Lab 4: Combinational Design

**Primary Objectives:**

1. Get experience implementing larger devices using a general iterative approach under a variety of constraints.
2. Implement and test a 2-bit maximum value selector that uses two 1-bit 16-to-1 multiplexers.
3. Implement and test a 2-bit maximum value selector that uses only 1-bit 2-to-1 multiplexers.
4. Implement and test a 2-bit maximum value selector that uses only NAND gates.
5. Implement and test a 1-bit maximum value selector designed to be used to solve larger problems using an iterative design approach.
6. Implement and test a 4-bit maximum value selector using my 1-bit bit maximum value selector designed for iterative implementation.

**Design**

Table 1: Symbol Mapping for Designs 1, 2, 3, 5

|  |  |
| --- | --- |
| Name | Symbol |
| Input A | A |
| Input B | B |
| Least Significant Bit of A | A0 |
| Most Significant Bit of A | A1 |
| Least Significant Bit of B | B0 |
| Most Significant Bit of B | B1 |
| Output C | C |
| Least Significant Bit of C | C0 |
| Most Significant Bit of C | C1 |

Table 1 lists the symbols used in designs 1, 2, 3, 5 later in the document.

Table 2: Symbol Mapping for Design 4

|  |  |
| --- | --- |
| Name | Symbol |
| Input A | A |
| Input B | B |
| Returned Bit | RC |
| Input Switch | IS |
| Output Switch | OS |

Table 2 lists the symbols used in design 4 later in the document.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A1 | A0 | B1 | B0 | C1 | C0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

Table 3: Truth table for designs 1, 2, 3

Table 3 describes all possible outcomes for designs 1, 2, 3, the table specifically shows that C will always output the greater of A and B.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IS | A | B | RB | OS |
| 00 | 0 | 0 | 0 | 00 |
| 00 | 0 | 1 | 1 | 01 |
| 00 | 1 | 0 | 1 | 10 |
| 00 | 1 | 1 | 1 | 11 |
| 01 | 0 | 0 | 0 | 01 |
| 01 | 0 | 1 | 0 | 01 |
| 01 | 1 | 0 | 1 | 01 |
| 01 | 1 | 1 | 1 | 01 |
| 10 | 0 | 0 | 0 | 10 |
| 10 | 0 | 1 | 1 | 10 |
| 10 | 1 | 0 | 0 | 10 |
| 10 | 1 | 1 | 1 | 10 |
| 11 | 0 | 0 | 0 | 00 |
| 11 | 0 | 1 | 1 | 01 |
| 11 | 1 | 0 | 1 | 10 |
| 11 | 1 | 1 | 1 | 11 |

Table 4: Truth table for design 4

Table 4 describes all possible outcomes for design 4, specifically the table shows that design 4 will always return the greater bit as RB and pass a 2-bit value in the order of AB forward while IS is either 00 or 11. Otherwise, if IS is 01, design 2 will return A and pass 01 forward and if IS is 10, design 2 will return B and pass 10 forward.

Notes about designs

Design 1 was built by following the truth table and using each mux for one bit of the output.

Design 2 was built by using 1-bit 2-to-1 multiplexer equivalents of 2-input AND and OR gates and following the general design of the circuit from Lab 3. (For more detail, see figure 2 below).

Design 3 was built using NAND gates following the circuit from Lab 3 and replacing every component with their NAND gate equivalent (through inverting their inputs and putting multiple NAND gates together) and then simplifying with bubble-to-bubble design.

Design 5 was built by stringing together multiple design 4s. (Design 4 notes below)

Design process of design 4

Components of Design 4

A switch (and input) that changes from default path to something similar to Design 2's check bit 0

Default path: When not disabled, checks both input bits and returns 1 if either is on (an or gate)

Switch should have 2-bit input and output to encode check both, check only A, check only B (potentially only 1 bit input/output is possible)

A: Bit from A

B: Bit from B

IS: Input Switch

OS: Output Switch

RB: Returned Bit

Realized that I wasn't limited to 2 input mux like I was in Design 2, so I used 4 input mux

Similar solution for returning the 2-bit number

**Implementation & Testing**

Design 1

Figure 1: Implemented 2-bit maximum value selector that uses two 1-bit 16-to-1 multiplexers.



The 2-bit maximum value selector that uses two 1-bit 16-to-1 multiplexers is implemented as seen above in figure 1.

Table 5: Log of outputs from design 1

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 00 | 00 | 00 |
| 01 | 00 | 01 |
| 11 | 00 | 11 |
| 10 | 00 | 10 |
| 10 | 01 | 10 |
| 11 | 01 | 11 |
| 01 | 01 | 01 |
| 00 | 01 | 01 |
| 00 | 11 | 11 |
| 01 | 11 | 11 |
| 11 | 11 | 11 |
| 10 | 11 | 11 |
| 10 | 10 | 10 |
| 11 | 10 | 11 |
| 01 | 10 | 10 |
| 00 | 10 | 10 |

The log is similar to the truth table seen above in table 3. This demonstrates that the implemented design functions appropriately.

Design 2

Figure 2: Implemented 2-bit maximum value selector that uses only 1-bit 2-to-1 multiplexers.



The 2-bit maximum value selector that uses only 1-bit 2-to-1 multiplexers is implemented as seen above in figure 2.

Table 6: Log of outputs from design 2

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 00 | 00 | 00 |
| 01 | 00 | 01 |
| 11 | 00 | 11 |
| 10 | 00 | 10 |
| 10 | 01 | 10 |
| 11 | 01 | 11 |
| 01 | 01 | 01 |
| 00 | 01 | 01 |
| 00 | 11 | 11 |
| 01 | 11 | 11 |
| 11 | 11 | 11 |
| 10 | 11 | 11 |
| 10 | 10 | 10 |
| 11 | 10 | 11 |
| 01 | 10 | 10 |
| 00 | 10 | 10 |

The log is similar to the truth table seen above in table 3. This demonstrates that the implemented design functions appropriately.

Design 3

Figure 3: Implemented 2-bit maximum value selector that uses only NAND gates.



The 2-bit maximum value selector that uses only NAND gates is implemented as seen above in figure 3.

Table 7: Log of outputs from design 3

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 00 | 00 | 00 |
| 01 | 00 | 01 |
| 11 | 00 | 11 |
| 10 | 00 | 10 |
| 10 | 01 | 10 |
| 11 | 01 | 11 |
| 01 | 01 | 01 |
| 00 | 01 | 01 |
| 00 | 11 | 11 |
| 01 | 11 | 11 |
| 11 | 11 | 11 |
| 10 | 11 | 11 |
| 10 | 10 | 10 |
| 11 | 10 | 11 |
| 01 | 10 | 10 |
| 00 | 10 | 10 |

The log is similar to the truth table seen above in table 3. This demonstrates that the implemented design functions appropriately.

Design 4

Figure 4: Implemented 1-bit maximum value selector designed to be used to solve larger problems using an iterative design approach.



The 1-bit maximum value selector designed to be used to solve larger problems using an iterative design approach is implemented as seen above in figure 4.

Table 8: Log of outputs from design 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IS | A | B | RB | OS |
| 00 | 0 | 0 | 0 | 00 |
| 00 | 0 | 1 | 1 | 10 |
| 00 | 1 | 1 | 1 | 11 |
| 00 | 1 | 0 | 1 | 01 |
| 01 | 1 | 0 | 1 | 01 |
| 01 | 1 | 1 | 1 | 01 |
| 01 | 0 | 1 | 0 | 01 |
| 01 | 0 | 0 | 0 | 01 |
| 11 | 0 | 0 | 0 | 00 |
| 11 | 0 | 1 | 1 | 10 |
| 11 | 1 | 1 | 1 | 11 |
| 11 | 1 | 0 | 1 | 01 |
| 10 | 1 | 0 | 0 | 10 |
| 10 | 1 | 1 | 1 | 10 |
| 10 | 0 | 1 | 1 | 10 |
| 10 | 0 | 0 | 0 | 10 |

The log is similar to the truth table seen above in table 4. This demonstrates that the implemented design functions appropriately.

Design 5

Figure 5: Implemented 4-bit maximum value selector using my 1-bit bit maximum value selector designed for iterative implementation.



The 4-bit maximum value selector using my 1-bit bit maximum value selector designed for iterative implementation is implemented as seen above in figure 5.

Table 9: Log of outputs from design 5

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 1000 | 1000 | 1000 |
| 1010 | 1000 | 1010 |
| 1011 | 1000 | 1011 |
| 1001 | 1000 | 1001 |
| 1001 | 1001 | 1001 |
| 1011 | 1001 | 1011 |
| 1010 | 1001 | 1010 |
| 1000 | 1001 | 1001 |
| 1000 | 1011 | 1011 |
| 1010 | 1011 | 1011 |
| 1011 | 1011 | 1011 |
| 1001 | 1011 | 1011 |
| 1001 | 1010 | 1010 |
| 1011 | 1010 | 1011 |
| 1010 | 1010 | 1010 |
| 1000 | 1010 | 1010 |

The log demonstrates the design’s ability to decide which input is greater beyond the most significant bits with both the most significant bits in both inputs set to 1 and 0 respectively. The log also demonstrates all remaining functionality through the changing of the two least significant bits.

**Conclusion**

Experience implementing larger devices using a general iterative approach under a variety of constraints has been gained, and the devices works as expected.

**Additional Mandatory Discussion**

I would use a design similar to design 5 to implement a 32-bit maximum value selector. This is because it is the simplest option to simply reuse a single device that can be strung together to create such a device. This solution may not be the most space efficient or time efficient solution; however, without any specified space or time constraints, I decided that simplicity would be the most important.