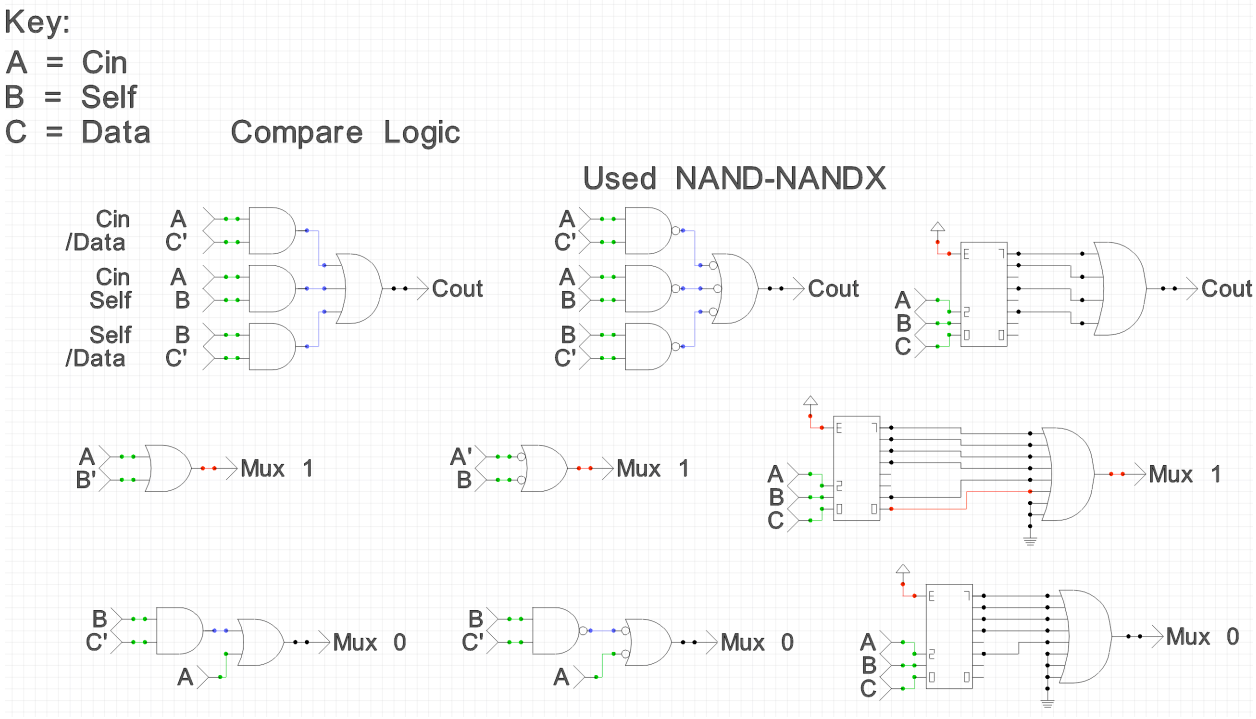

Figure 9: Three Implementations for Each Compare Logic

An entire 1-Bit Cell was constructed by attaching the the Data Flip Flop to the Self line, which was attached to the Compare Logic and also comes back around to the Mux. If

the combination of Mux 1 and Mux 0 comes to 00, Self is loaded. If the combination equals 01, the incoming Data is loaded. If the combination equals 10, Above is loaded. If the combination equals 11, the Below is loaded. Again, the connectors are placed on every outgoing or incoming connection on the edge of the cell and aligned horizontally and vertically to make it easily able to be copied.

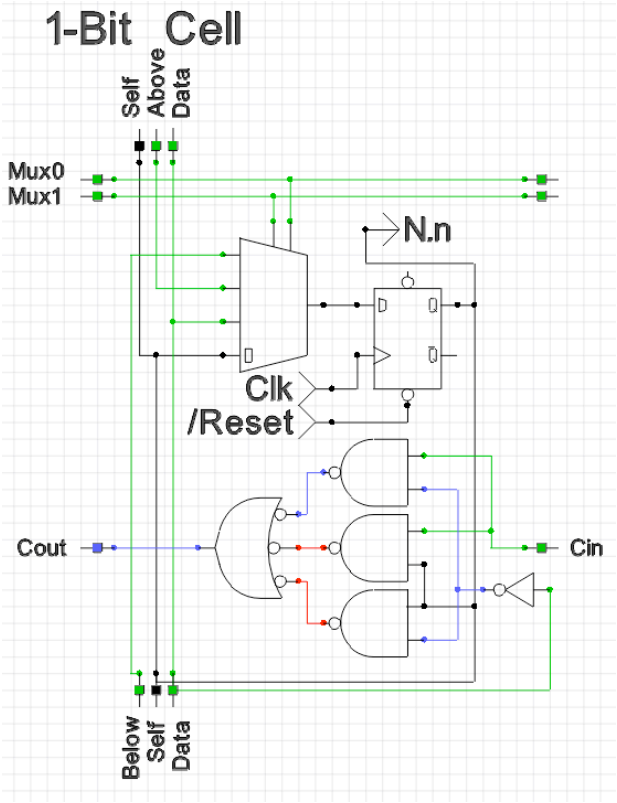

Figure 10: Complete 1-Bit Cell

At this point, the Complete 1-Bit Cell can easily be copied four (4) times, and a Complete 4-Bit Unit can be constructed by adding the Decision Logic to the far right side. Also note that a new line and connectors was added to the far right side of the 4-Bit Unit, as shown in Figure 11, which will allow a ground or power to be connected to the far right Cin. In the 4-Bit Unit, the small n is replaced with the 1-Bit Unit position.

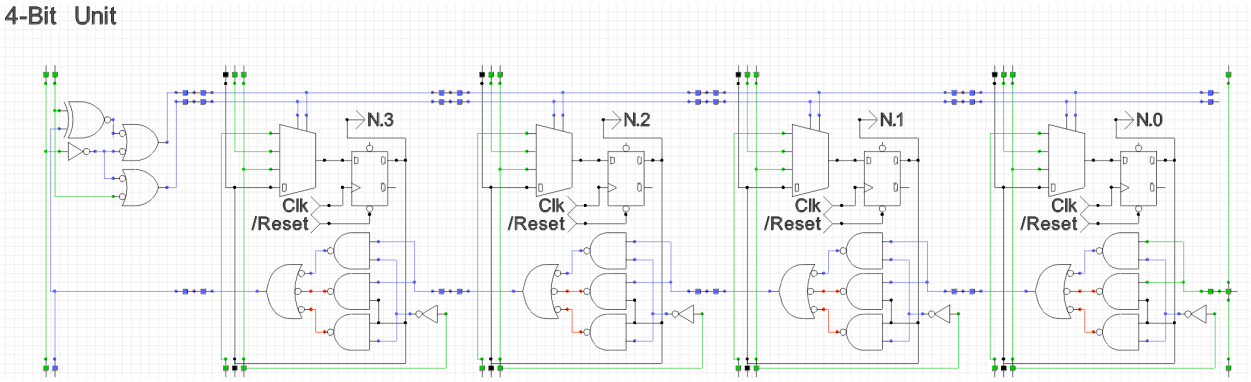

Figure 11: Complete 4-Bit Unit with Decision Logic on Left and Ground Connections on Right

Finally, the 4-Bit Unit was easily copied four more times to make up the four (4) units in the Four by Four (4 x 4) Sorting Stack. The large N is replaced by which unit in the stack that 4-Bit Unit represents. Notice that the N.n labels match the labels in the I/O Interface in Figure 1. The final circuit is shown in Figure 12.

The far right edge connection is tied to ground, so when the current unit value and the incoming data value are equal, the current value is automatically set to “lose”, thus the data is loaded, because the Cin is grounded. Ground is also connected to the Above and Below inputs to the Unit 3 and Unit 0 Mux’s, thus loading a zero (0) when the top item of the stack is popped out. The large R/W, shown in Figure 8 showing the Decision Logic, is connected to the r/w. The Above CF is connected to power, which will make the top unit, Unit 0, realize it is the top unit, so if a larger unit is on the incoming data line, it cannot be placed in an above unit, since Unit 0 is the top-most unit, so the Data must be loaded.

The Data lines are connected to their respect Data inputs, taken directly from the I/O Interface shown in Figure 1. This is the incoming data which is compared to the current data in each unit, so notice that D3, D2, D1, and D0 are all passed from the bottom unit all the way up to the top unit to be compared to each individually.

The N.3, N.2, N.1, and N.0 N variables were renamed to the respective 4-Bit Units which they represented on the Stack.

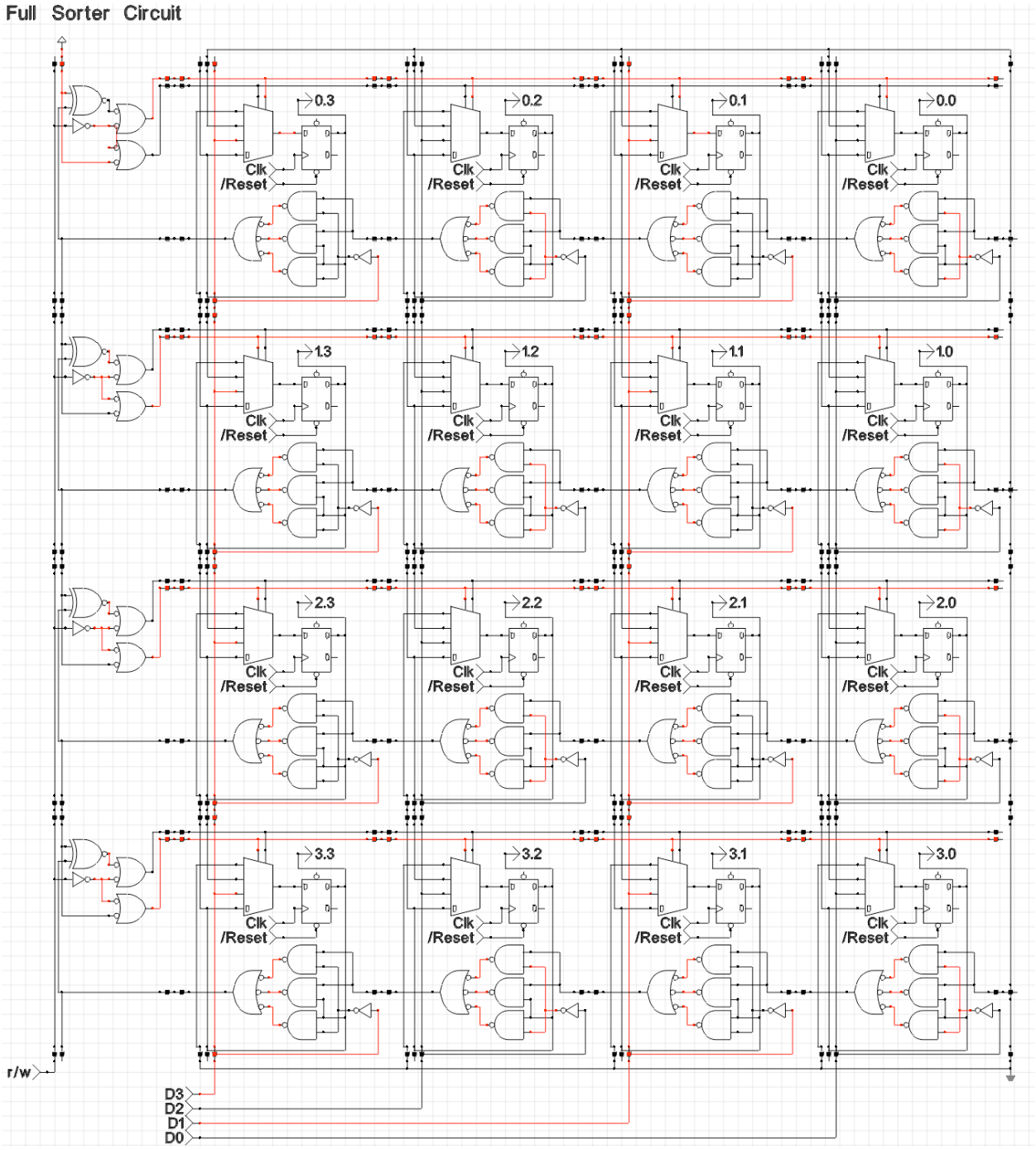

Figure 12: Complete Circuit Design

Conclusion and Suggestions:

After this implementation, the circuit was complete and worked with the I/O Interface. The circuit was tested fully and worked properly. In lab, we showed our circuit to Dr. Kohl and answered his questions about the circuit.

Me and my partner each did an equal amount of work on this project, so I would rate my partners contribution as 50/50 with mine.

Overall, this was the most fun and it was a very fruitful educational experience. It gave me a good perspective on significant design problems and learning how to break larger problems down into smaller problems via hierarchical methods. It was a very rewarding feeling to finally complete this circuit and realize that it worked!