Table of Contents

# Project 1

## 1a: Eight Bit Four-to-One Multiplexer………………………………………….pg. 2

## 1b: Eight Bit Full Adder………………………………………………………………..pg. 5

# Project 2

## 2a: Logical Unit (LU)…………………………………………………………………….pg. 9

## 2b: Arithmetic Unit (AU)………………………………………………………………pg. 16

## 2c: Arithmetic Logic Unit (ALU)……………………………………………………pg. 21

# Project 3

## 3a: Instruction Register (IR)…………………………………………………………pg. 26

## 3b: Register File (RF)…………………………………………………………………….pg. 29

## 3c: Memory……………………………………………………………………………………pg. 35

# Project 4

## 4a. Memory with MDR and MAR……………………………………………………..pg. 36

## 4b. Control Unit……………………………………………………………………………...pg. 38

## 4c. Central Processing Unit (CPU…………………………………………………….pg. 39

# References**……………………………………………………………………………………..……..pg. 41**

## 

Project 1

# 1a: Eight Bit Four-to-One Multiplexer

## Specification

A multiplexer is a unit that selects one input line out of several and directs it to the output. The select lines dictate which line is outputted. Typically a multiplexer of *2n* inputs has *n* select lines.

The 2-to-1 multiplexer, defined by equation Y = S’I0 + S I1, is extrapolated to a 4-to-1 multiplexer to be used as a sub design. The 4-to-1 multiplexer takes four input lines, I0…I3, directs one of them to output Y, based on select lines S0 and S1. Let S represent the two select lines and Y, the output, represent the input that is selected, which yields the following truth table:

#### ***Table 1a.1: Single Bit*** Four-to-One ***Multiplexer Truth Table***

|  |  |  |
| --- | --- | --- |
| **S1** | **S0** | **Y** |
| 0 | 0 | I0 |
| 0 | 1 | I1 |
| 1 | 0 | I2 |
| 1 | 1 | I3 |

(Mano & Kime, 132).

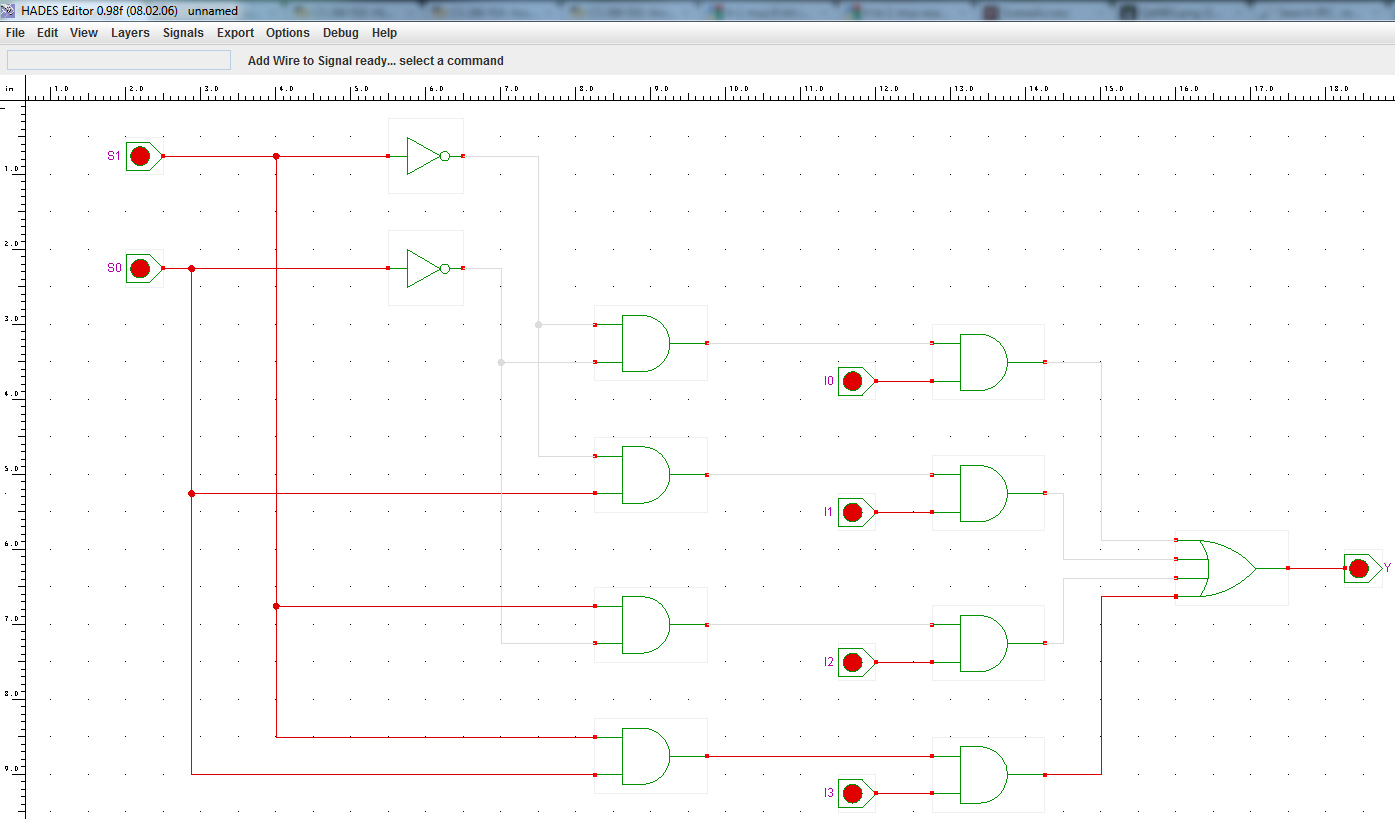
This truth table will also extend to the 8-bit 4-to-1 multiplexer, where I0…I3, the series of inputs, will be 8-bit busses, or vectors, instead of single bits.

In order to accompany 8-bit bus inputs and outputs, four busses - yielding 32 bits – are inputted to eight single bit 4-to-1 multiplexers where the 8 bits of an input are selected for the one 8-bit output based on the two select lines.

## Design

Single bit 2-to-1 multiplexers feature a 1-to-2 decoder for the select line, two AND gates to form enabling circuits with the select lines and input, and then feed into an OR gate to produce the output Y. The 4-to-1 multiplexer is composed of two 2-to-1 multiplexers. Thus, a 2-to-4 decoder is used with the select lines, so two lines give four binary combinations to choose out of the four inputs. These select lines AND with the inputs to produce enabling circuits, and each output directs to an OR gate that yields the output Y. This is implemented in the following 4-to-1 multiplexer sub design:

#### Figure 1a.1: Single Bit Four-to-One Multiplexer Circuit Sub Design

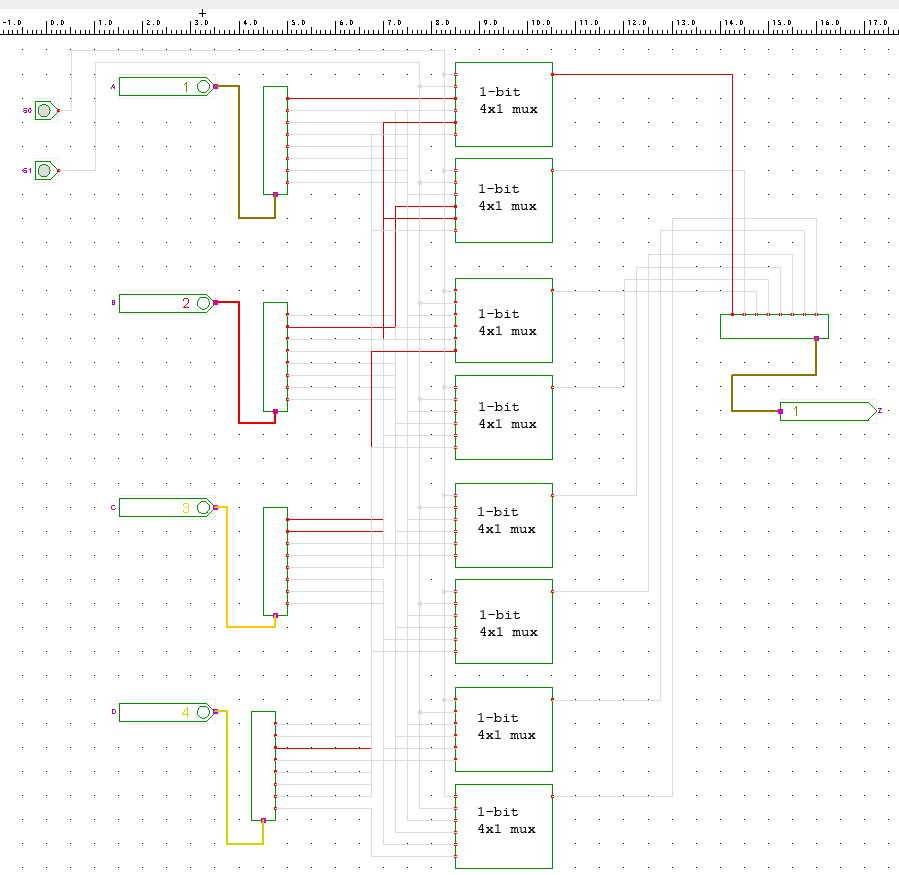


This sub design is employed by the full 8-bit multiplexer circuit. Eight of these single bit 4-to-1 multiplexers are used in unison to choose one of the four inputs.

The input busses are directed to an expander that split the input into individual bits. Input A, which is 8 bits, feeds each of its bits to the I0 input of the single bit multiplexer design. Input B feeds each of its bits into the I1 input of each sub design, and so on for inputs C and D. The bits are then processed by the multiplexer logic and directed to a merger where the bits are delivered as the 8-bit output Z. Note that the sub design labels in Figure 1a.2 were added for clarity with a graphic editing program after being developed in Hades; no other additional changes were made in the editing program.

This is the full circuit design of the 8-bit multiplexer:

#### Figure 1a.2: Eight Bit Four-to-One Multiplexer Circuit



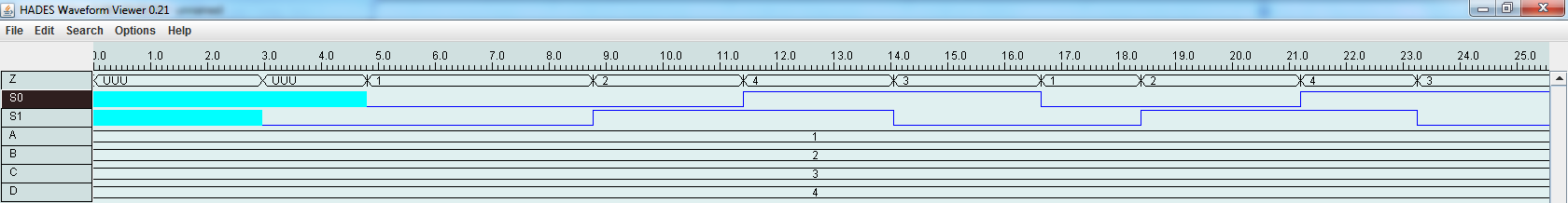
## Verification

The sub design is verified first to give confidence to the full design. The waveform for the single bit 4-to-1 multiplexer is exhaustive. It was produced using clock generators set on different periods/frequencies. Every possible combination of the select lines and inputs are generated in order to confirm the circuit’s behavior. The results of the waveform are expected, where Y shifts value to the specified input based on the select lines. All positive and negative combinations are explored, correspond to the truth table, and can be seen in this figure:

#### Figure 1a.3 Single Bit Four-to-One Multiplexer Waveform

It is clear that the single bit multiplexer performs as expected. To confirm the 8-bit multiplexer’s performance, a waveform is produced using all combinations of the select lines to make sure that the appropriate output is being directed to output Z. The following waveform confirms the functionality of the full circuit. Output Z switches between 1, 2, 3, and 4 in conjunction to the select lines and truth table.

#### Figure 1a.4 Eight Bit Four-to-One Multiplexer Waveform



The multiplexer circuit has thus been verified. With the full confidence of the single bit multiplexer, and the waveform exploring the full 8-bit multiplexer, it is clear that the circuitry is meeting the specifications.

# 1b: Eight Bit Adder

## Specification

The full adder, comprised of two half adders, takes three 1-bit binary numbers as input and two one-bit binary numbers are computed as output. X and Y represent the input binary numbers to add, and Z represents the input carry. Output S represents the sum of X and Y and output C represents the “next” carry. The carry can be, and is later, used in a series of adders; as input (i.e. Z) from a previous circuit and to feed another circuit (i.e. C). This gives the truth table:

#### Table 1b.1: Single Bit Full Adder Truth Table

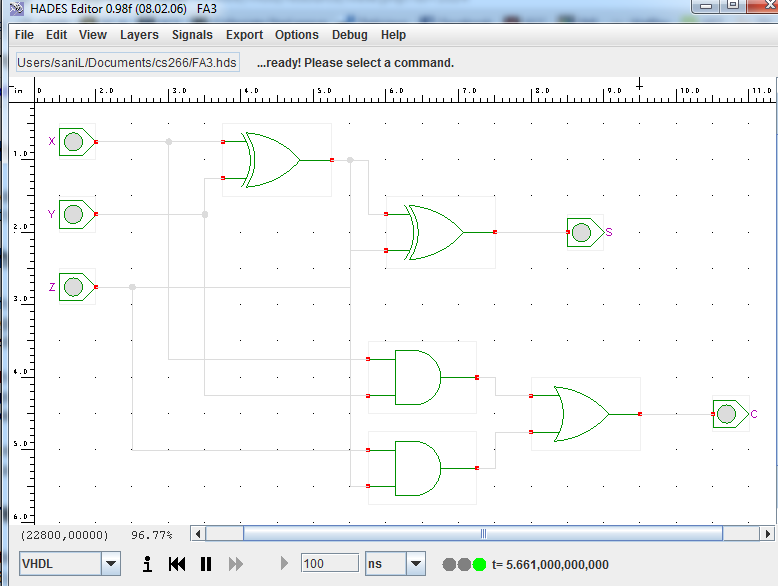
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **X** | **Y** | **Z** | **C** | **S** |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

(Mano & Kime, 152-153).

This is the sub design used to achieve computation between two 8-bit bus inputs. A ripple adder is developed with eight full adders that feed into each other. This will produce two outputs, the 8-bit sum and the carry for the next computation.

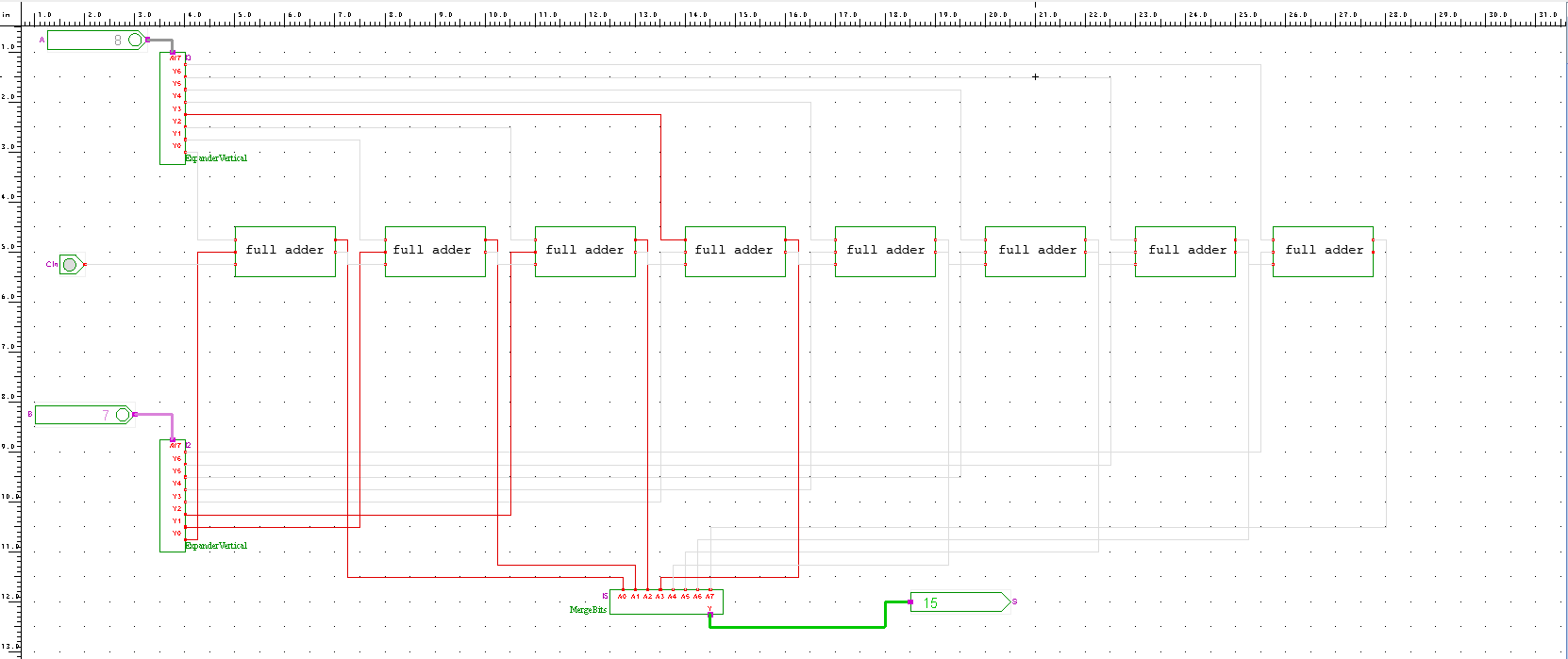
## Design

From the truth table, equations for S and C emerge. These equations are implemented in the sub design. The full adder incorporates two XNOR gates, two AND gates, and an OR gate:

Figure 1b.1: Full Adder Circuit Sub Design

This sub design is replicated eight times in order to produce the ripple carry adder. The inputs are expanded into individual bits and fed into the appropriate adder. A0 will feed into the first full adder, A1 in the second, and so on. The carry-in is provided for the first full adder, and the carry out of the first full adder feeds into the carry-in of the next adder. The outputs of the adders are merged to deliver the summed output, with the last adder providing a carry out. Together, this forms the arithmetic circuit seen here (note that labels are added after Hades in this design as well):

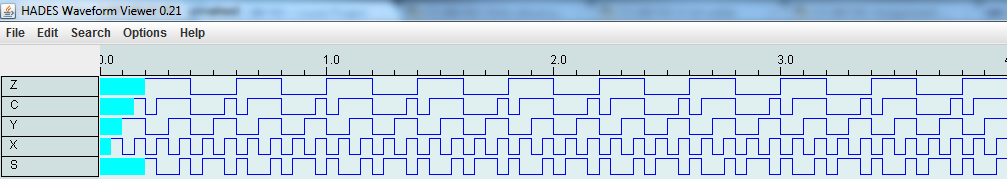
#### Figure 1b.2 Eight Bit Full Adder Circuit



## Verification

The waveform for the sub design, the full adder, is exhaustive. Each combination documented in the truth table is found here and repeats. Clock generators were also used on different periods/frequencies to achieve each permutation. The results conform to the expectations. This gives confidence to the full design and confirms the sub design is working per specification.

#### Figure 1b.3: Single Bit Full Adder Waveform



To verify the functionality of the 8-bit full adder, a few test cases must be developed. First, random numbers will be added together to get an idea of the functionality.

#### Figure 1b.4: Eight Bit Full Adder Waveform, Testing Round One

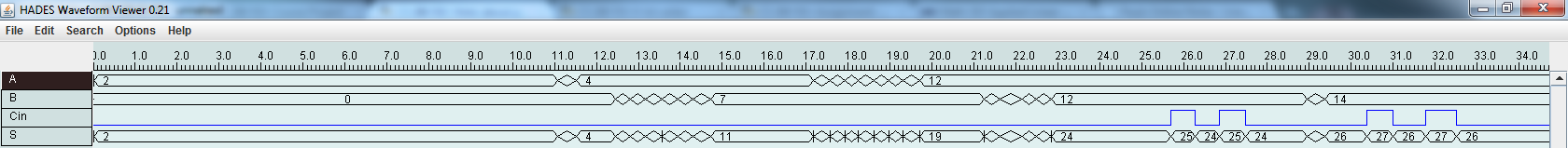


Figure 1b.4 displays the waveform for basic tests. It is clearly shown that 2+0 = 2, 4+7 = 11, 12+7 = 19, 12+12 = 24, etc. The circuit is operating well so far.

#### Figure 1b.5: Eight Bit Full Adder Waveform, Testing Round Two

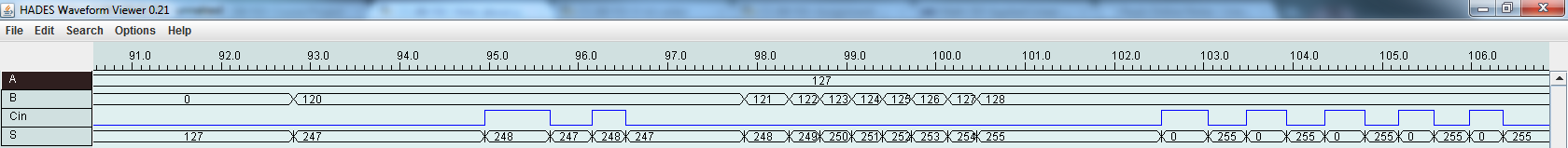


Figure 1b.5 shows the waveform for a more difficult test. A and B are incremented to produce a sum of 255. As expected, when the output sum is greater than 255, it goes back to 0.

#### Figure 1b.6: Eight Bit Full Adder Waveform, Testing Round Three

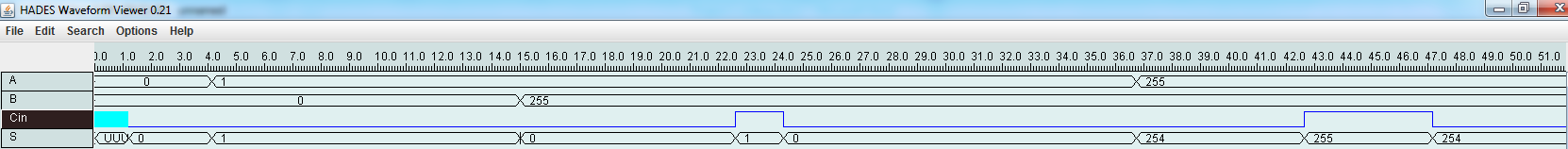


Figure 1b.5 shows the waveform for the last few tests. Again, testing around 255 and and 0, it can be seen that the circuit operating as expected.

Though the eight bit adder waveforms are not exhaustive, the single bit adder is. This gives a large amount of confidence to the design of the second half of the project. The values chosen to be computed come out as expected, even around boundary conditions.

Thus, it has been verified that the eight bit full adder is functioning per specification.

Project 2

# ALU Introduction

The arithmetic logic unit is a device that performs arithmetic and logical operations on inputs based upon a select input. This design will house the arithmetic functions in an arithmetic unit (AU) and the logical functions in a logical unit (LU) to form the final arithmetic logic unit (ALU).

The select input is a 3-bit bus name alu\_sel that will feed into a 3-to-8 decoder to enable the desired functions in the AU or LU. The operations will be performed on 8-bit inputs A and B. There are two additional single bit inputs, C\_In for arithmetic functions and an enable for the ALU. There is an 8-bit output bus M that represents the result of the operations, and single bit outputs M7, C\_Out, and V for sign bit, carry out, and overflow respectively.

This write-up will break down the project by LU and AU, and follow a bottom up approach by elaborating how the sub components form the larger digital system.

# 2a: Logical Unit (LU)

## Specification

The logical unit must perform four different operations: NOT, OR, XOR, AND. It must perform these operations based upon alu\_sel, as can be seen in Table 2a.1. The operations are performed on input 8-bit busses A and B, and there will be an 8-bit bus result output M.

#### Table 2a.1: ALU Select and Corresponding LU Operations

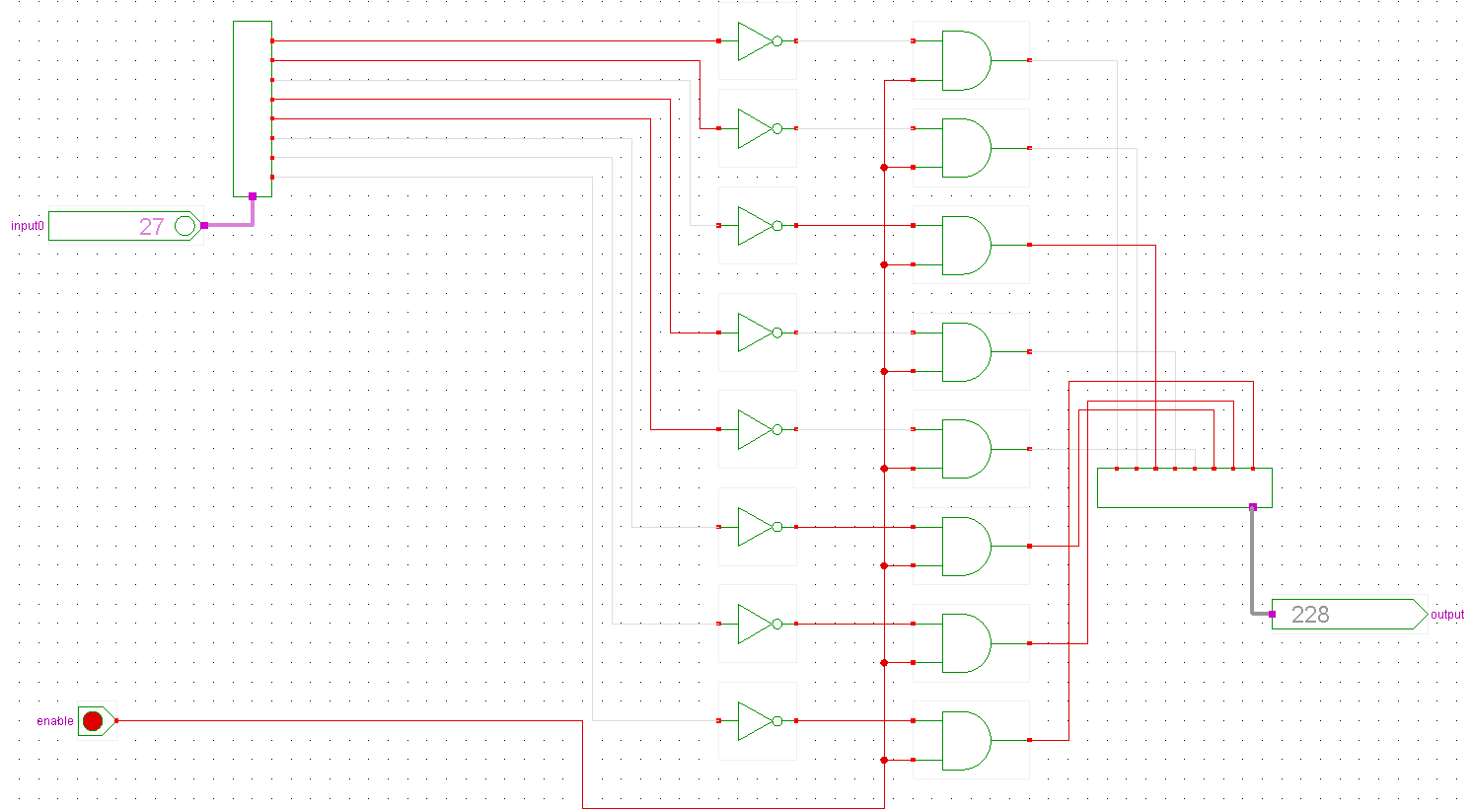
|  |  |
| --- | --- |
| Alu\_sel | Operation |
| 0 | ~A |
| 1 | A OR B |
| 4 | A XOR B |
| 5 | A AND B |

## Design

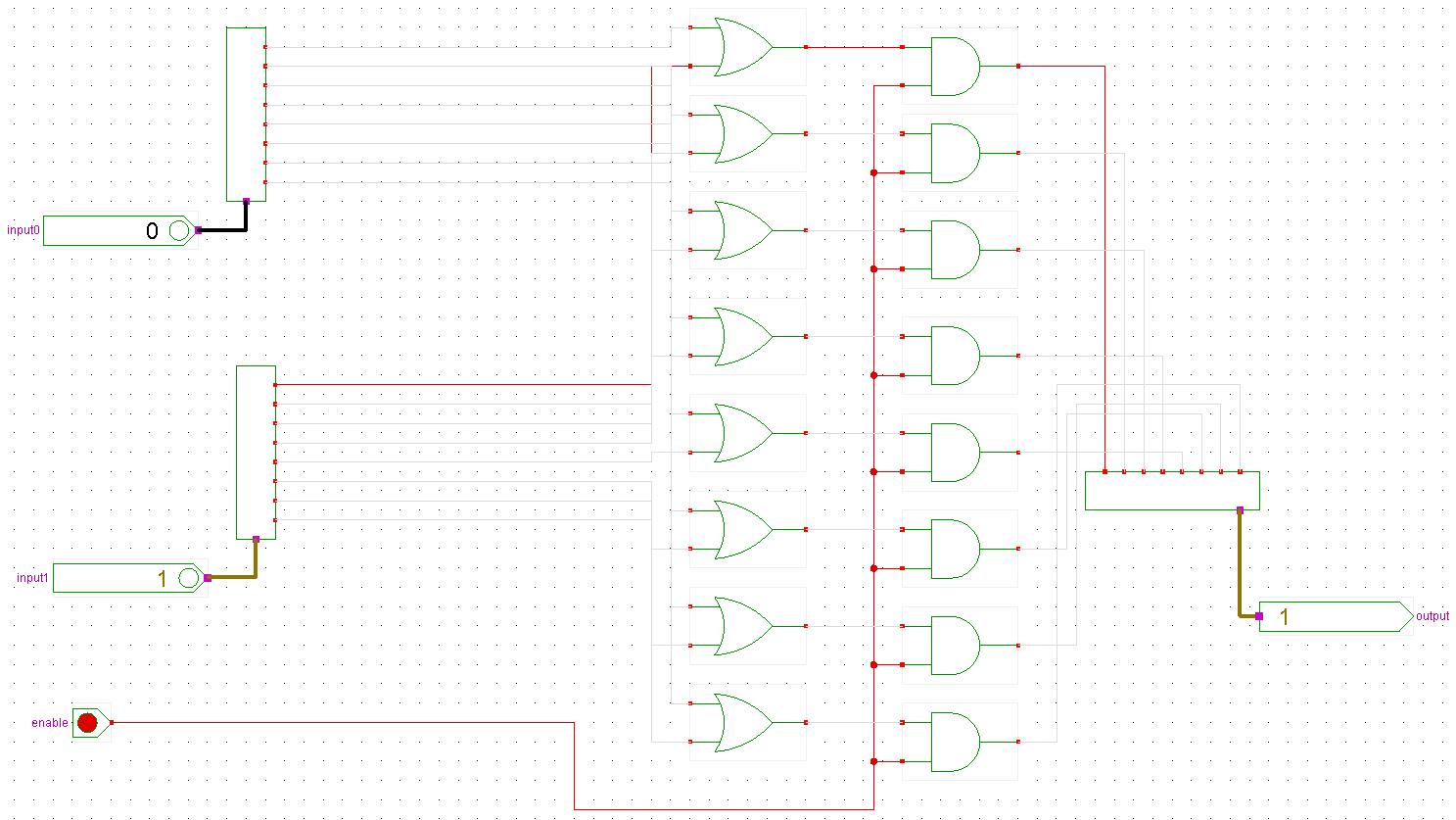
To perform the logical operations desired on the given inputs, 8-bit NOT, OR, XOR, and AND devices are developed. The devices must also have an enable so the operation follows through when appropriate.

The design of these circuits is quite straight forward, and follows the same pattern. The 8-bit input busses are expanded and each bit is compared using the appropriate logic gate. The results from each ith bit logic gate are AND’ed with a single bit to form the enabling circuits. Since this design is repeated for each logical operation, this explanation applies to each of them.

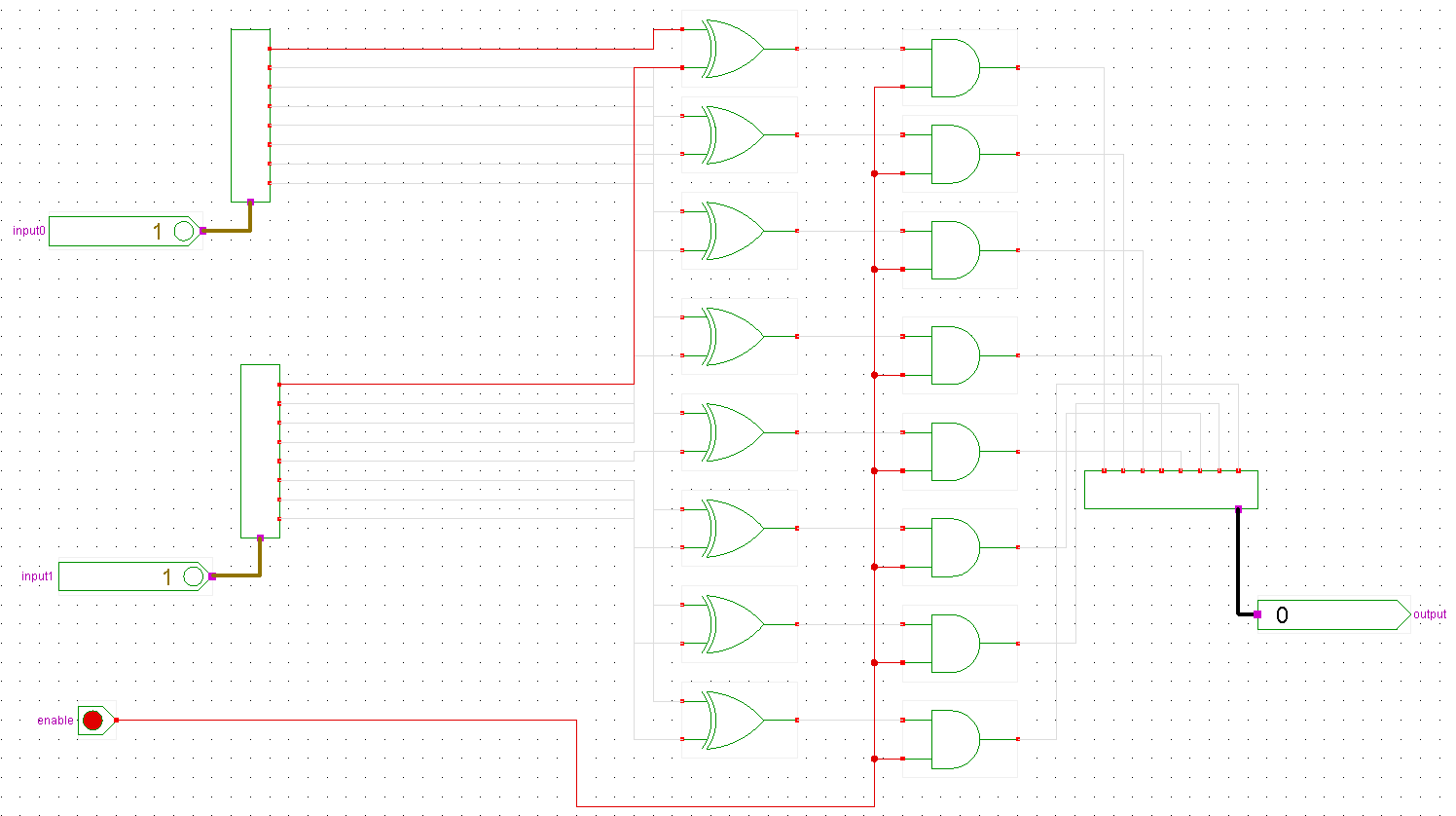
#### Figure 2a.1: Eight Bit NOT Circuit Sub Design



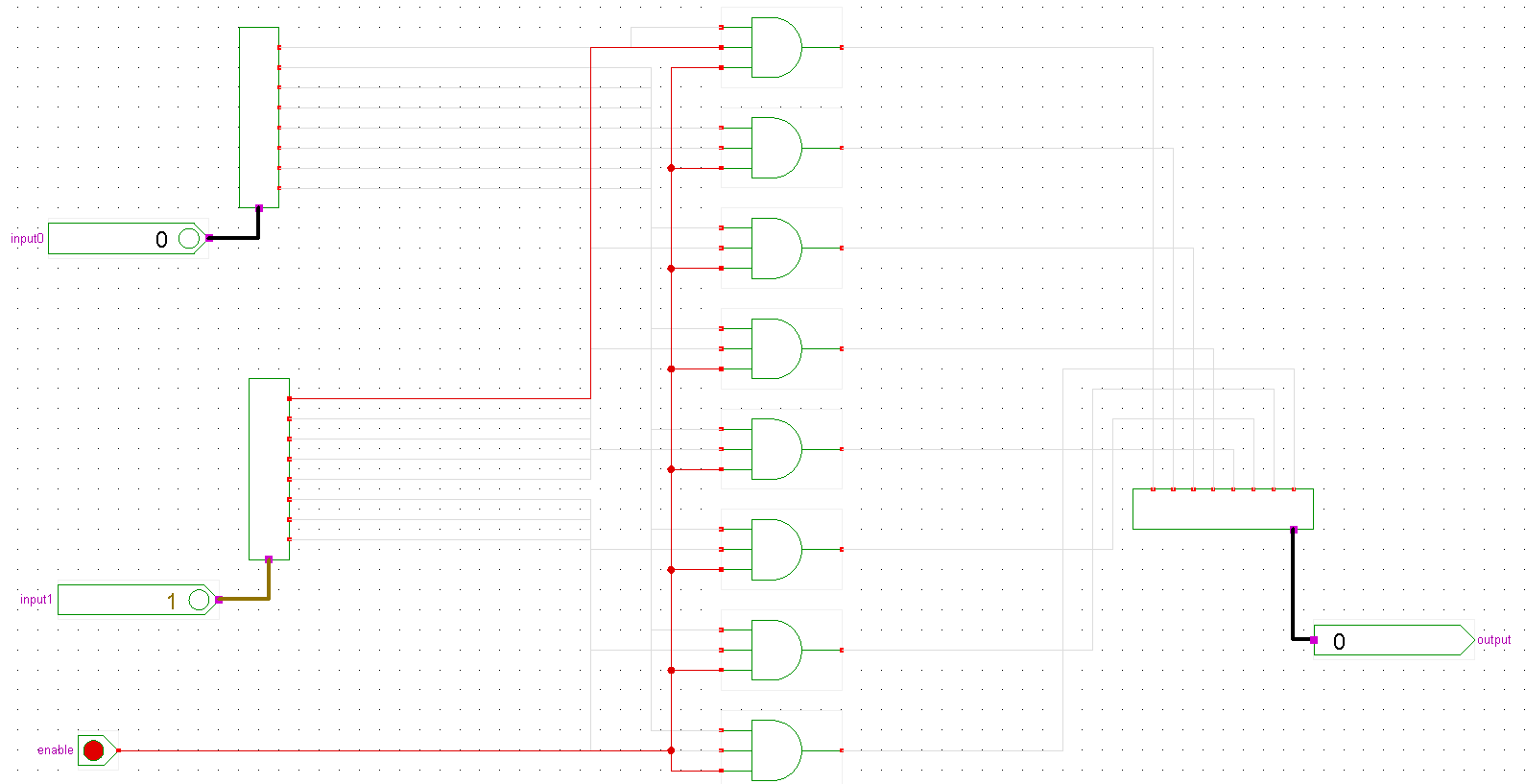
#### Figure 2a.2: Eight Bit OR Circuit Sub Design



#### Figure 2a.3: Eight Bit XOR Circuit Sub Design



#### Figure 2a.4: Eight Bit AND Circuit Sub Design



It is perhaps worth noting that with the 8-bit AND circuit, 3 input AND gates are used with the enabler. Appropriate test cases are also displayed in the figures, though these will be investigated further later.

These four devices now allow the formation of the LU. Four single bit input pins are used to represent alu\_sel and feed into the enables of the logical devices. The two input busses feed into the input busses of the devices. Three single bit OR gates are used with the input enables and alu\_sel to allow the desired result, and three 8-bit OR devices are used to choose the result from the logic operations.

#### Figure 2a.5: Logical Unit Circuit

## Verification

First the 8-bit logic devices are verified. Since these devices operate on 8-bits, exhaustive testing is not feasible. However, since they are dependent on the fundamental gates and the design is rather simple, there is confidence in basic testing. The tests for each logic device will consist of the first permutations of the eight bits, similar as if testing with single bits. Later permutations of the eight bits, and the enable, will also be tested.

#### Figure 2a.6: Eight Bit NOT Waveform

This waveform is summarized in the following table. The test cases accurately describe the expected behavior of an 8-bit NOT logic device. The enable can be seen working.

#### Table 2a.2: Eight Bit NOT Test Case Results

|  |  |  |
| --- | --- | --- |
| Enable | Input0 | Output |
| 0 | 0 | 0 |
| 1 | 0 | 255 |
| 1 | 1 | 254 |
| 1 | 2 | 253 |
| 1 | 1 | 254 |
| 1 | 0 | 255 |
| 1 | 255 | 0 |
| 1 | 254 | 1 |
| 1 | 253 | 2 |
| 1 | 252 | 3 |
| 0 | 252 | 0 |
| 1 | 252 | 3 |

#### Figure 2a.7: Eight Bit OR Waveform

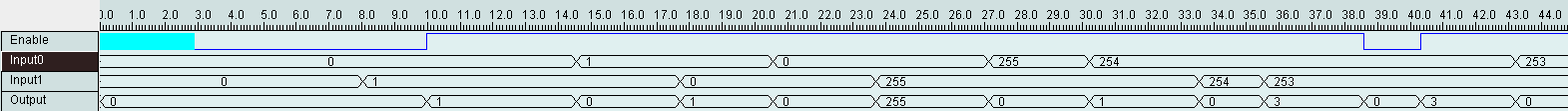
This waveform is summarized in the following table. The test cases accurately describe the expected behavior of an 8-bit OR logic device. The enable can be seen working.

#### Table 2a.3: Eight Bit OR Test Case Results

|  |  |  |  |
| --- | --- | --- | --- |
| Enable | Input0 | Input1 | Output |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 255 | 255 |
| 1 | 0 | 254 | 254 |
| 1 | 255 | 254 | 255 |
| 1 | 254 | 254 | 254 |
| 1 | 254 | 253 | 255 |
| 1 | 254 | 252 | 254 |
| 1 | 253 | 252 | 253 |
| 1 | 252 | 252 | 252 |
| 0 | 252 | 252 | 0 |
| 1 | 0 | 0 | 0 |

#### 

#### Figure 2a.8: Eight Bit XOR Waveform

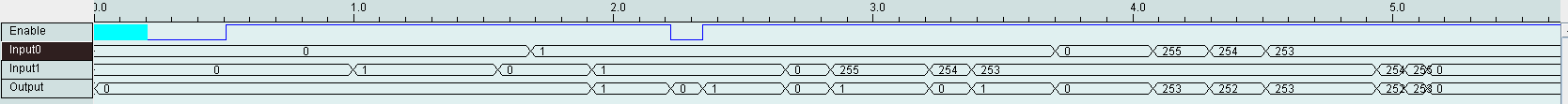


This waveform is summarized in the following table. The test cases accurately describe the expected behavior of an 8-bit XOR logic device. The enable can be seen working.

#### Table 2a.4: Eight Bit XOR Test Case Results

|  |  |  |  |
| --- | --- | --- | --- |
| Enable | Input0 | Input1 | Output |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 255 | 255 |
| 1 | 255 | 255 | 0 |
| 1 | 254 | 255 | 1 |
| 1 | 254 | 254 | 0 |
| 1 | 254 | 253 | 3 |
| 0 | 254 | 253 | 0 |
| 1 | 254 | 253 | 3 |
| 1 | 253 | 253 | 0 |

#### Figure 2a.9: Eight Bit AND Waveform



This waveform is summarized in the following table. The test cases accurately describe the expected behavior of an 8-bit AND logic device. The enable can be seen working.

#### Table 2a.5: Eight Bit AND Test Case Results

|  |  |  |  |
| --- | --- | --- | --- |
| Enable | Input0 | Input1 | Output |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 255 | 1 |
| 1 | 1 | 254 | 0 |
| 1 | 1 | 253 | 1 |
| 1 | 0 | 253 | 0 |
| 1 | 255 | 253 | 253 |
| 1 | 254 | 253 | 252 |
| 1 | 253 | 253 | 253 |

Since all logic units are working as expected, there is confidence in their performance in the full LU. Thus, testing the enabling circuitry in the LU, and guaranteeing the appropriate output is put through, is the primary objective. Keep in mind that only one bit of alu\_sel will be activated at one time due to the 3-to-8 decoder that is housed in the ALU, so need to worry if two bits are selected. The expected output will be represented by M.

#### Figure 2a.10: Logical Unit Waveform

This waveform is summarized in the following table. It primarily tests the enabling circuitry so the correct output is put through, but the output M is also investigated to make sure the logical operation is performed as expected.

#### Table 2a.6: Logical Unit Test Case Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **alu\_sel0 (NOT)** | **alu\_sel1 (OR)** | **alu\_sel4 (XOR)** | **alu\_sel5 (AND)** | **A** | **B** | **M** |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 254 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |

The logical operations are color coded. M expectedly displays the correct output for the given logical operation. Again, since alu\_sel will be fed by a 3-to-8 decoder, it is guaranteed that only one of these select bits will be active at one time. Hence, it has been confirmed that the LU is operating to its specification.

# 2b: Arithmetic Unit (AU)

## Specification

The arithmetic unit is responsible for computing four different operations determined by alu\_sel as can be seen in Table 2b.1. The AU takes in two 8-bit busses A and B, and a single bit C\_In. The output result is in the 8-bit bus M, with single bit outputs M7 for the sign bit, V for overflow, and C\_Out for the carry out.

#### Table 2b.1: ALU Select and Corresponding AU Operations

|  |  |
| --- | --- |
| Alu\_sel | Operation |
| 2 | A -1 + C\_In |
| 3 | A + B + C\_In |
| 6 | A + (~B) + C\_In |
| 7 | A + C\_In |

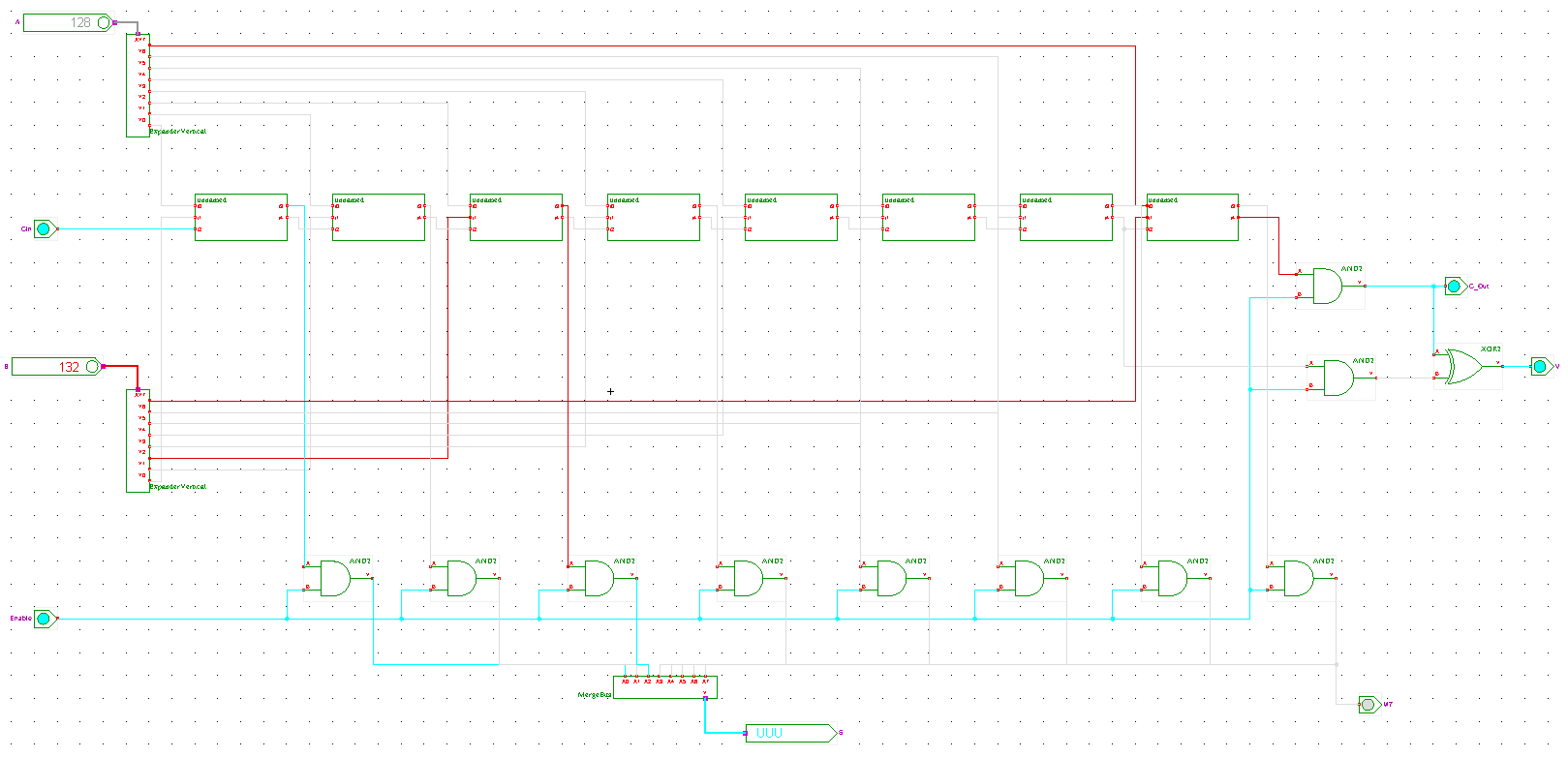
## Design

The core component of the AU is the 8-bit full adder. This was developed in Project1 in Figure 1b.2, and has already been verified. Four of them will be wired for each operation. Certain changes have been made for use in the AU, however. There are now enabling circuits tied to the output ith bits to allow alu\_sel to choose which adder to use appropriately. Also, the C\_out single bit output is placed from the carry out of the last adder, and the overflow bit V is the XOR of the ith and i-1th bit.

The M7 bit is only used to detect the sign bit in a 2’s Complement encoded operation. Since M7 may only be applicable to one operation, alu\_sel 2, an 8-bit full adder with sign bit is developed where the ith bit is connected to M7. The other three operations use full adders without the M7 output since it was not defined in the specification which functions should output the sign bit, though it could be easily implemented by wiring the full adder with the sign bit.

The design and verification of the new full adder will be the one with the sign bit. Since most of the circuitry has already been explained and arithmetic operations have already been verified, these sections will focus on the new additions and their correctness.

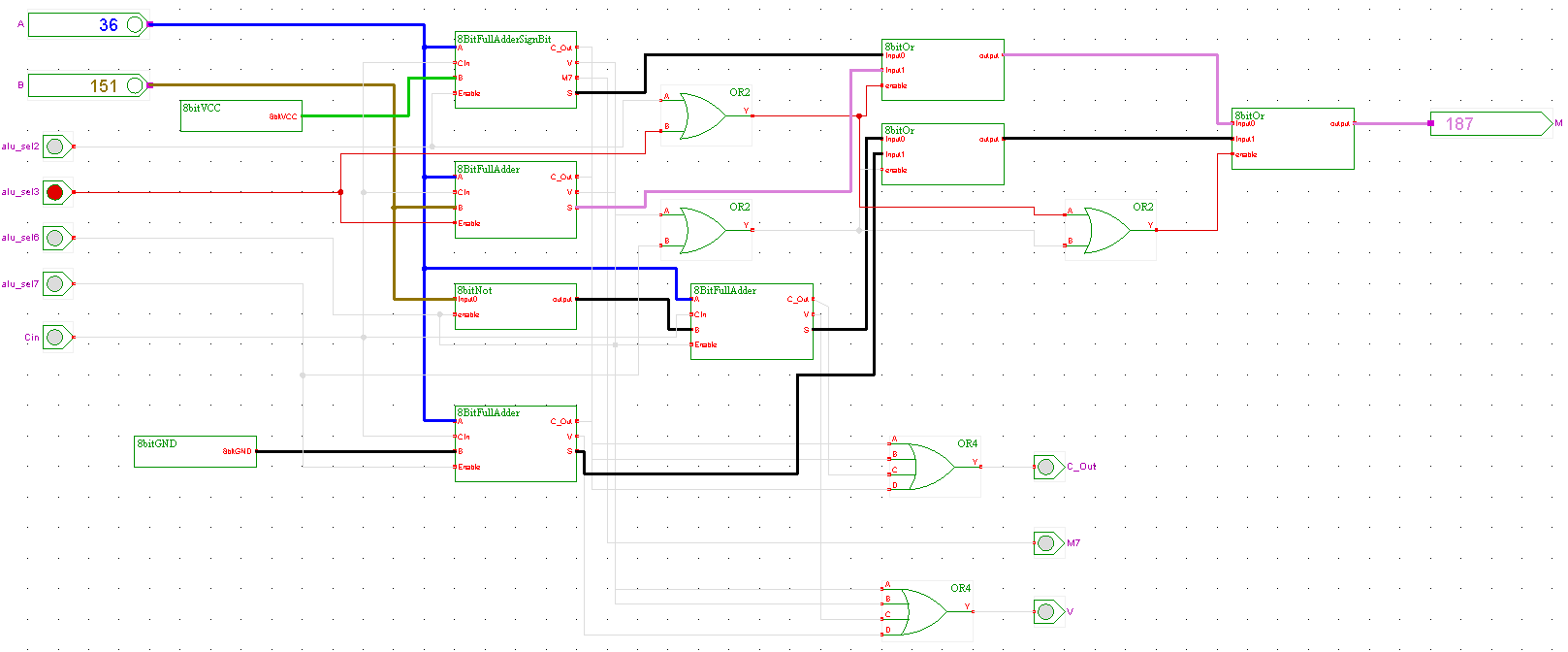
#### Figure 2b.1: Eight Bit Full Adder Sign Bit Circuit Sub Design



It can be seen that the enable is tied to the resulting bits to allow flow when activated. The carry out is the C\_Out of the last adder, and the overflow bit V is XOR’ed with the last and second to last result bit. M7 is wired to the most significant bit result of M, useful as the sign bit in 2’s Complement encoded operations.

The AU uses three 8-bit standard full adders and one full adder with the sign bit described above. It also incorporates two new sub designs, an 8-bit VCC and 8-bit GRD, which represent 8-bit 1 and 8-bit 0. These circuits are used in operations alu\_sel2 and alu\_sel6, and are rather simple. They don’t warrant their own diagrams and testing due to their triviality, though their correctness will be verified in the final testing of the AU. Three 8-bit OR and an 8-bit NOT logic devices are also reused from the LU.

#### Figure 2b.2: Arithmetic Unit Circuit



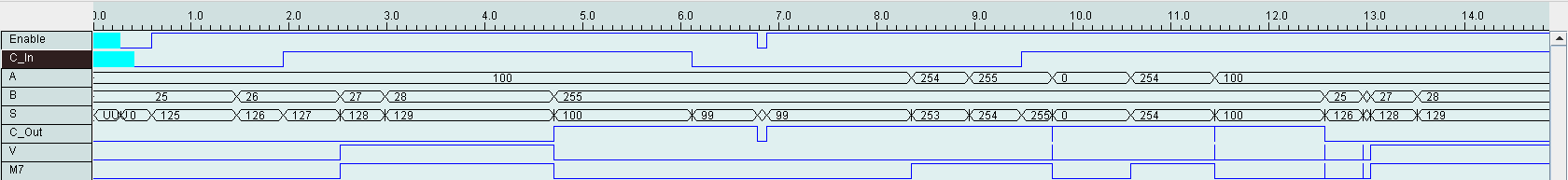
The alu\_sel lines feed into their corresponding adder’s enable input. Similar to the LU, the select lines are put through 3 OR gate to make sure the correct function is performed. Just as before, it is guaranteed that only one alu\_sel will be activated by the the 3-to-8 decoder. Also, three 8-bit OR logic devices are used to choose the correct result for output M. Four input OR gates are employed to generate the C\_Out and V from each adder’s operation, and the M7 bit is only attached to the adder governed by alu\_sel2 as noted earlier. If it was necessary to generate M7 out of each operation as explained earlier, an additional four input OR gate would be used to output M7.

The 8-bit VCC is used in alu\_sel2 because A – 1 in 2’s Complement is really A + 1 in terms of binary addition. The VCC connects to the B input in this function. The 8-bit GRD is used in alu\_sel7 to feed into the B input, since A + C\_In is where simply B is zeroed out. The 8-bit NOT logic device is implemented for alu\_sel6 to complement input B. C\_In is fed into all adders since it is required for all operations.

## Verification

First is the full adder with sign bit testing. Arithmetic operations have already been verified in Project1 in Figures 1b.4-1b.6, though now it is prudent to check the behavior of the single bit outputs of M7, V, and C\_Out. The following tests are done to trigger them appropriately. Function alu\_sel2 is also tested with A-1 translated to A+255.

#### Figure 2b.3: Eight Bit Full Adder Sign Bit Waveform



This waveform is summarized in the following table where it is clear that the arithmetic operations are being performed as expected and single bit outputs signal when appropriate.

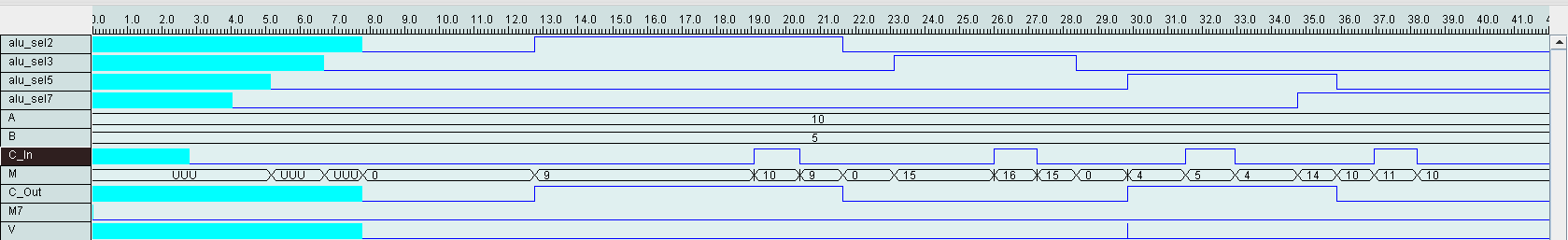
#### Table 2b.2: Eight Bit Full Adder Sign Bit Test Case Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Enable | C\_In | A | B | M | M7 | C\_Out | V |
| 0 | 0 | 100 | 25 | 0 | 0 | 0 | 0 |
| 1 | 0 | 100 | 25 | 125 | 1 | 0 | 0 |
| 1 | 0 | 100 | 26 | 126 | 1 | 0 | 0 |
| 1 | 1 | 100 | 26 | 127 | 1 | 0 | 0 |
| 1 | 1 | 100 | 27 | 128 | 1 | 0 | 1 |
| 1 | 1 | 100 | 28 | 129 | 1 | 0 | 1 |
| 1 | 1 | 100 | 255 | 100 | 0 | 1 | 0 |
| 1 | 0 | 100 | 255 | 99 | 0 | 1 | 0 |
| 0 | 0 | 100 | 255 | 0 | 0 | 0 | 0 |
| 1 | 0 | 100 | 255 | 99 | 0 | 1 | 0 |
| 1 | 0 | 254 | 255 | 253 | 1 | 0 | 0 |
| 1 | 0 | 255 | 255 | 254 | 1 | 0 | 0 |
| 1 | 1 | 255 | 255 | 255 | 1 | 1 | 0 |

It can be seen in the waveform and table that the circuit is operating as expected. M7, C\_Out, and V trigger when necessary. M7 is 1 when output bus M has a 1 in its most significant bit. C\_Out is 1 when the last adder has a carry bit, typically seen when doing functions alu\_sel2 and alu\_sel6. V is 1 when there is an overflow, when M exceeded 127. The subtraction works when B is set to 255, C\_In works as expected, and the enable lets output flow appropriately. Thus, it is confirmed that the adder is operating to the specification, especially with the test cases performed in Project1.

The testing of the AU is mainly to verify that the correct operation is performed when alu\_sel dictates it to. The M7, C\_Out, and V outputs have already been verified in the Figure 2b.3 and Table 2b.2.

#### Figure 2b.4: Arithmetic Unit Waveform



This waveform is summarized in the following table. It is clear that the alu\_sel lines govern the correct operations, and the outputs meet expectations.

#### Table 2b.3: Arithmetic Unit Test Case Results

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Alu\_sel2** | **Alu\_sel3** | **Alu\_sel6** | **Alu\_sel7** | **A** | **B** | **C\_In** | **M** | **M7** | **C\_Out** | **V** |
| 0 | 0 | 0 | 0 | 10 | 5 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 10 | 5 | 0 | 9 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 10 | 5 | 1 | 10 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 10 | 5 | 0 | 9 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 10 | 5 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 10 | 5 | 0 | 15 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 10 | 5 | 1 | 16 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 10 | 5 | 0 | 15 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 10 | 5 | 0 | 4 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 10 | 5 | 1 | 5 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 10 | 5 | 0 | 4 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 10 | 5 | 0 | 10 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 10 | 5 | 1 | 11 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 10 | 5 | 0 | 10 | 0 | 0 | 0 |

The table is color coded by operation. It is clear that all tan operations produce A-1+C\_In, blue produces A+B+C\_In, red produces A+(~B)+C\_In, and green A+C\_In. All operations have performed in conjunction to their select line and produce the expected output. Remember, the alu\_sel is guaranteed to only have one bit activated at once, so testing of the functions when their alu\_sel is 1 and the others are 0 is sufficient. The select line testing was the primary goal, as the operations were confirmed beforehand, so the AU has now been verified.

# 2c: Arithmetic Logic Unit (ALU)

## Specification

The general specification has already been outlined in the introduction, though this is a good place for the full alu\_sel table. It describes each operation the ALU must perform on the inputs and deliver the results to the outputs. It is color coded for the verification table that is presented later in this section.

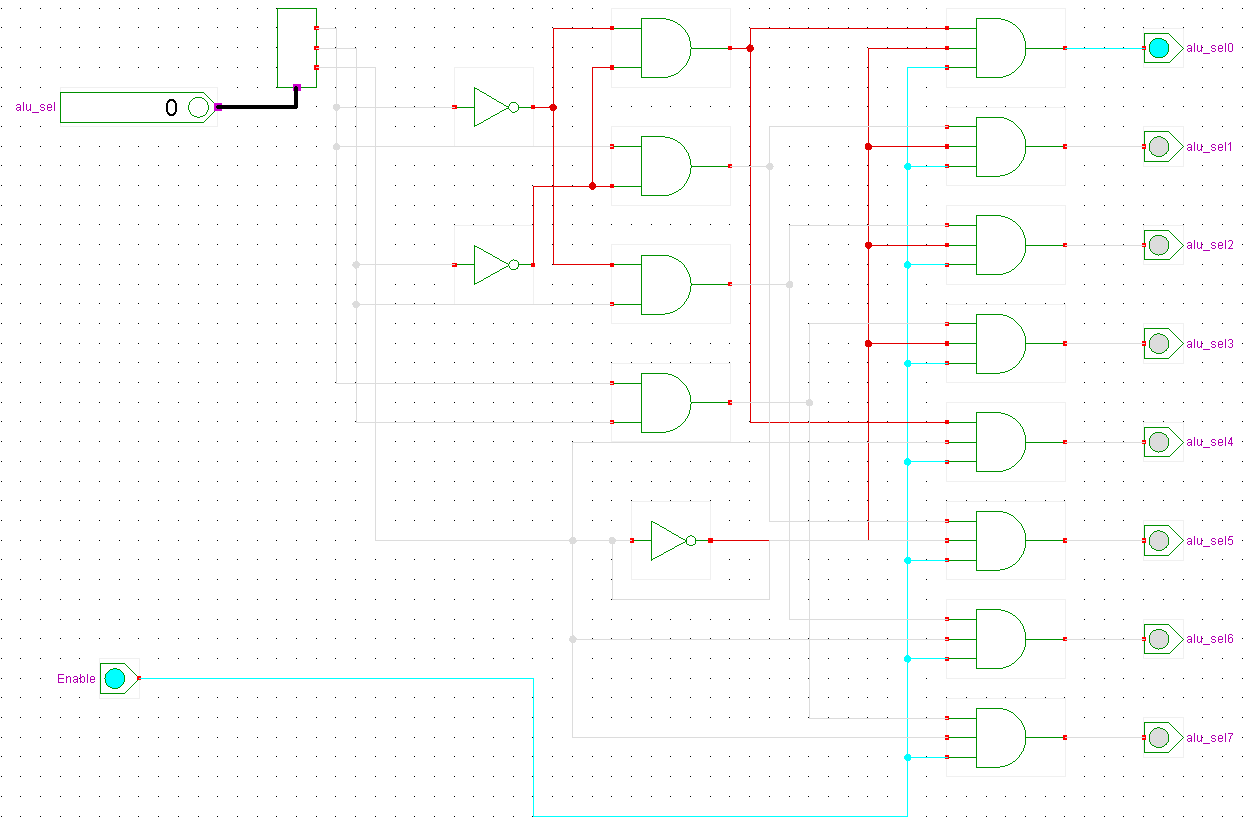
#### Table 2c.1: ALU Select and All Corresponding Operations

|  |  |
| --- | --- |
| **Alu\_sel** | **Operation** |
| 0 | ~A |
| 1 | A OR B |
| 2 | A – 1 + C\_In |
| 3 | A + B + C\_In |
| 4 | A XOR B |
| 5 | A AND B |
| 6 | A + ~B + C\_In |
| 7 | A + C\_In |

## Design

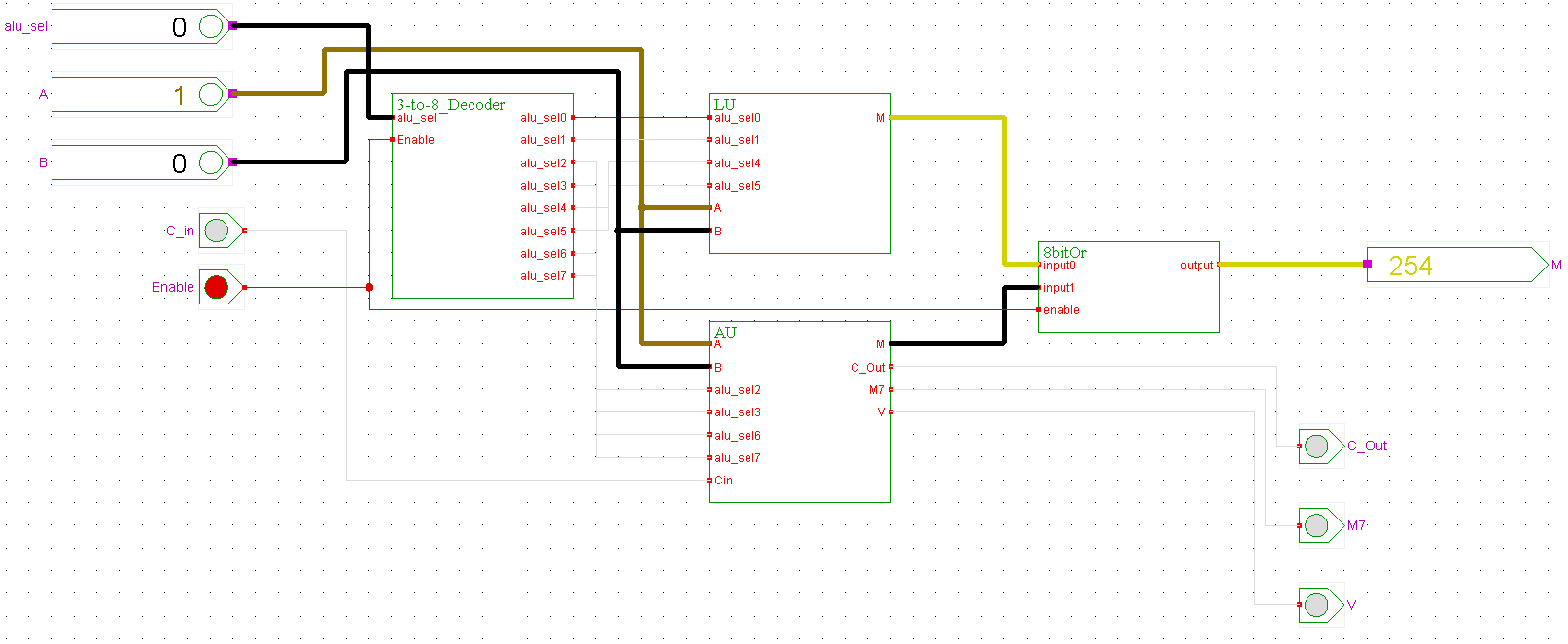
Now that the LU and AU have been developed, the final ALU can be constructed. A 3-to-8 decoder is constructed to take 3-bit alu\_sel and feed the eight enables of the AU and LU. Inputs A, B and C\_In are also fed into the sub components. The 8-bit OR logic device chooses the 8-bit output for bus M, and the output pins M7, C\_Out, and V are wired to their respective outputs of the AU.

#### Figure 2c.1: Three-to-Eight Decoder Circuit Sub Design



The 3-to-8 decoder expands the 3-bit input and produces each of the input’s minterms. These minterms translate to alu\_sel0-7 and control each operation. It is guaranteed that only one bit of output will be 1 at a given time. There is an enable for the decoder that will be wired to the enable of the ALU. The enabling circuitry use three input AND gates to allow output.

#### Figure 2c.2: Arithmetic Logic Unit Circuit

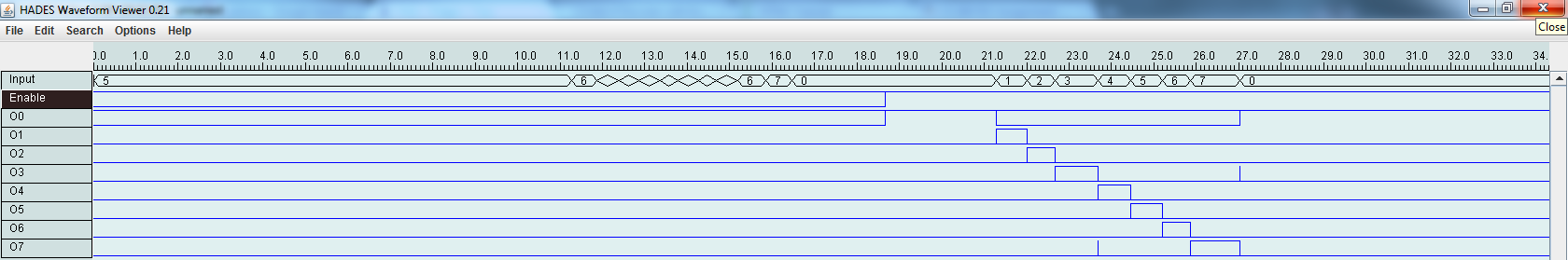


The ALU incorporates the 3-to-8 decoder, LU, AU, and the 8-bit OR device. All inputs and outputs are clearly labeled and are quite straight forward at this point. The enable of the ALU feeds to the 3-to-8 decoder. Then alu\_sel is expanded to produce each of its bits, which enable the operations of the AU and LU. The 8-bit OR device chooses which output of the AU and LU to put out, and the other output pins are wired to their respective locations of the AU.

## Verification

First is the 3-to-8 decoder investigation. Testing to make sure that only one output is given at a time is key for enabling the operations of the LU and AU.

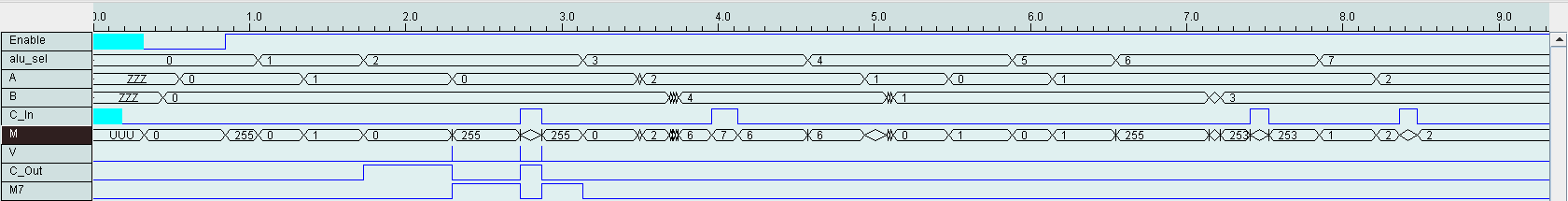
#### Figure 2c.3: Three-to-Eight Decoder Waveform



It is clear from the waveform that each output bit is 1 corresponding to the 3-bit input. When one output bit is 1, every other output is 0. The enable works. The waveform is straight forward so a table is not necessary. The testing is exhaustive, and there is full confidence in the performance of the decoder.

Testing the final ALU is rather simple. Most of the verification is done in the subsequent stages of development. At this stage, it is known that the decoder, LU, and AU and their subcomponents function per specification. Thus, testing the ALU is reduced to seeing the correct operation performed and output given. The output bus will M will mostly confirm that the expected functions are being done. The output pins M7, C\_Out, and V have already been tested, so though they may trigger in the following cases, their investigation is better surmised by Figure 2b.2.

#### Figure 2c.4: Arithmetic Logic Unit Waveform



The waveform confirms that the expected operations are performed. The results are summarized in the following table for easier viewing. It is also worth mentioning again that sections of the waveform where the inputs are ZZZ or XXX or <\*><\*> is just the changing of values and don’t reflect real test cases, even if the output busses respond. All test cases are represented in the table.

#### Table 2c.2: Arithmetic Logic Unit Test Case Results

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Enable** | **Alu\_sel** | **A** | **B** | **C\_In** | **M** | **V** | **C\_Out** | **M7** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 255 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 2 | 0 | 0 | 0 | 255 | 0 | 0 | 1 |
| 1 | 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 2 | 0 | 0 | 0 | 255 | 0 | 0 | 1 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 2 | 0 | 0 | 2 | 0 | 0 | 0 |
| 1 | 3 | 2 | 4 | 0 | 6 | 0 | 0 | 0 |
| 1 | 3 | 2 | 4 | 1 | 7 | 0 | 0 | 0 |
| 1 | 3 | 2 | 4 | 0 | 6 | 0 | 0 | 0 |
| 1 | 4 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 5 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 6 | 1 | 1 | 0 | 255 | 0 | 0 | 0 |
| 1 | 6 | 1 | 3 | 0 | 253 | 0 | 0 | 0 |
| 1 | 6 | 1 | 3 | 1 | 254 | 0 | 0 | 0 |
| 1 | 6 | 1 | 3 | 0 | 253 | 0 | 0 | 0 |
| 1 | 7 | 1 | 3 | 0 | 1 | 0 | 0 | 0 |
| 1 | 7 | 2 | 3 | 0 | 2 | 0 | 0 | 0 |
| 1 | 7 | 2 | 3 | 1 | 3 | 0 | 0 | 0 |
| 1 | 7 | 2 | 3 | 0 | 2 | 0 | 0 | 0 |

The operations are color coded. It is clear that the correct operations are being executed according to alu\_sel and M reflects the correct result. The enable is tested first and works. Each operation is done in order of Table 2c.1. One may follow columns A, B, and C\_In and compute the operation to find it in column M. The output bits also flare when necessary.

Thus, the ALU has been assembled and verified. There is full confidence in the device as each sub component was tested before used in a larger hierarchical design. There was significant reuse of developed components throughout the project. Although this may have increase gate-input cost, there are savings in time and labor costs. Optimization is expensive, and since this ALU won’t be used in critical systems in the near future, possible propagation delay is a practical trade off.

Project 3

Project 3 primarily deals with data transfer and storage. It is made up of three large components, the Instruction Register (IR), Register File (RF) and Memory. This project will continue to follow a bottom up approach by discussing the subdesigns and how they form the larger digital system.

# 3a: Instruction Register (IR)

## Specification

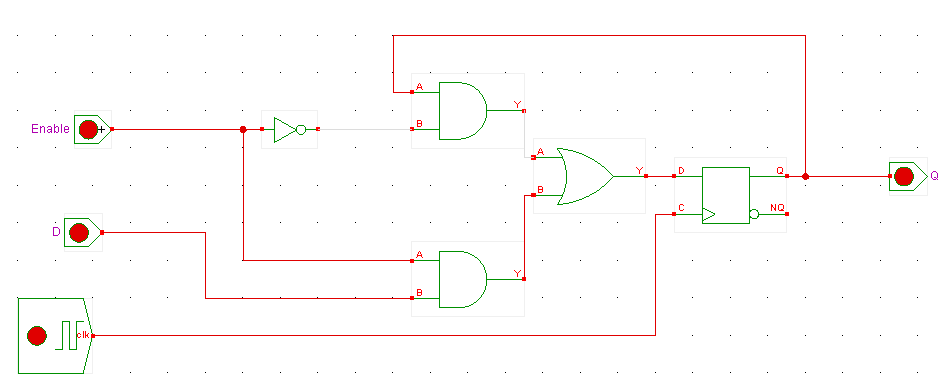
The IR is a 16-bit register that is loaded from an 8-bit bus. The input can be placed on the low or high byte of the output dependent on select lines IR0 and IR2. There is also an output enable that places the 16-bit value on the 16-bit bus.

## Design

The IR is composed of two 8-bit registers. The 8-bit register is composed of D Flip-Flops with enable. Eight D Flip-Flops are used since each gives one bit of storage.

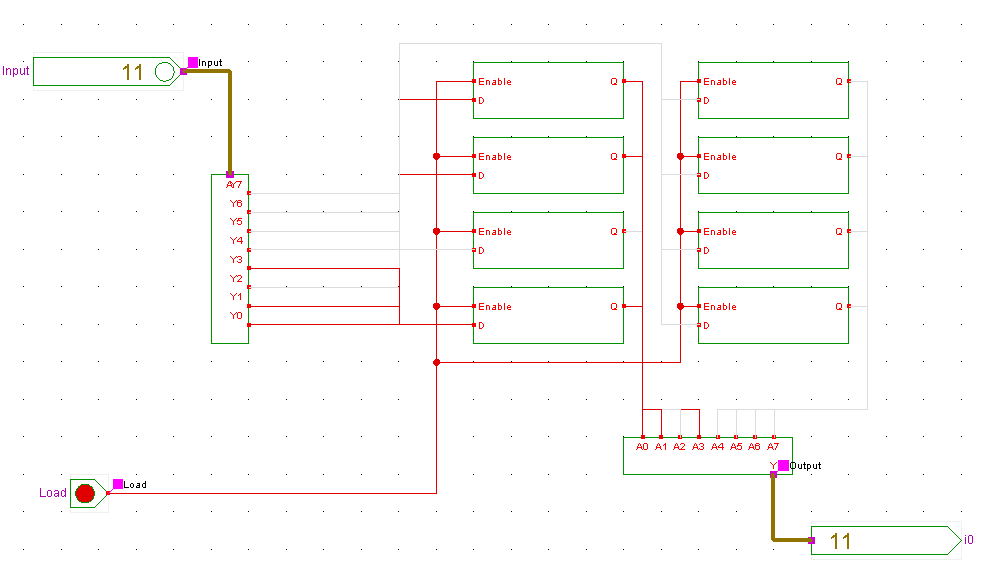
It proved easier to include clocks at this stage and also avoided timing issues, but at a slight extra cost in hardware. An enable is included for the future Load/Hold ability of the register. The circuitry stops input to the D Flip-Flop with the enable, and Q is OR’ed with D for the next state.

#### Figure 3a.1: D Flip-Flop with Enable Circuit



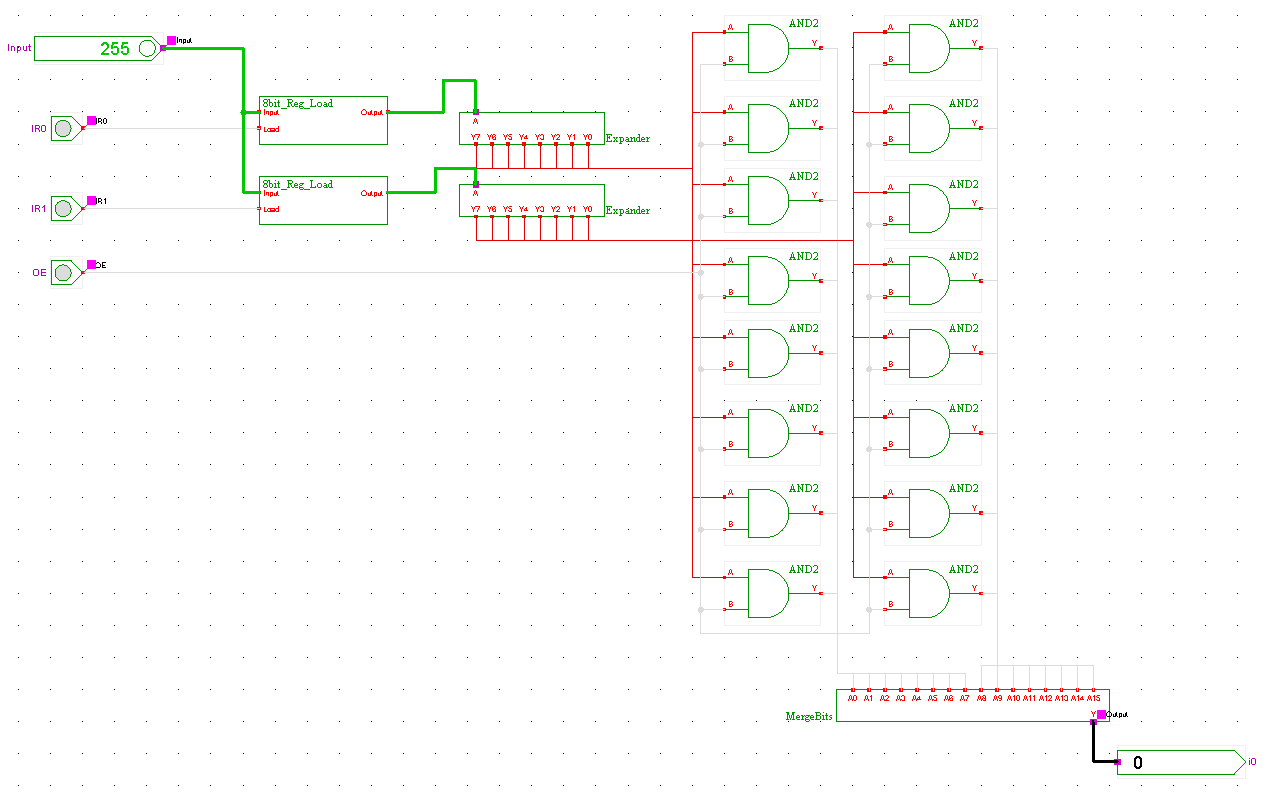
For the full register, eight flip flops with enable are used. The 8-bit input bus is expanded and fed into D, as the single bit Load pin is fed into each enable for writing and rewriting. Each Q output is merged for the 8-bit output bus.

#### Figure 3a.2: Eight Bit Register with Load Circuit



This register is then duplicated twice to form the full 16-bit IR. The IR select lines feed into the Load of the two registers, and the 8-bit input is fed into both registers. The bits are expanded and AND’ed with the OE line for the enabling circuitry, and feed into their corresponding places in the merger for the 16-bit output.

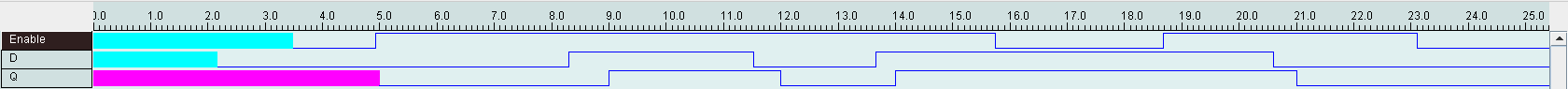
#### Figure 3a.3: Sixteen Bit Instruction Register Circuit



## Verification

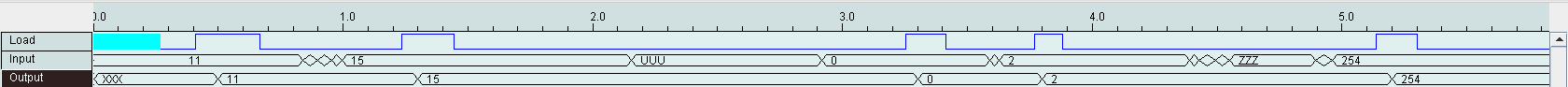
The most fundamental unit is tested first. The D Flip-Flop with enable waveform is exhaustive and it’s easily seen that it adheres to the expected behavior. The purple is the undefined state when nothing was stored.

#### Figure 3a.4: D Flip-Flop Waveform



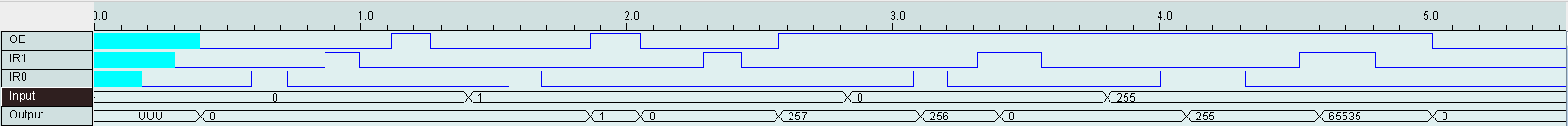
The register is tested next. Since the single D Flip-Flop works as expected, this waveform is not exhaustive and does simple test cases to ensure the write and rewrite functions work in the Load line. It’s easy to see that several different values are Loaded and Held in the register as expected with the Load line.

#### Figure 3a.4: Eight Bit Register Waveform



The full IR is tested next. Since both sub components operate to the specification, we’re primarily checking the functionality of the select and enable lines.

#### Figure 3a.5: Sixteen Bit Instruction Register Waveform



It is clear that IR1 and IR0 output their respective values on the high and low byte when activated. The OE line is tested and works by outputting all 16 bits when 1. The registers are also tested by writing and rewriting between the low and high end of the spectrum. Thus, it has been verified that the IR is functioning to specification.

# 3b: Register File (RF)

## Specification

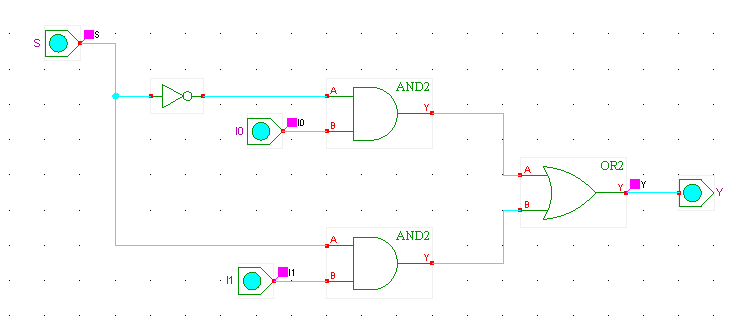
The RF is composed of eight of the same registers used in 3a. RES is the 8-bit input bus, and R0\_W through R7\_W are the Loading, or writing and rewriting, signals for the registers. There are two 8-bit outputs, A and B, that output a given register’s contents according to two 8-to-1 multiplexers and their A\_Sel and B\_Sel lines.

## Design

The register has already been developed and verified in 3a. The 8-bit 8-to-1 multiplexer is made up of two 8-bit 4-to-1 multiplexers and one 8-bit 2-to-1 multiplexer. The 8-bit 4-to-1 mux was developed and verified in Project 1, so the 8-bit 2-to-1 mux is developed here.

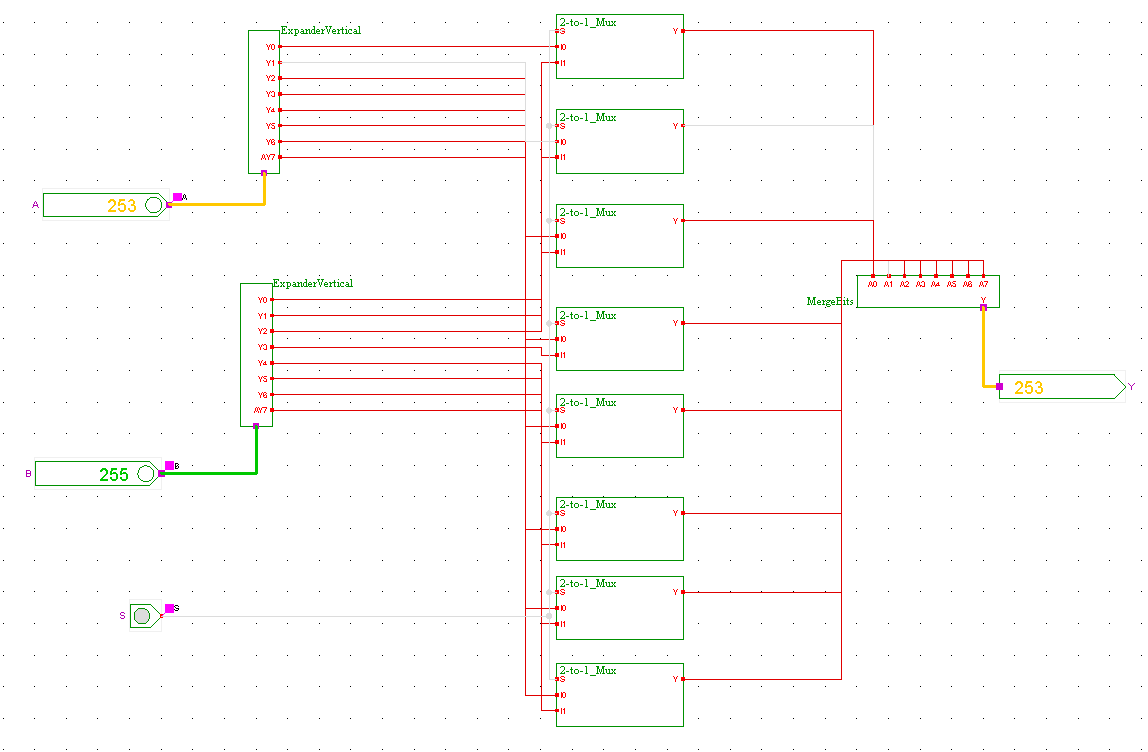
The single bit 2-to-1 mux is a simple design featuring a couple AND gates, an OR gate, and an inverter. The single bit select chooses between the two inputs I0 and I1.

#### Figure 3b.1: Single Bit Two-to-One Multiplexer Circuit



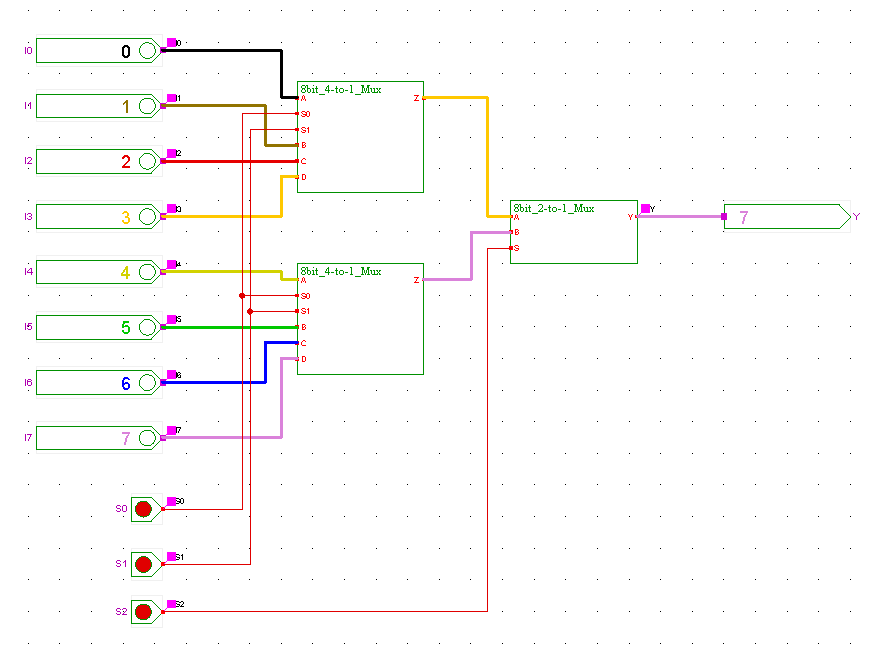
This subdesign is replicated eight times to form the 8-bit 2-to-1 mux. The input busses are expanded and fed to each component, and single bit select line is also wired to each subdesign. The bits are merged for the 8-bit output.

#### Figure 3b.2: Eight Bit Two-to-One Multiplexer Circuit



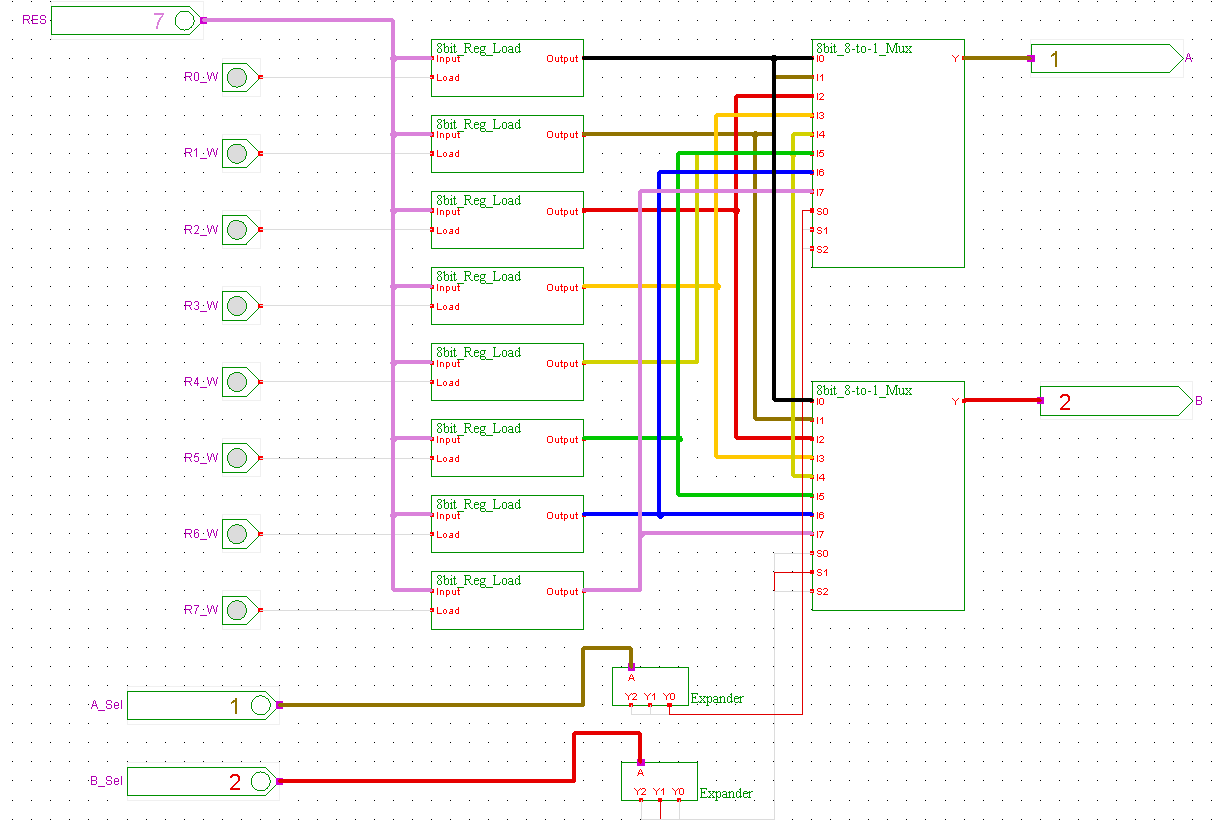
Now that the 8-bit 2-to-1 mux is developed, the 8-bit 8-to-1 mux can be constructed. Using two 8-bit 4-to-1 muxes and an 8-bit 2-to-1 mux, one input of eight can be selected. The busses, and S0 and S1, are fed into the 4-to-1 muxes. The result of the 4-to-1 muxes feed into the 2-to-1 mux with the S2 select line. All bit expansions are taken care of in the subcomponents.

#### Figure 3b.3: Eight Bit Eight-to-One Multiplexer Circuit



Now that the multiplexer has been developed, the full RF can be built. Eight registers are placed with R0\_W through R7\_W connected to their Load signals, and the RES input connected to each input register bus. The register outputs are sent to two 8-to-1 multiplexers. A\_Sel and B\_Sel are expanded and fed into the select lines of the mux, they choose which register to output and on which output bus.

#### Figure 3b.4: Register File Circuit



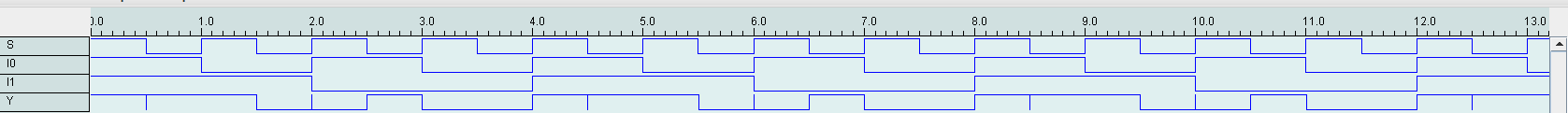
## Verification

The single bit 2-to-1 mux is tested first. It is exhaustive since it was generated using clocks, and adheres to the full truth table describing its behavior. The following is the condensed truth table for ease of reading.

#### Table 3b.1: Single Bit 2-to-1 Mux Truth Table

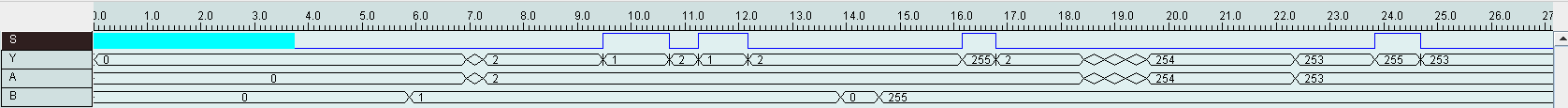
|  |  |
| --- | --- |
| **S** | **Y** |
| 0 | **I0** |
| 1 | **I1** |

#### Figure 3b.5: Single Bit 2-to-1 Mux Waveform



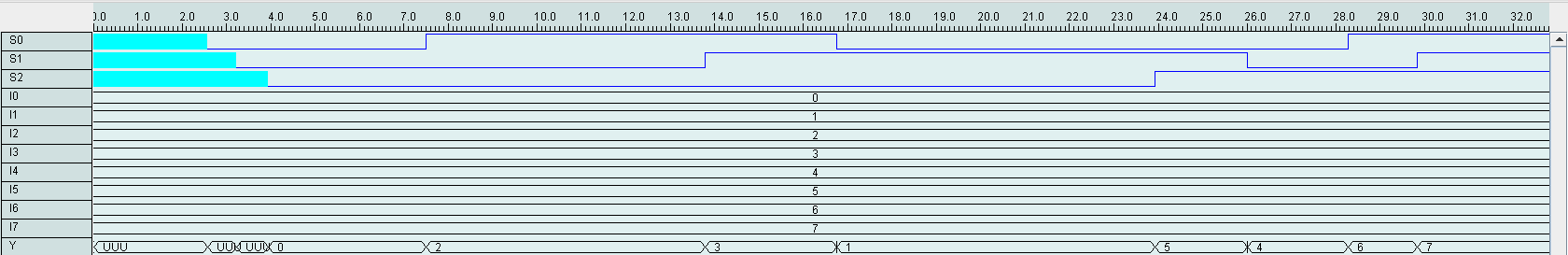
This gives confidence to the 8-bit 2-to-1 mux since all values are accounted for. The 8-bit 2-to-1 mux then simply tests that the values are switching appropriately.

#### Figure 3b.6: Eight Bit 2-to-1 Mux Waveform



It can be easily seen that the values on A and B are switching in conjunction with the select line. Thus, this smaller multiplexer has been verified.

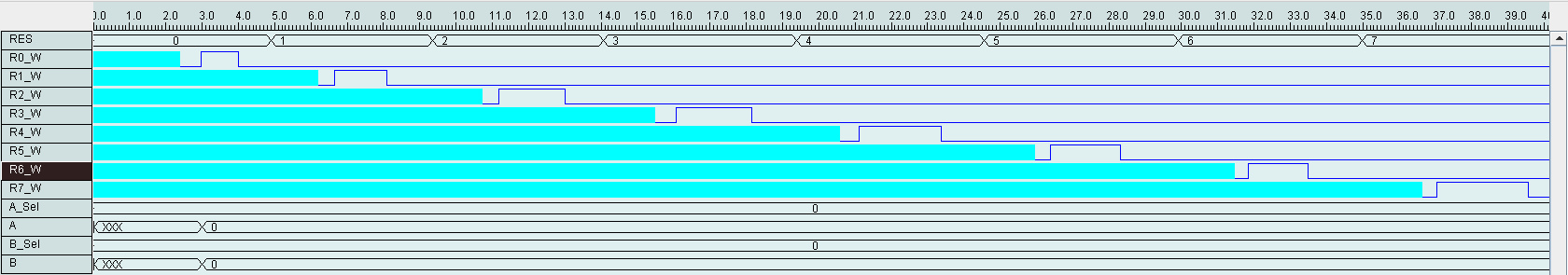
#### Figure 3b.7: Eight Bit 8-to-1 Mux Waveform



Since all subcomponents have been confirmed, testing the 8-to-1 mux is simply checking that each input bus is placed on the output according to the select lines. This waveform confirms that each value is outputted when the select lines dictate them to. The undefined stage in the beginning of the waveform for Y is simply when the select lines were undefined.

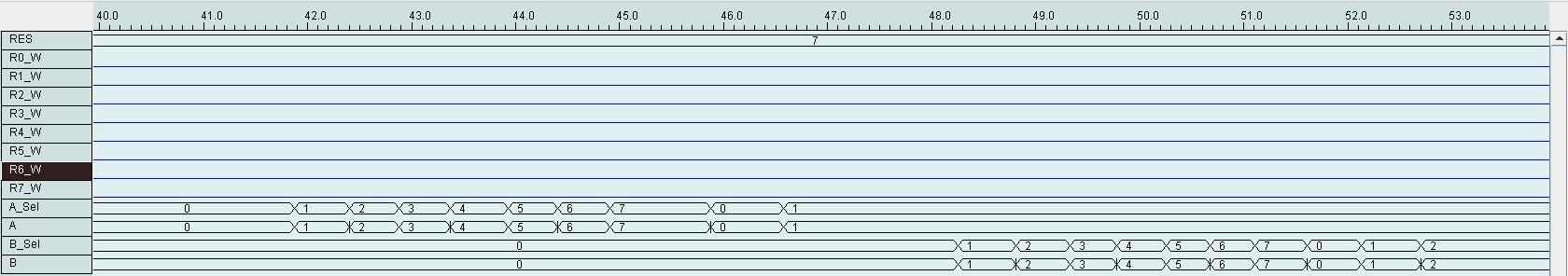
The 8-bit 8-to-1 mux has been verified, as well as all other components, so it’s time for the final testing of the RF, which is done in three waveforms. First values are loaded into the registers, then are tested to be outputted based on the select lines, and finally rewritten to show that the registers and load and hold.

#### Figure 3b.8: RF Waveform 1



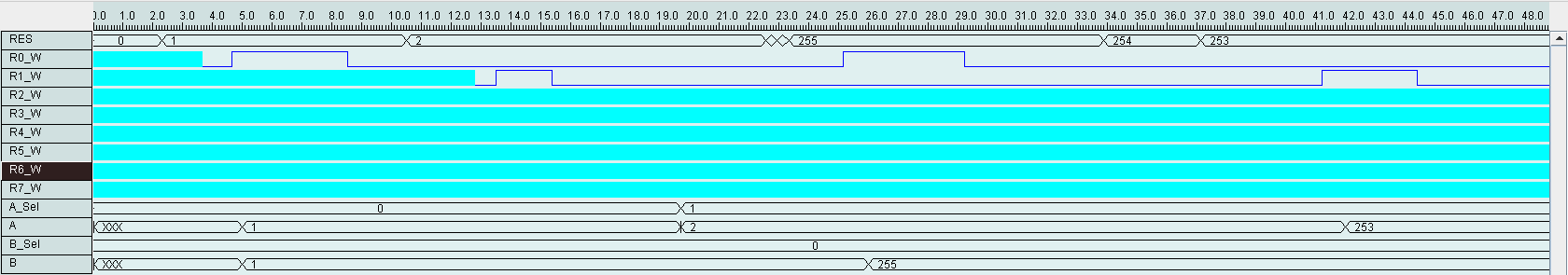
This waveform simply shows the values 0-7 are loaded into registers 0-7. This helps in seeing the correct output as when A\_Sel is 0, 0 will show on the output bus. This can be seen in this waveform.

#### Figure 3b.9: RF Waveform 2



The output busses A and B accurately display the register’s content governed by their respective select lines as it cycles from 0-7.

#### Figure 3b.10: RF Waveform 3



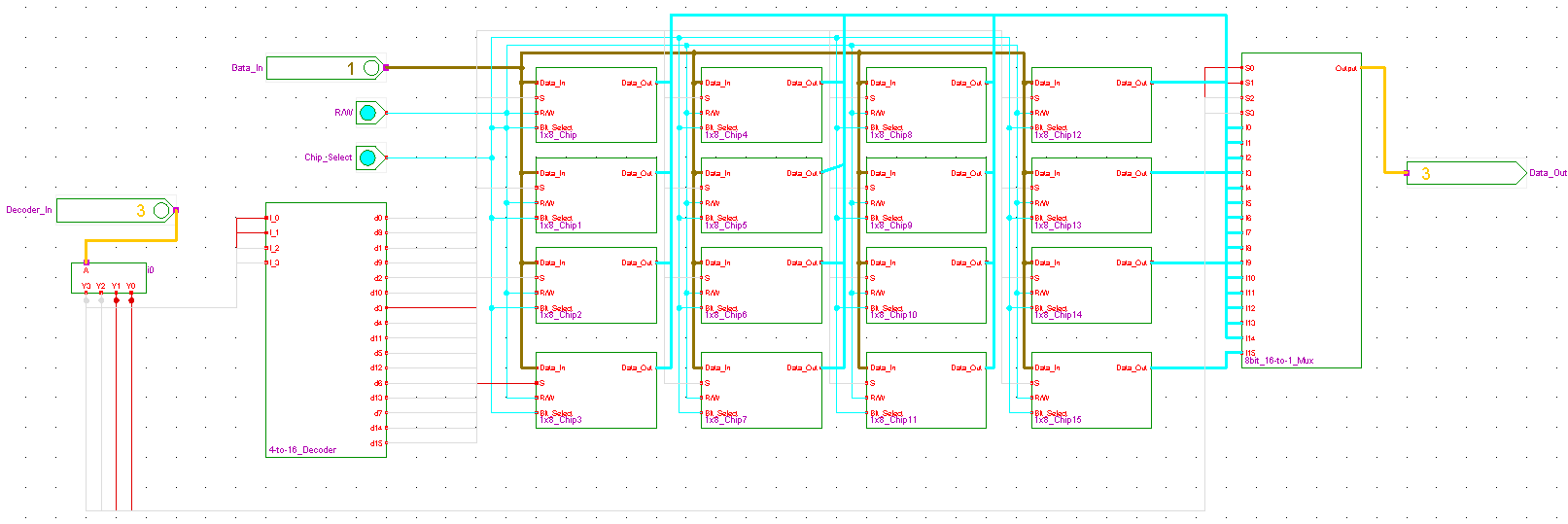
Now that it is known the registers output correctly, this waveform simply demonstrates that values can be reloaded. It is easily seen that values on the A and B bus change as RES values are loaded into different registers and outputted using the select lines. Not all registers needed to be tested for this, so many of the write inputs were left undefined.

Project 4

This section of the project assembles all pre-existing components and utilizes a control unit to produce the expected operations. It should be stated here that the final stage of the project was not implemented to complete fruition, though significant steps were taken to accomplish this task. It should also be noted that this section of the project was finished by one person. Unfortunately, the other person has the document containing the write-up for 3c – the memory – but circuit pictures and descriptions will be provided to ease the reader into Project 4.

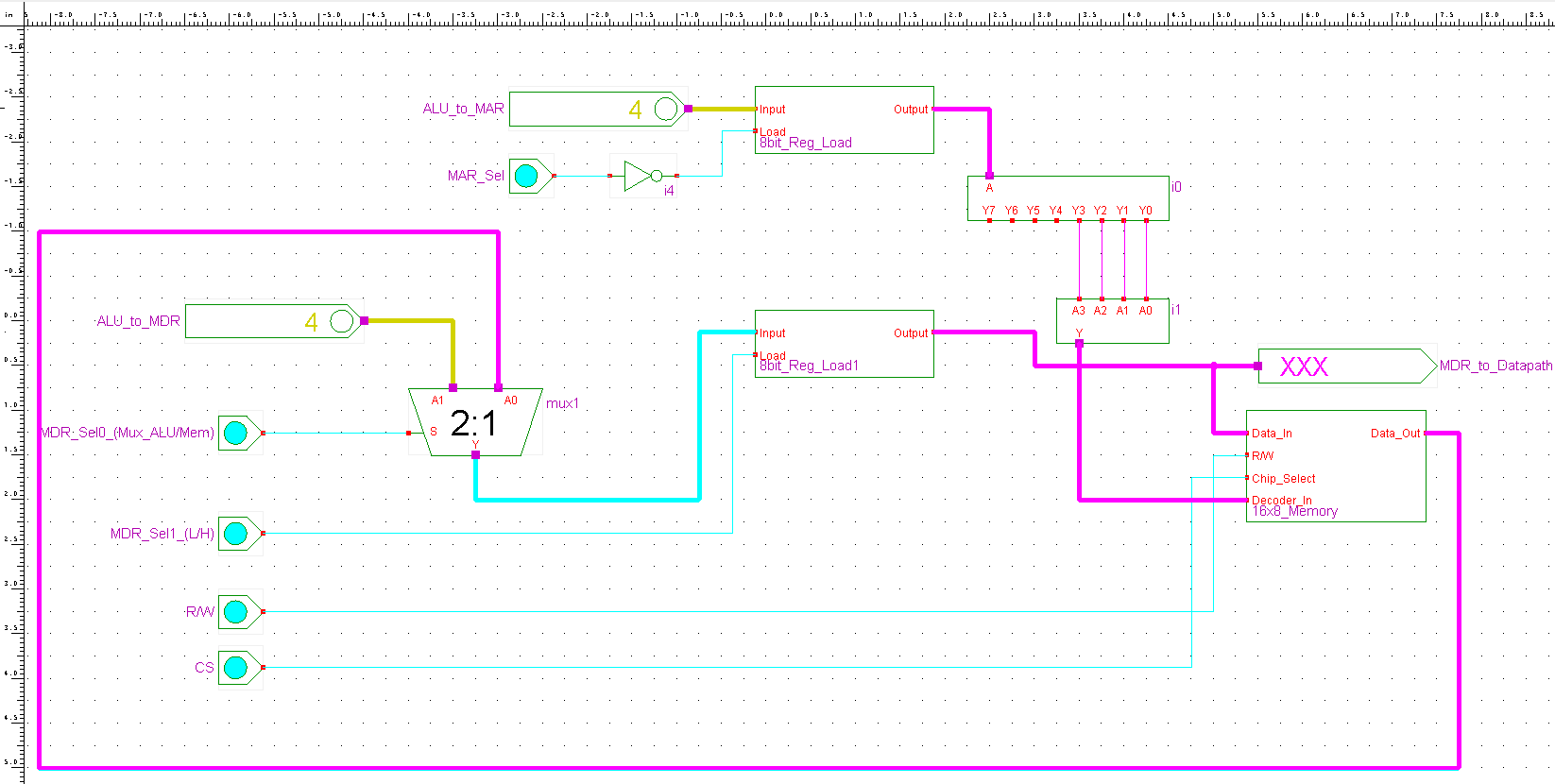
In Project 3, the task was to construct a memory unit suitable for the CPU designed in this course. This task was not completed adequately in the subsequent section, so a brief discussion is offered here.

The MDR and MAR are featured in this full memory circuit. The MDR is the data bus and the MAR is the address bus, both for the memory. The 16x8 memory, formed by RAM slices, can be seen here:



It is worth noting that this memory is constructed using RAM slices instead of a remix of the Register File found in part 3.b. This means that the memory is controlled by a Read/Write toggles and a Chip Select toggle. These inputs are featured in the full CPU to follow.

Next is the full memory with the MDR and MAR attached:

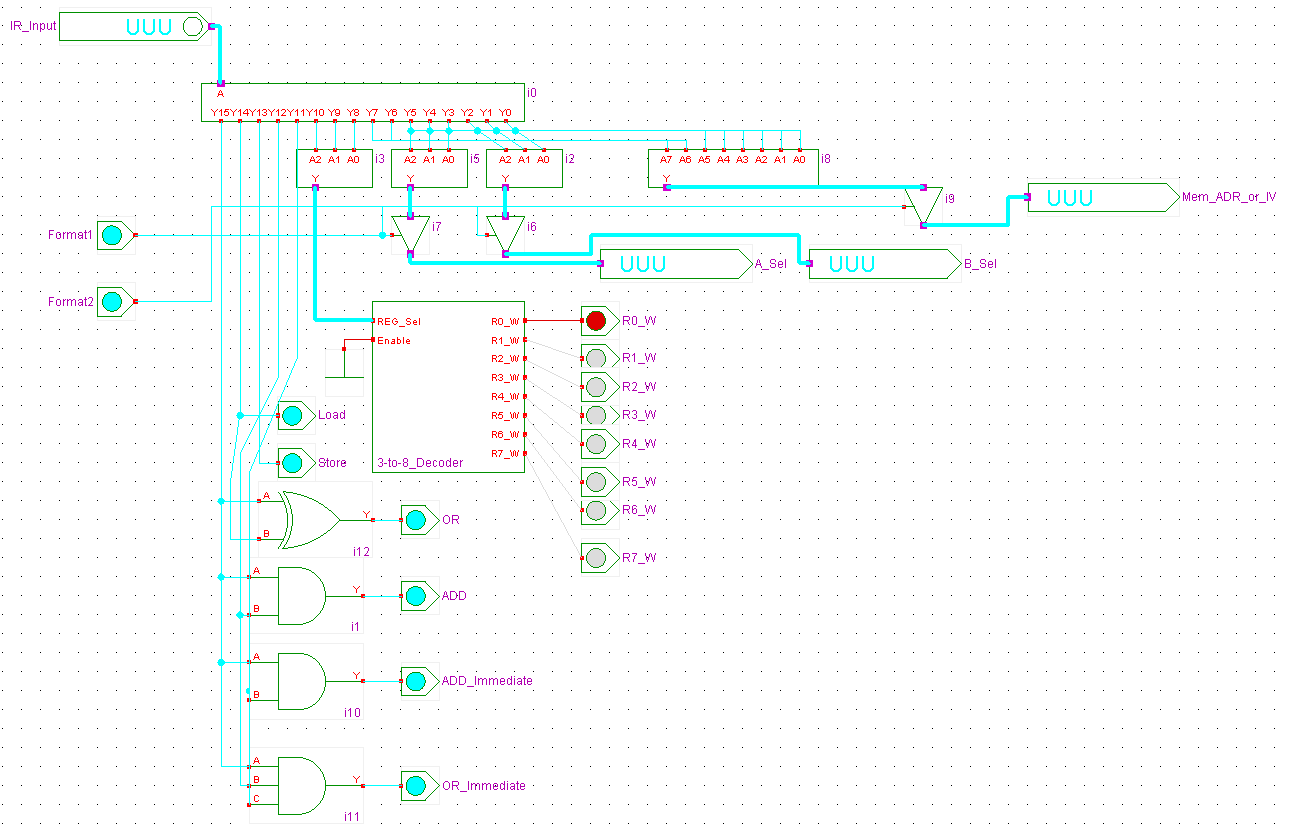


The MAR least significant bits are used as the decoder in of the memory. The memory’s 1x8 chip select is controlled by these bits, which means that whichever word to be investigated can be controlled by these bits. The memory single bit toggles of R/W and CS are featured as well as the MAR and MDR select lines.

The MAR select operated with an inverted by its specification to load and hold when necessary.

A 2-to-1 MUX is implemented to distinguish between the data from the memory and from the data path. This signal is controlled by the second bit of MDR select.

This is the control unit developed to deal with opcodes and to control all selecting circuits:

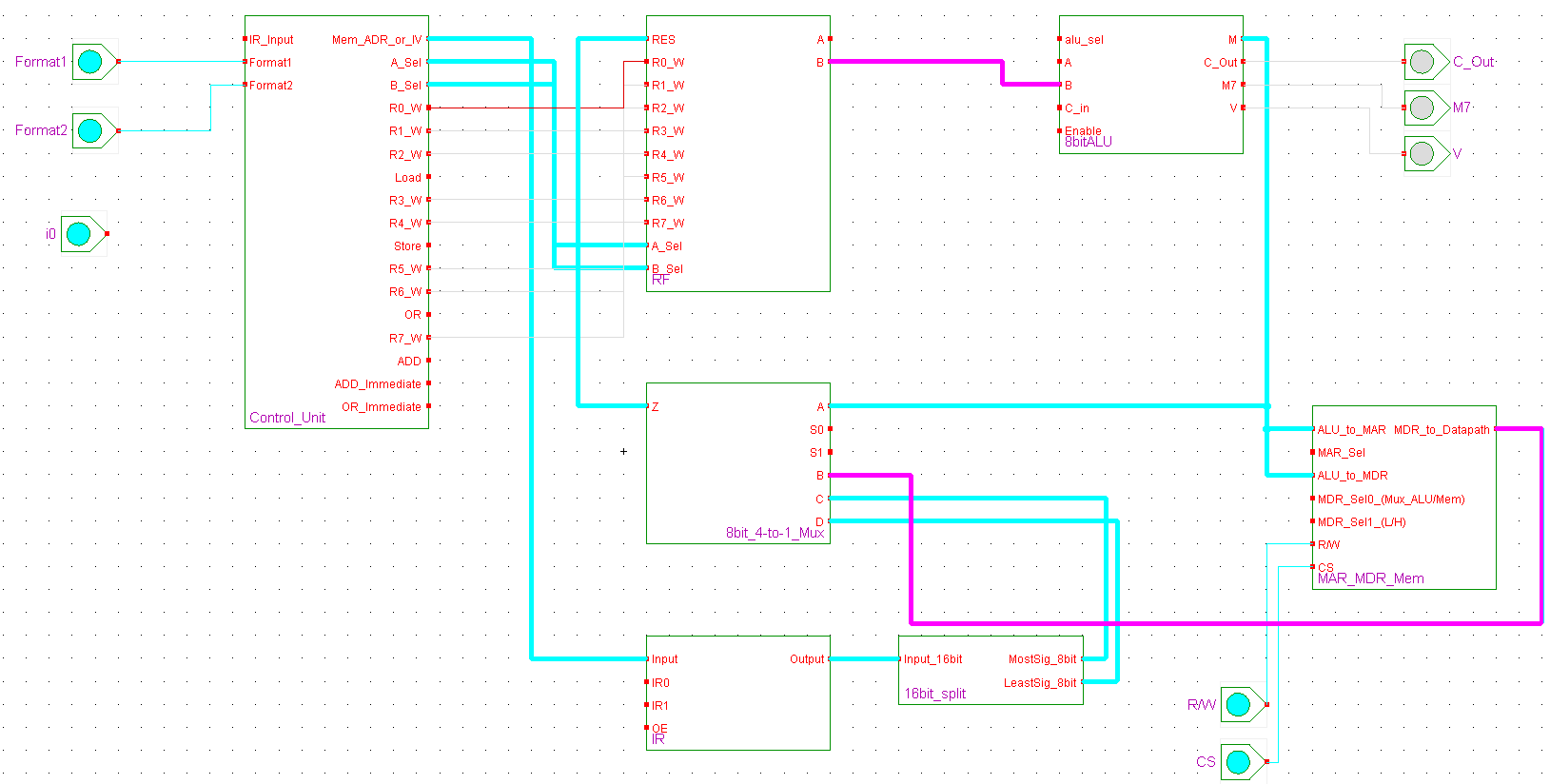


Here, the instruction register input is dissected into a control word for the CPU. There are two formats, one where bits 7-0 control the operand and the other that controls the memory address or immediate value. These formats can be toggled between two control switches.

This is accomplished by expanding the bits of the instruction and performing combinational logic to isolate the operations to be executed. A 3-to-8 decoder is employed, taken from Project 2 with the ALU select, to determine the register writes from the 3-bit REG select.

The opcodes are distinguished by individual bits of the 5. The four different functions, OR, ADD, ADD Immediate, and OR Immediate, can be determined by an XOR, 2 input ANDs, and a 3 input AND respectively. Each operation is given one bit which can be used to toggle the corresponding function in the ALU.

This is the circuit of the entire CPU:



This figure incorporates all components developed. The general outline of the project is surmised in this one circuit, however, with discussion of its shortcomings.

One concern is the PC and the memory. Since the memory developed is not exactly traditional with the RW and CS select lines found in Chapter 8, the control unit needs additional bits to absolutely control these functions. Perhaps instead of trying to use REG7 as the PC, an extra register should be developed in this kind of scenario. This way, operations would handle in the way Figure 9-15 describes in the textbook. However, everything else is wired as it should.

Given more time I would have tested both formats of the opcode, as controlled by the Format1 and Format2 signals found in the control unit. By toggling these signals, the user may choose between two different instructions. Using REG Select lines to choose the register, the opcode to choose the function, and the other bits to choose the memory address/immediate value or operands, all functions could have been tested.

Again, it is known that the each individual component operates to its specification. The task of Project4 is to bring it all together within a datapath. The CPU circuit is close to matching the project picture and operates as expected thus far.

## References

Adder (electronics). (n.d.). In *Wikipedia*. Retrieved February 7, 2011, from http://en.wikipedia.org/wiki/Adder\_(electronics)

Arithmetic logic unit. (n.d.). In *Wikipedia*. Retrieved March 1, 2011, from http://en.wikipedia.org/wiki/Arithmetic\_logic\_unit

Mano, Morris M., & Kime, R. Charles (2008). *Logic and Computer Design Fundamentals*. Upper Saddle River, NJ: Pearson Education.

Multiplexer. (n.d.). In *Wikipedia*. Retrieved January 18, 2011, from http://en.wikipedia.org/wiki/Multiplexer