**Lab 4: Combinational Iterative Design**

**Primary Objectives**

1. Analyze and design an iterative combinational system through various circuits

2. Implement the systems using the Logisim software

3. Test to verify the functionality of said systems

*Objective 1 Design*

The overall purpose of this lab is to create a general combinational system that works as a maximum value selector, similarly to Lab 3. This specific max value selector, however, should be usable as an iterative element in more complex designs. This will allow for a max value selector of any bit-length to be made using several of the max value selector elements.

Design #0

This design is identical to the two-bit max value selector which was designed and implemented in Lab 3. Refer to previous lab for information on this design.

Design #1

This design has the purpose as the two-bit max value selector from design #0. This one uses only 2 16-to-1 multiplexers and nothing else. Various designs were attempted and thrown out at first as I tried to understand how to use the four bits of input (A0, A1, B0, and B1) and convert them to output using 16-to-1 multiplexers. Eventually, I realized that the 16 input bits for the C0 and C1 multiplexers corresponded to the binary value of each output dependent on the total value of A1, A0, B1, and B0 written as a 4-bit binary value. By writing the four inputs this way, I was given 16 total outputs, which corresponded to either a 1 or 0 for the value of C1/C0. Then, I just set each multiplexer input to its corresponding constant value of 1 or 0 and made C1/C0 its output. Various testing on the input values revealed that the system worked as intended.

Design #2

This design works very similarly to design #1, this time using only 2-to-1 multiplexers. After working through design #1, I was much more comfortable with multiplexers, so this design went rather quickly. I started with the paths from the four inputs to C1, realizing that C1 was solely dependent on the MSB of each pair of values, which were A1 and B1. That meant that only one multiplexer with A1 on 0 and B1 on 1 and the switch could be used to encode C1. Since C0 was reliant on all four of the inputs, I used two multiplexers in parallel that determined which MSB was greater for the two pairs of inputs. A third multiplexer was then used to take the LSB of the greater pair and output it for C0. After various testing on the input values, this system was determined to work properly.

Design #3

This design also had the same purpose as designs #0-2, but only allowing for the usage of NAND gates. Using the truth table from design #0, I was quickly able to construct a circuit using just NAND gates in the same way I would construct one using basic logic gates. Nothing of note happened during this design process. Testing by changing the input values revealed that the system functioned as intended.

Design #4

This design is intended to be a more generalized version of the max value selectors previously designed. This circuit will determine the maximum value between two bits (A or B) based on which set of bits is greater overall. There are four inputs: A, B, Am, and Bm. A and B are the values of the two bits to be compared, and Am and Bm are data which track whether the A or B series of bits is greater overall. There are three outputs: C, Amo, and Bmo. C takes the value of the greater of the two bits, A and B, while Amo and Bmo continue to communicate which series of bits is greater. This design was constructed by creating a truth table, drawing up k-maps, and using those to write the simplest Boolean expressions for C, Amo, and Bmo. Implementation is demonstrated under Objective 2 below. If designed properly, the truth table of the circuit should be as shown below in Table 1.

Design #5

This design is simply intended to use design #4 to create a max value selector for two 4-bit values. By using four devices with the implementation of design #4 iteratively, a max value selector can be created as intended. With the way design #4 was implemented, a max value selector can now be made for any number of bits, so long as the corresponding number of design #4 devices is used.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **Am** | **Bm** | **C** | **Amo** | **Bmo** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |

Table 1

*Objective 2 Implementation*

Figure 1 below shows one way the system could be implemented via Logisim. The Boolean expressions that represent this circuit are as follows:

C = B ~Am + B Bm + A ~Bm + A Am,

Amo = Am ~Bm + A ~B ~Bm + A ~B Am,

Bmo = ~Am Bm + ~A B ~Am + ~A B Bm.

A diagram of a computer circuit

Description automatically generated

Figure 1 Implementation

*Objective 3 Testing*

To test this circuit, I tried various individual combinations of inputs manually in the circuit. I then checked their outputs to see if they represented the maximum value of the inputs and, consequently, matched up with the truth table. The testing was successful, ensuring that the circuit works as intended. Additionally, Log 1 shows a portion of the data taken from testing, which shows that the circuit was implemented properly.

A screenshot of a computer code

Description automatically generated

Log 1

**Conclusion**

The lab seems to be successful. Each design gave me the results I expected after going through various trials and re-implementations. As for how I would implement a 32-bit max value selector, I would use the exact same method I used in design #5, just with 32 iterations of design #4 instead of four. This would work much more efficient than using any of the other methods, given that this is the only iterative approach and simply requires duplicating a few elements in the circuit.