

There are a total of three tasks for Lab 11. Within this report, I will discuss each of those tasks.

Design of Task (a).

In task (a), I built and packaged a 4-bit tri-state, (ts) buffer with an active high enable control. After creating the buffer, I then used three of the buffers to experiment with a 4-bit bus. **Figure 1** shows what my *prepackaged* 4-bit ts buffer looks like. Figure 2 shows what my *packaged* 4-bit ts buffer looks like as well as my implementation of three ts's.

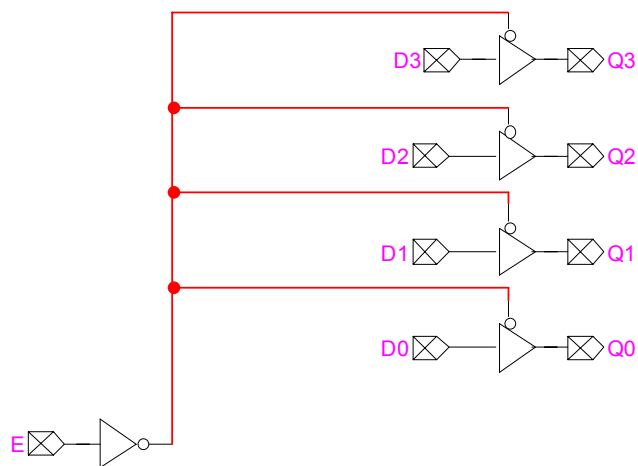


Figure 1
 Prepackaged 4-bit ts buffer

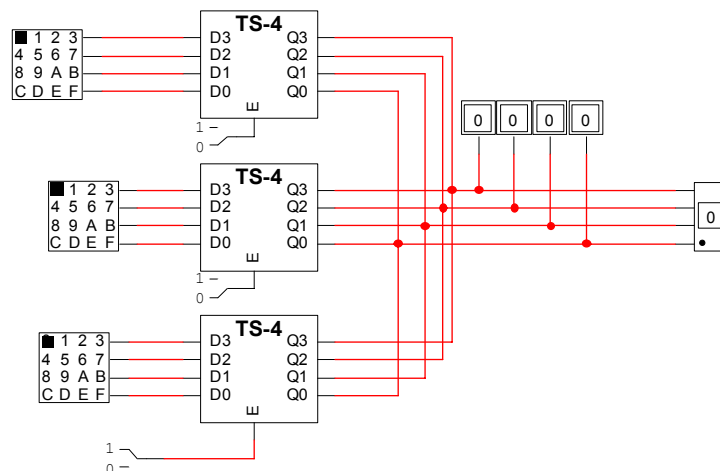


Figure 2
 Packaged 4-bit ts buffer and implementation of three ts buffers

Testing Task (a)

After completing the design of the three ts buffers, I experimented with a 4-bit bus to see how the ts buffers worked. I already know that the purpose of a ts buffer is to keep conflicts arising between values. This idea is amplified when I entered 2 or three different conflicting values into each ts buffer and enabled the active-high control. The ts buffer showed a C for conflict, letting me know that there was a conflict with the two values I entered. I did two different experiments, which lead me to believe that my ts buffers were working correctly.

Table 1 shows the values I used for experiment 1.

Note: **Values (x,y)** are values I entered into the first and second ts buffers. **Outcome** is equal to the values shown in the probes when two of the three ts buffer controls were enabled.

Values (x,y)	Binary value of x	Binary value of y	Outcome
(1,1)	0001	0001	0001
(1,2)	0001	0010	00CC
(1,3)	0001	0011	00C1
(0,F)	0000	1111	CCCC

Table 1
Experiment 1 Values

Table 2 shows the values I used for experiment 2.

Note: **Values (x,y,z)** is the value I put into each of the three ts buffers. These values were stored in the ts buffers and then all controls were disabled until the test. I then enabled the third ts buffer first, then the second, and then the first. So in this example, when I enabled the third ts buffer, the value in the probes were 0110 (6). I kept the third control enabled and then enabled the second control. The values shown in the probe were now 0C10. I had a conflict because I was trying to enter a 0010 while a 0011 was already the output value. **Table 2** shows the complete test respectively.

Values (x,y,z)	Binary value of x	Outcome	Order of Enable
X = 3	0011	0C1C	3
Y = 2	0010	0C10	2
Z = 6	0110	0110	1

Table 2
Experiment 2 Values

Task (b)

For task (b), I was asked to create the sample register transfer system that is in the Lec-5 lecture notes. To complete this task, I had to do 6 subtasks. Those subtasks are listed below:

1. Create the simple ALU
2. Test simple ALU
3. Design the datapath as seen in the Lec-5 lecture notes
4. Test datapath
5. Design the control unit as seen in the Lec-5 lecture notes
6. Implement the control unit within the datapath and test the conditions alpha, beta, and gamma, respectively.

Because all of the work has already been done/given (i.e. top level design of datapath, activation table, and Boolean equations), I am going to discuss each subtask briefly. I understand that the focus of this task was to enhance my understanding of a register transfer system. I also understand that it was my responsibility to study the top level design of datapath, activation table, and Boolean equations for this example in order to better my understanding.

Subtask 1 – Creat the simple ALU

The *prepackaged* simple ALU I created for this task is shown in **Figure 3**.

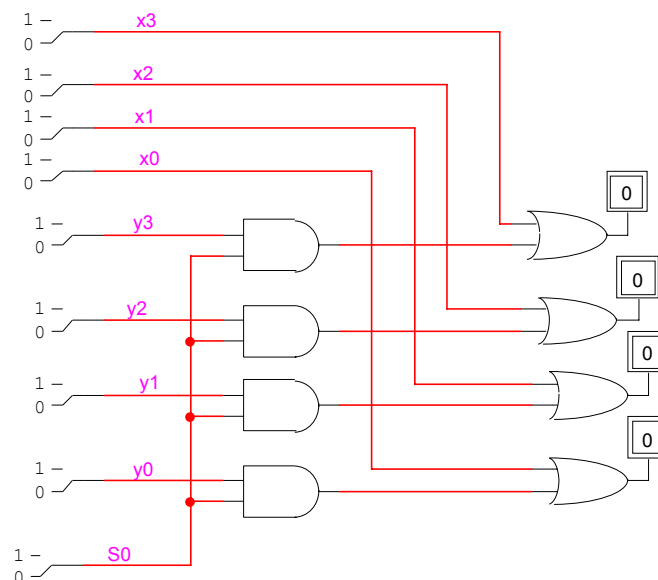


Figure 3
Prepackaged Simple ALU

The packaged simple ALU is shown in **Figure 4**.

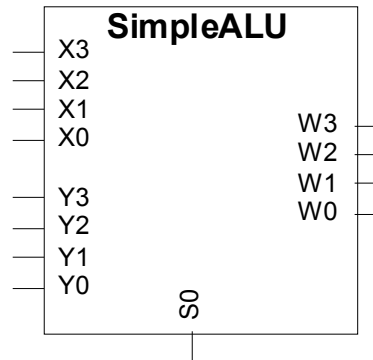


Figure 4
Packaged Simple ALU

Subtask 2 - Test simple ALU

From reading the Lec-5 lecture notes, I already know what the ALU is supposed to do. When $S0 = 1$, do an OR operation. When $S0 = 0$, pass input from X to the output. When I tested the simple ALU as seen in **Figure 3**, I kept those facts in mind.

I did 3 tests on the simple ALU circuit. I first tested to see if the OR operation worked correctly when I set $S0 = 1$. I did this by first setting the values of X3..0 to 1010. I then set the values of Y3..0 to 0010. The results of the output should be the OR of these two values.

If we look at 1010 and 0010 like this:

```

1010
+
0010
-----
1010
    
```

1010 should be my output. **Figure 5** will show that this is true:

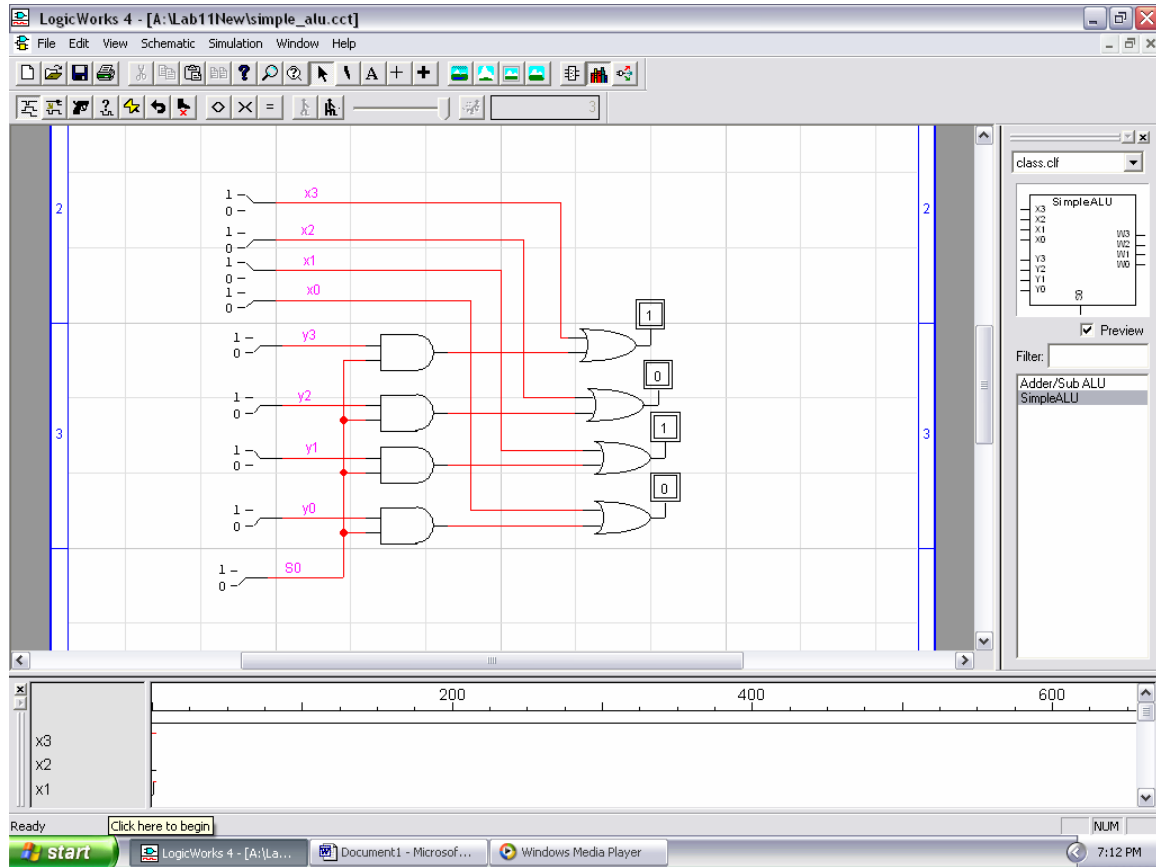


Figure 5
Test 1 For Simple ALU

I still wanted to make sure the OR operation worked correctly, so I put a set of different values in X3..0 and Y3..0. I first put 0100 in X3..0 and then I put 1000 in Y3..0. If we at 0100 and 1000 like this:

$$\begin{array}{r} 0100 \\ + \\ 1000 \\ \hline 1100 \end{array}$$

1100 should be my output. **Figure 6** will show that this is true:

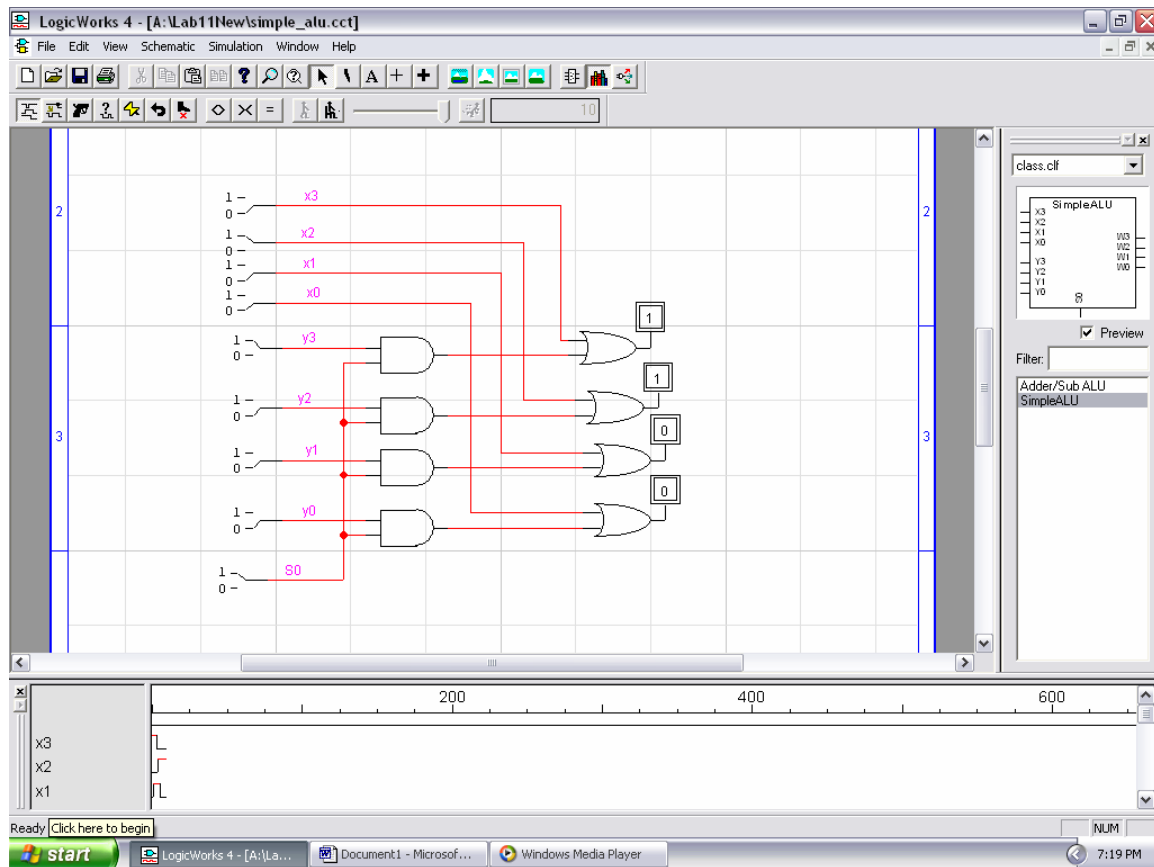


Figure 6
Test 2 For Simple ALU

For my third test, I wanted to make sure that when I set $S0 = 0$ that the current values in X would be in the shown in the output. For ease, I will use the same values I used in test 2, (0100 and 1000 respectively). When I set $S0 = 0$, I expect my values to be 0100 and not 1100. **Figure 7** will show that the correct values appear:

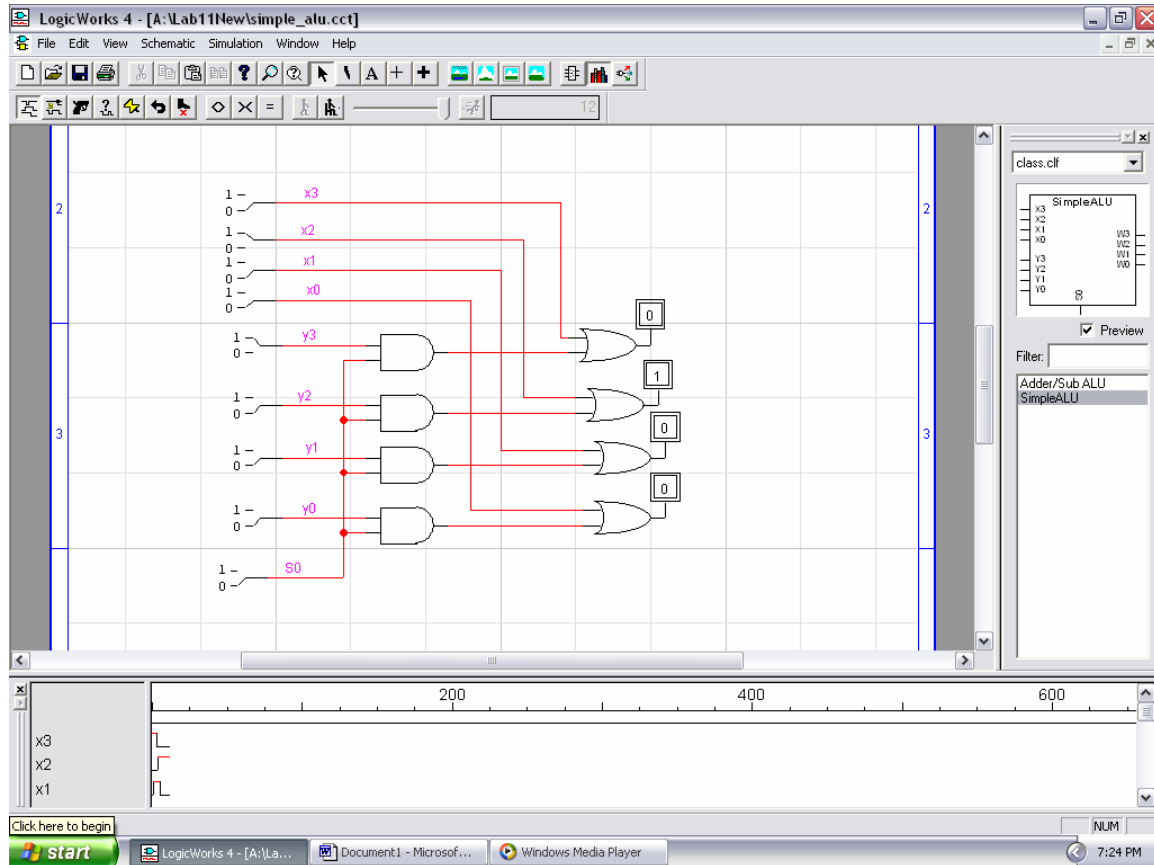


Figure 7
Test 3 For Simple ALU

Subtask 3 - Design The Datapath As Seen In The Lec-5 Lecture Notes

The design for this example included 3 4-bit registers, the simple ALU I created, and three ts buffers. 4 NOT gates were also incorporated to handle the gamma condition, ($Y \leftarrow Z'$). **Figure 8** shows my completed design:

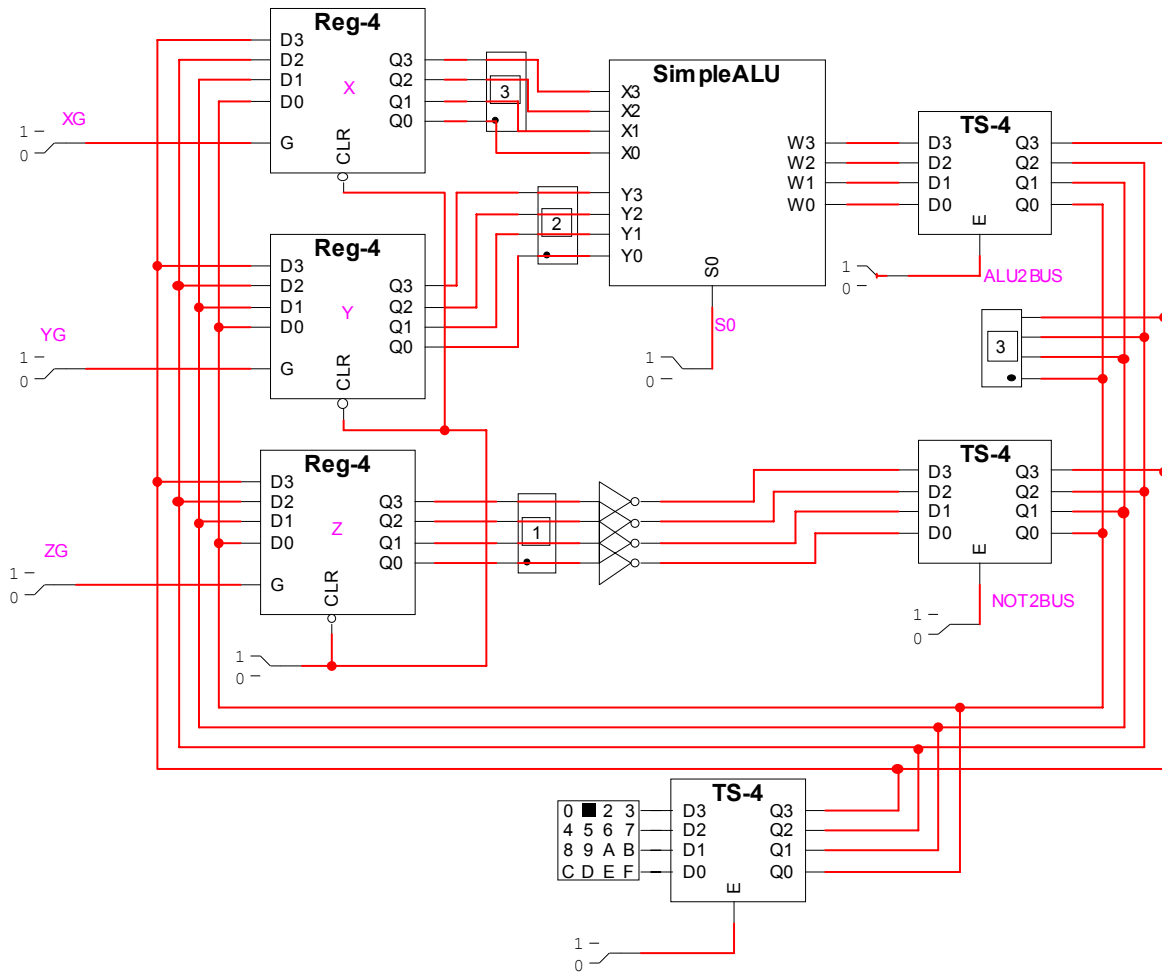


Figure 8
 Design Of Datapath

Subtask 4 - Test Datapath

Testing the datapath was relatively easy to do after I figured out exactly what was going on within the register transfer system. I did LOTS of testing on this design, but for ease, I will discuss 3 of the tests I performed.

I was given three conditions to test on this design. Each condition and their definition is listed below:

1. Alpha: $X \leftarrow X \vee Y$ – When alpha is true, make a bit-by-bit OR operation between the contents of register X and Y and put the result back to register X at the rising edge of G

2. Beta: $Y \leftarrow X$ – When beta is true, take the contents of register Y and put them to register X at the rising edge of G.

3. Gamma: $Y \leftarrow Z'$ – When gamma is true, make a bit-by-bit NOT operation on the contents of register Z and put the result back to register Y at the rising edge of G.

Because I know what is supposed to happen for each condition, I am able to test the design properly.

My first test was to check the alpha condition. I did this by first inputting the values of A and 3 into the X and Y registers, respectively. I know that the binary value of A is 1010 and the binary value of 3 is 0011. If we look at 1010 and 0011 like this:

```
1010
+
0011
-----
1011
```

1011 or the decimal number for this value should be 11 (i.e. B in hexadecimal).

Figure 9 will show that this is true. The arrows within Figure 9 show how the SO, ALU2BUS, and the KEYBOARD were set for this test to work correctly. Notice that I had to set the KEYBOARD ts buffer to 0 in order to set the ALU2BUS ts buffer to 1. This goes along with the rule that only 1 ts buffer can be active-high at one time.

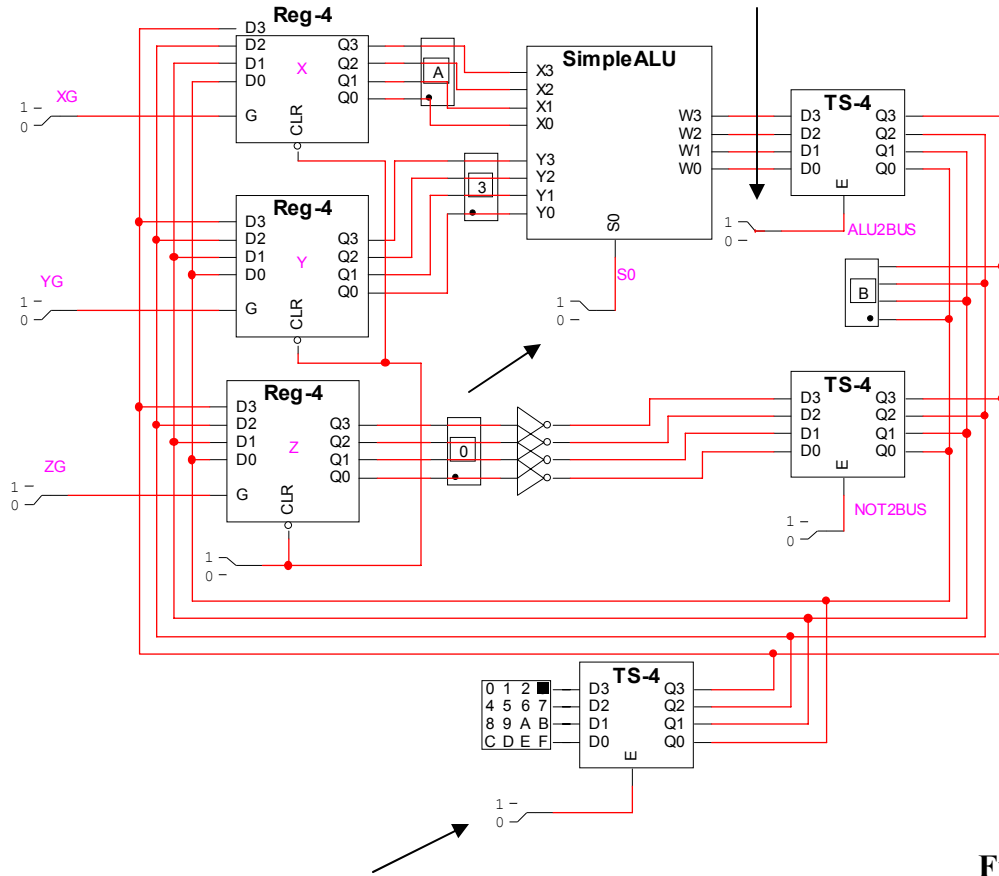


Figure 9
Test 1 Of Datapath

For my second test, I tested to see if the condition for beta worked correctly. For this test, I chose to keep the current values I had in the X and Y registers, (A and 3, respectively). If I set my ALU2BUS to 1, my S0 to 0, and my LD-Y to 1, the value currently in the X register should go into the Y register. In this example, the value of A should go into the Y register. **Figure 10** will show that this is true. As in the previous example, also note how the control units are set.

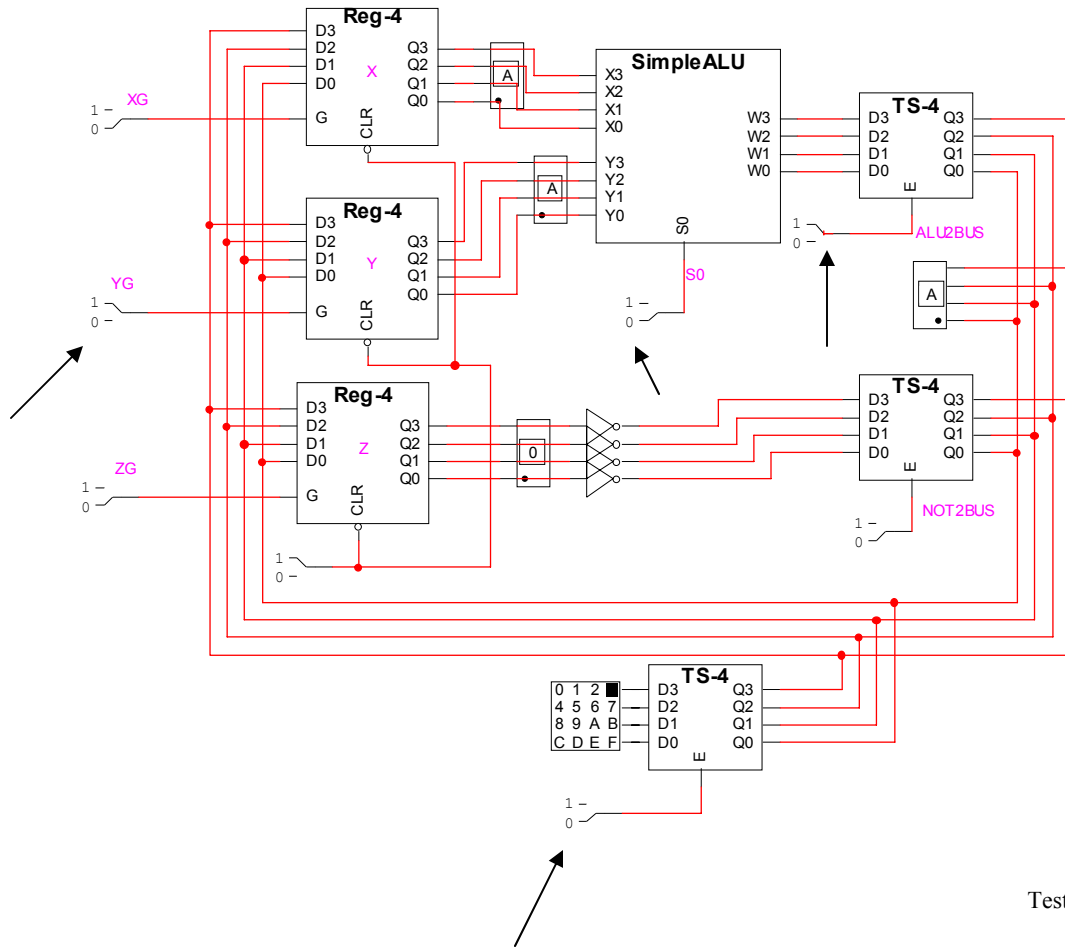


Figure 10
Test 2 Of Datapath

For my third test, I checked to see if the gamma condition worked correctly. To test the results, I first input the value of F into register Z. So in keeping with the current values I have, I know should have the value of A in register X, A in register Y, and F in register Z. If I set my NOT2BUS to 1 and my LD-Y to 1, the value of 0000 or simply 0, (compliment of F) should appear in the Y register. **Figure 11** will show that this is true. As before, notice the how the control units are set.

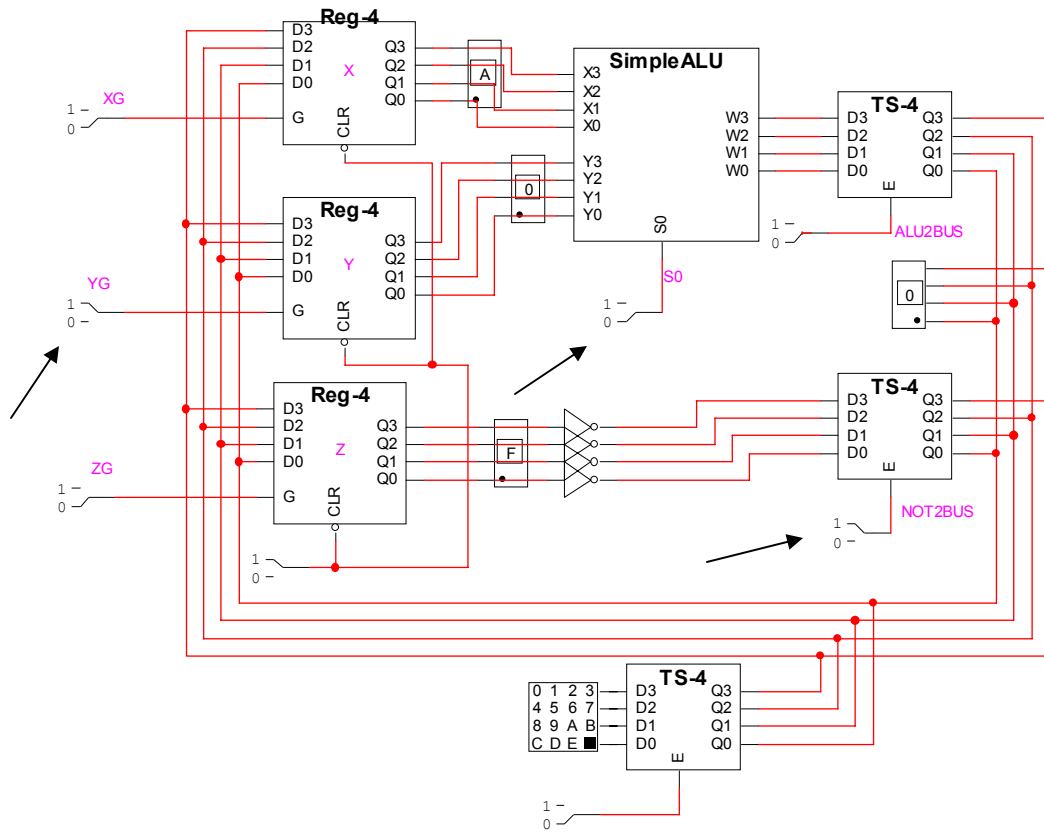
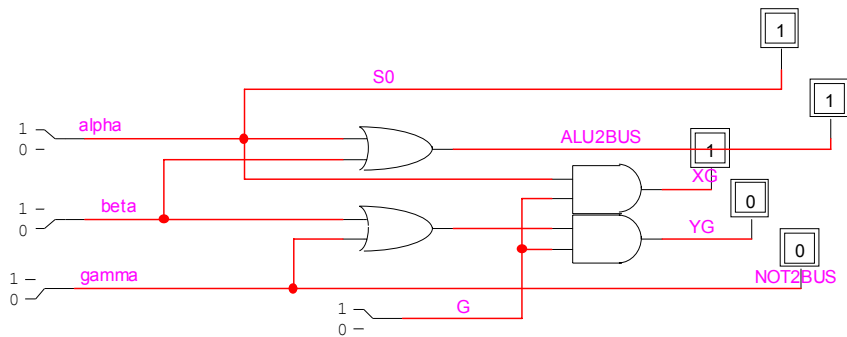


Figure 11
Test 3 Of Datapath

Subtask 5 - Design The Control Unit As Seen In The Lec-5 Lecture Notes

The control unit for this example was created on the basis of the S0 function table and the control points activation table. Both of these tables are in the Lec-5 Lecture Notes. **Figure 12** shows my design as well as a test on the unit. As you can see, alpha is set to 1. According to the control points activation table, when alpha is true, the ALU2BUS, S0, and the LD-X (or in this case, XG), should be true, or equal to 1. My example in **Figure 12** proves that the table is the unit is correct based on the table give in the Lec-5 Lecture Notes.

Figure 12
Control Unit Design



Subtask 7 - Implement The Control Unit Within The Datapath And Test The Conditions Alpha, Beta, And Gamma, Respectively

Once I had the control unit designed, I was able to incorporate it into the datapath that you saw earlier in **Figure 8**. **Figure 13** shows the final register transfer system I created:

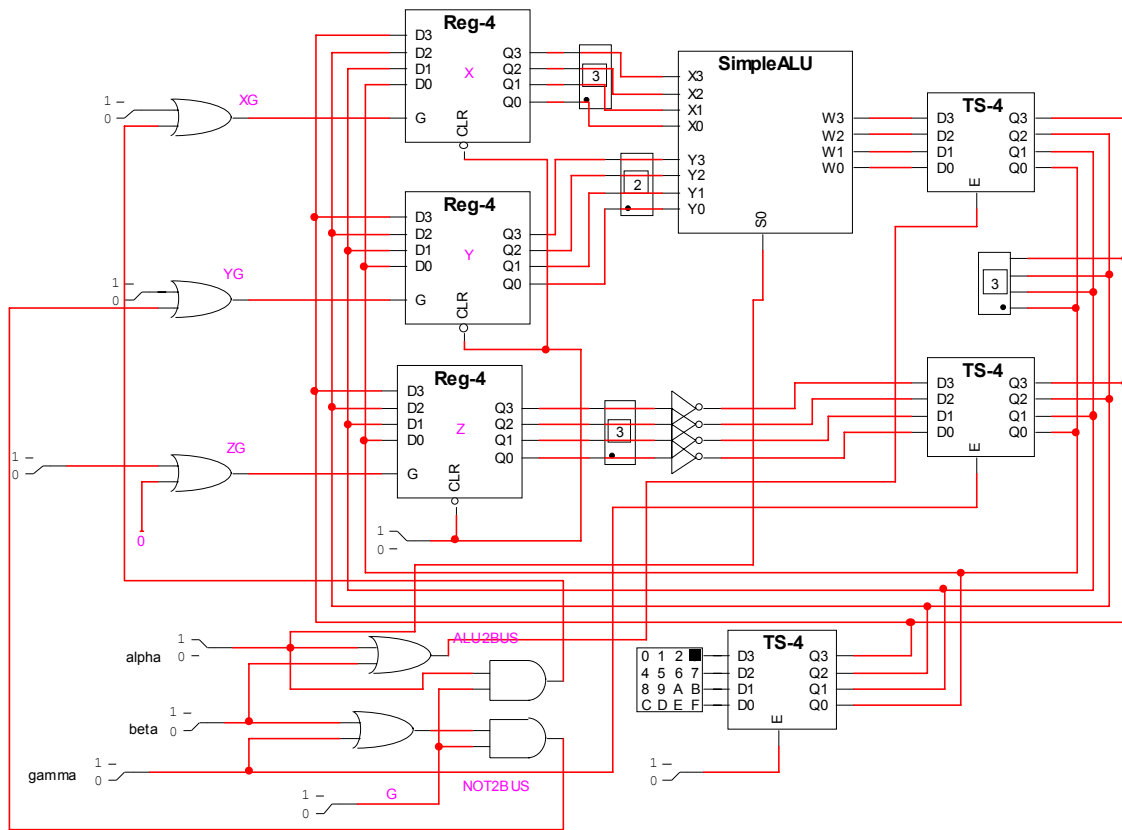


Figure 13
 Register Transfer System

The testing I did was very similar to that of when I tested my datapath. The only different now is that I have the control unit incorporated into my design. I will briefly show three tests I performed to make sure this register transfer system worked correctly.

Figure 14 shows my initial values of 7, 8, and 5 in registers X, Y, and Z respectively.

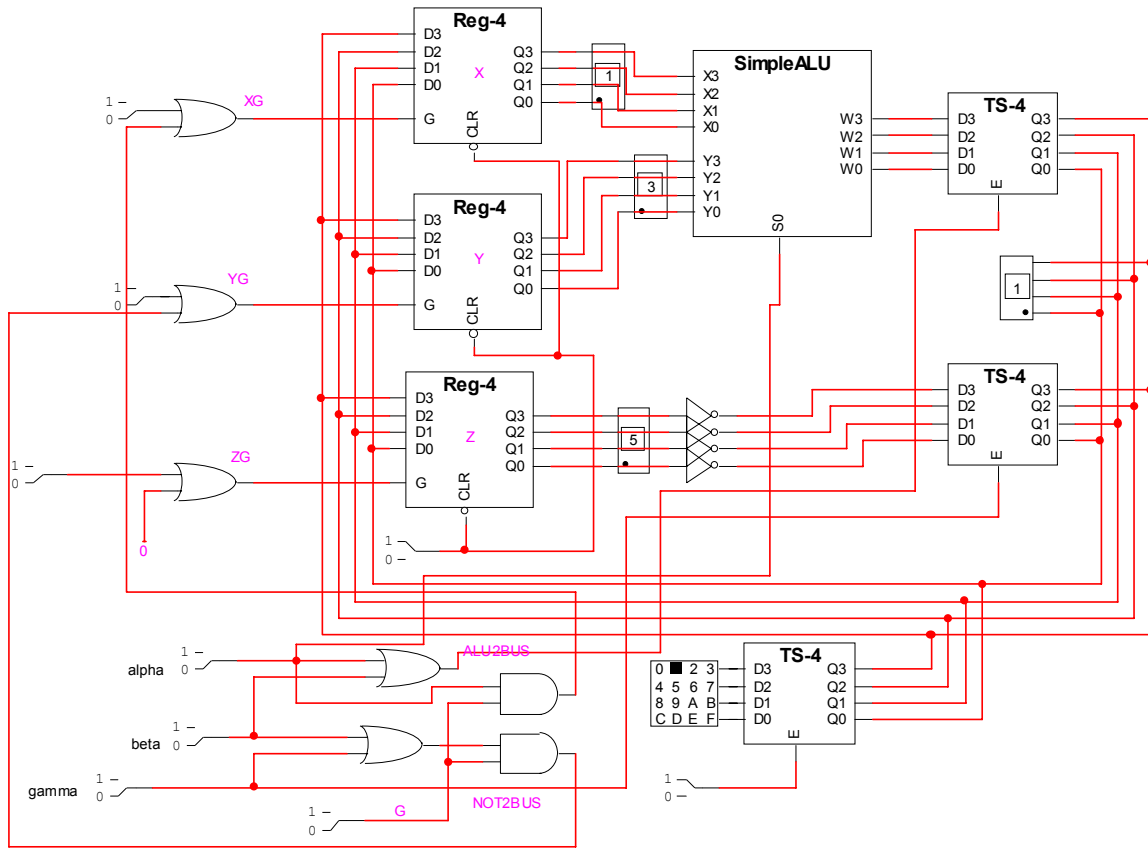


Figure 14
Initial Values

Once I had my values set, I tested to see if the alpha condition worked correctly. When I set my alpha = 1, I should see the value of F go into the output.

Then, once I set the common control of G to 1, the value of F should go into the X register. **Figure 15** shows that this is true:

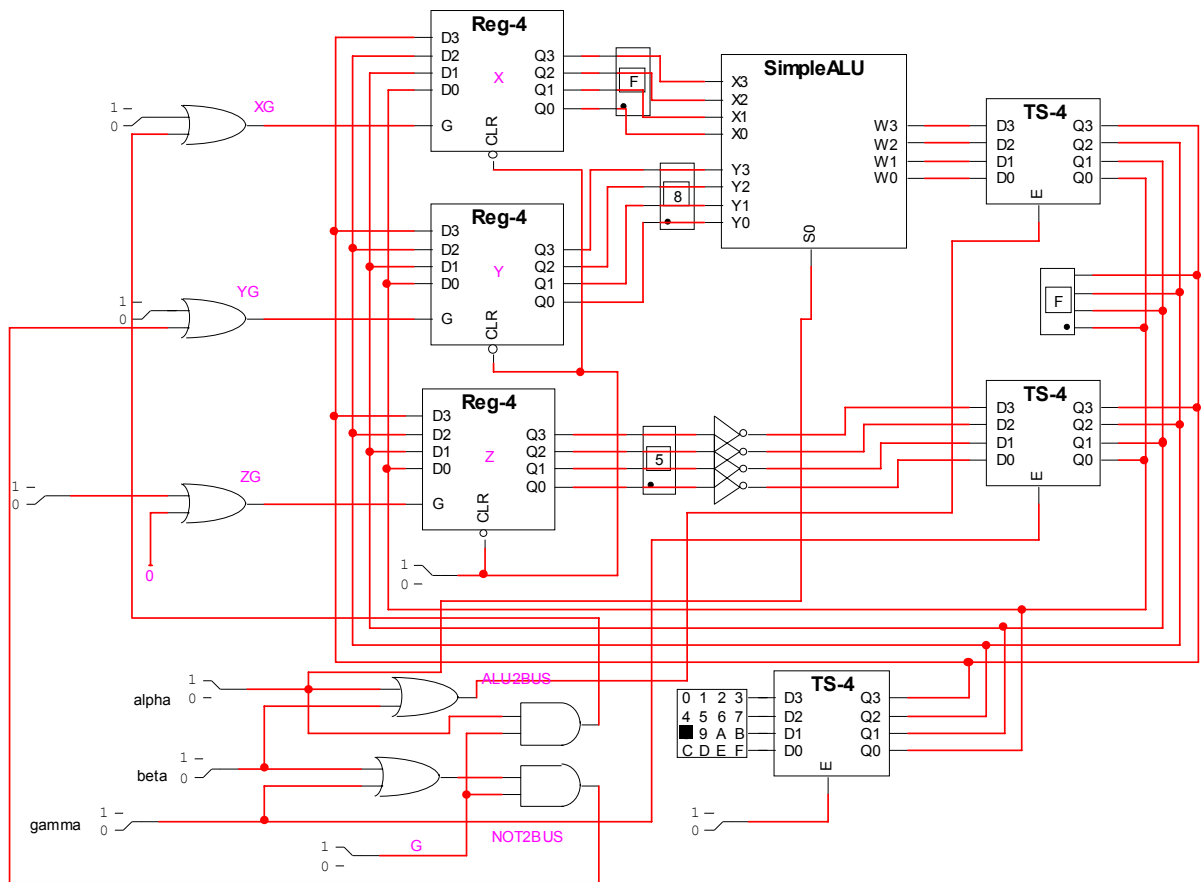


Figure 15
Test 1 Of RTS

For my third test, I tested the gamma condition. As you can see in **Figure 16**, the current value is 5. When I set beta back to 0 and then set gamma to 1, which in turn also sets the NOT2BUS to 1, the new output value will be the compliment of 5, which is A (10). Then, after I set the common G to 1, the value of A should be transferred into the Y register. The current value in the Y register will change from F to A. **Figure 17** shows that this is true:

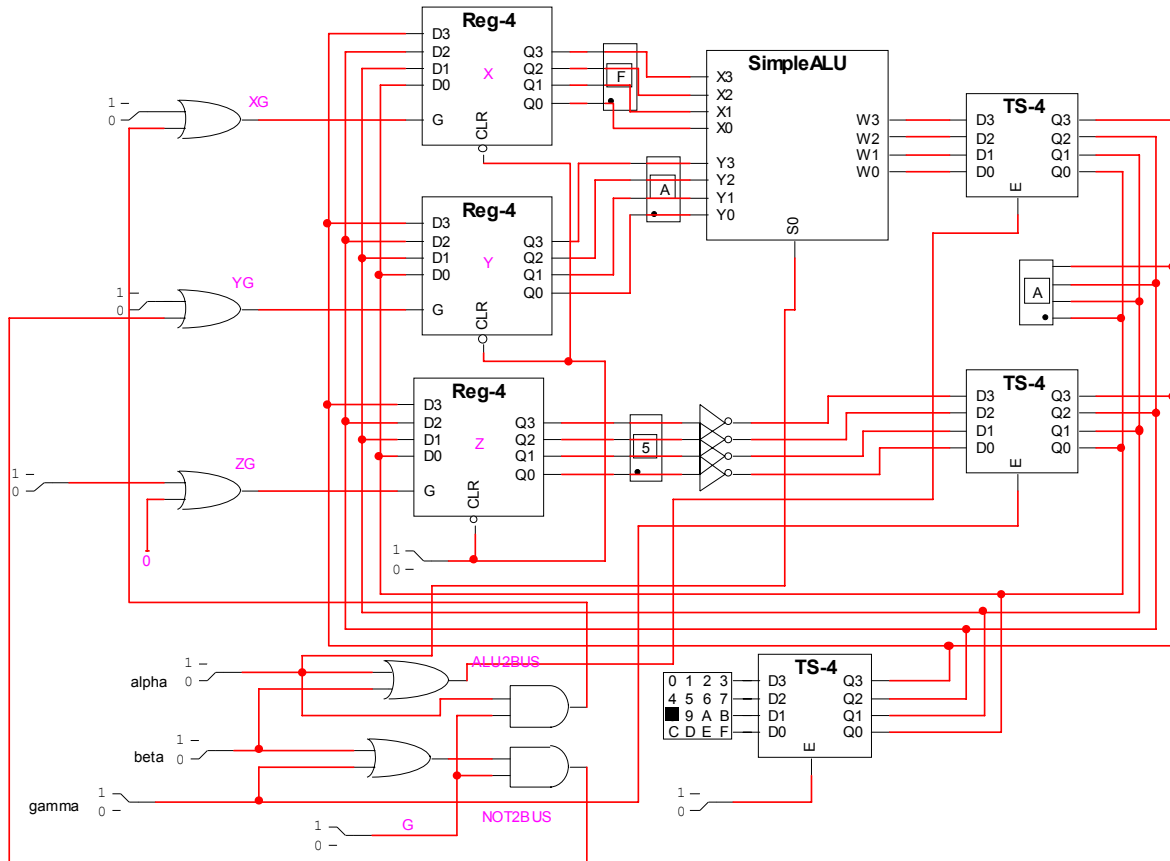


Figure 17
Test 3 Of RTS

The above discussion concludes task (b) of assignment 3. I will now begin discussing task (c).

Task (c)

For task (c), I was asked to create my own register transfer system that implemented a new set of conditions. To complete this task, I had to do 7 subtasks. Those subtasks are listed below:

1. Create the ALU, test is and create its function table.
2. Design the top level design of the datapath and list the control points.
3. Design the design of the datapath.

4. Test the datapath.
5. Design the control unit activation table.
6. Form Boolean equations based on my activation table.
7. Design and incorporate the control unit with the designed datapath and do final testing.

Subtask 1 - Create The ALU And Its Function Table

I was given three conditions to test on this design. Each condition and their definition is listed below:

1. **Alpha:** $X \leftarrow X + Y$ – When alpha is true, add the contents of registers X and Y and put the result back to register X at the rising edge of G
2. **Beta:** $Y \leftarrow X - Y$ – When beta is true, subtract the contents of registers X and Y and put the result back to register Y at the rising edge of G.
3. **Gamma:** $X \leftarrow Z'$ – When gamma is true, make a bit-by-bit NOT operation on the contents of register Z and put the result back to register X at the rising edge of G.

Because I know these conditions, I am able to decide how to build an appropriate ALU. Because alpha and beta perform addition and subtraction, I simply designed an adder/subtractor for my ALU. I already had a 4-bit carry-look-ahead adder that I designed in a previous lab, so the only work to be done was to implement the subtracter part of the circuit. I was able to find the design in the lecture notes from Lec1-2.

Figure 18 and 19 show the inside of my 4-bit carry-look-ahead adder and the full adder/subtractor, respectively:

Note: Because I have previously built the 4-bit carry-look-ahead adder and have already tested it, I am certain that it works correctly.

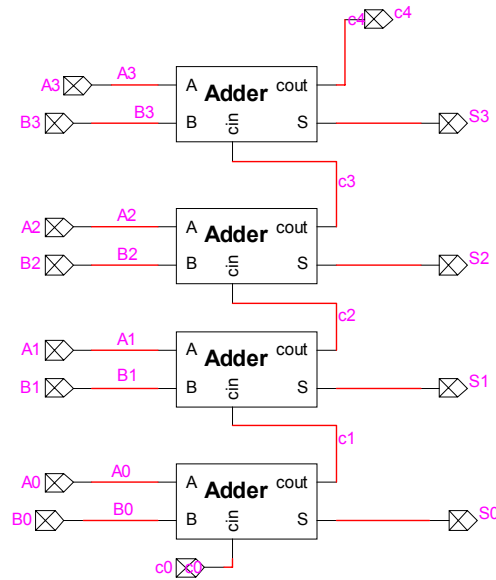


Figure 18
Prepackaged 4-Bit Carry-Look-Ahead Adder

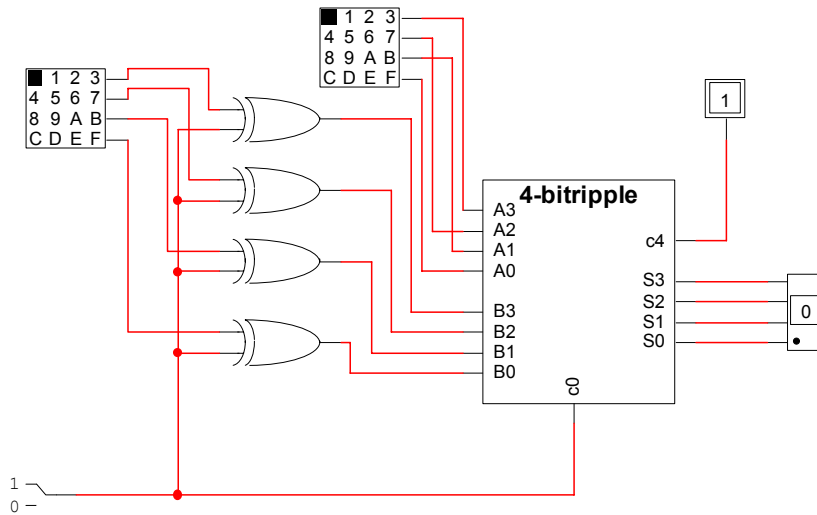


Figure 19
Adder/Subtractor

After I designed my ALU, I tested it for correctness. I already know that if I set $S_0 = 0$, then my ALU will perform addition. So knowing that leads me to believe that setting $S_0 = 1$ will result in the ALU performing subtraction.

If I input the values of 5 and 3 into my adder/subtractor, when I set $S_0 = 0$, I should see the value of 8 in the output. On the contrary, if I use those same two values and set $S_0 = 1$, I should see the value of 2 in the output. Figure 20 and 21 show that these values do indeed appear at the correct setting of S_0 .

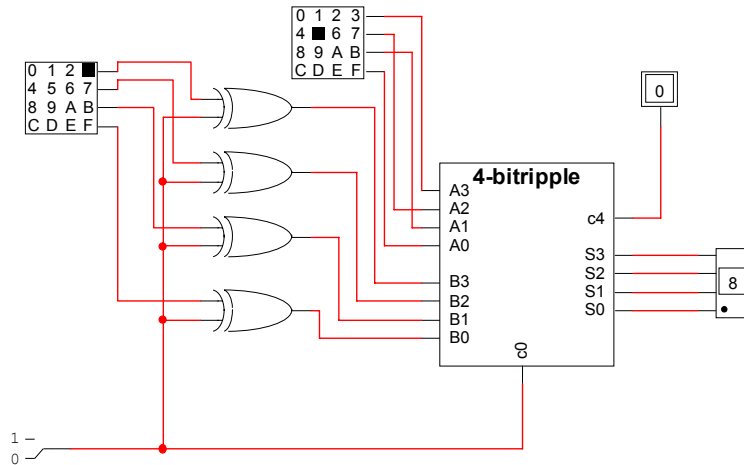


Figure 20
Test 1 Of ALU

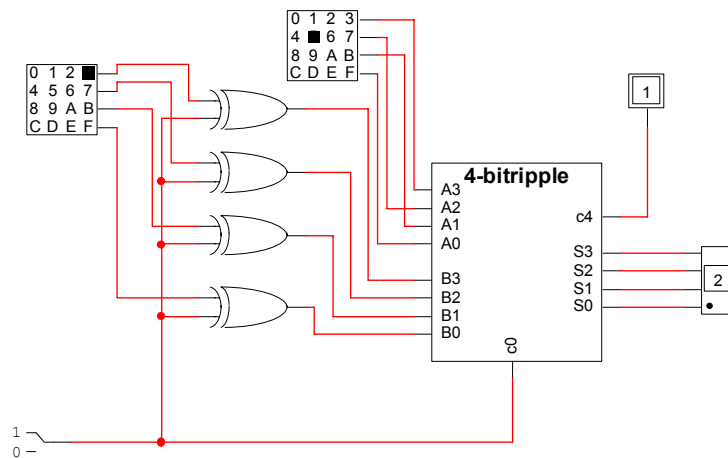


Figure 21
Test 2 Of ALU

After I created my ALU, I made an ALU function table to describe the output of the ALU according to its control points. **Table 3** shows the table I created:

S0	Output
0	$X + Y$
1	$X - Y$

Subtask 2 - Design The Top Level Design Of The Datapath And List The Control Points

Figure 22 shows the diagram of the datapath. Note: The blue lines represent the bus, and the short lines represent the control signals.

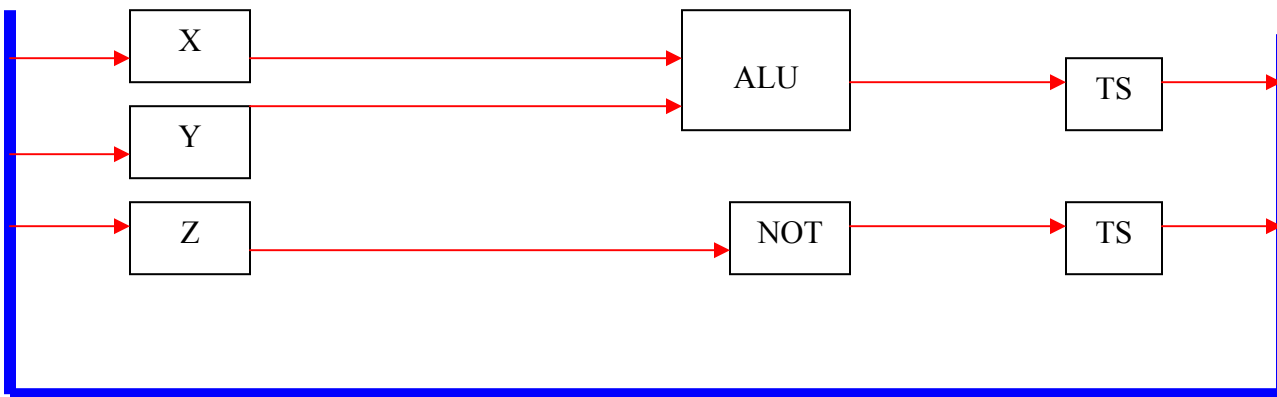


Figure 22
Diagram Of Datapath

Subtask 3 - Design The Design Of The Datapath

Once I had a diagram to go by for designing the datapath, I was able to implement it. Basically what I did to implement it was followed the diagram. **Figure 23** shows my design:

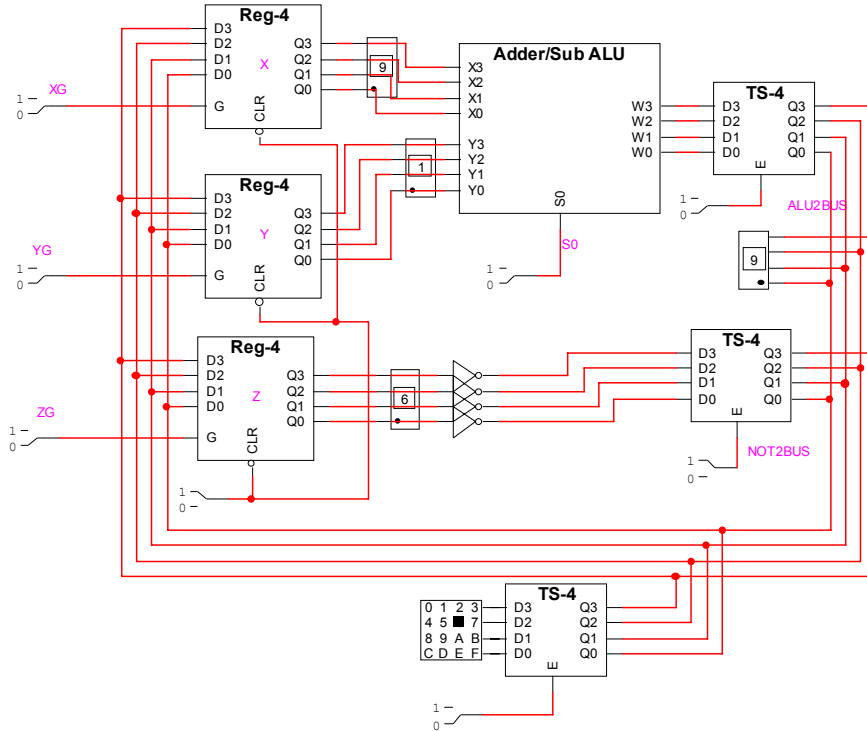


Figure 23
Datapath Design

Subtask 4 - Test The Datapath

I tested this circuit the same way I tested the circuit in task (b). My first test was to test the alpha condition. I know already that if I enter in the values of 5 and 3 into the X and Y registers and set $S_0 = 0$ and the ALU2BUS to 1, that the output should reflect the value of 8. I also know that upon the rising edge of G, the value of eight should be transferred into the X register. **Figure 24** shows that this is true.

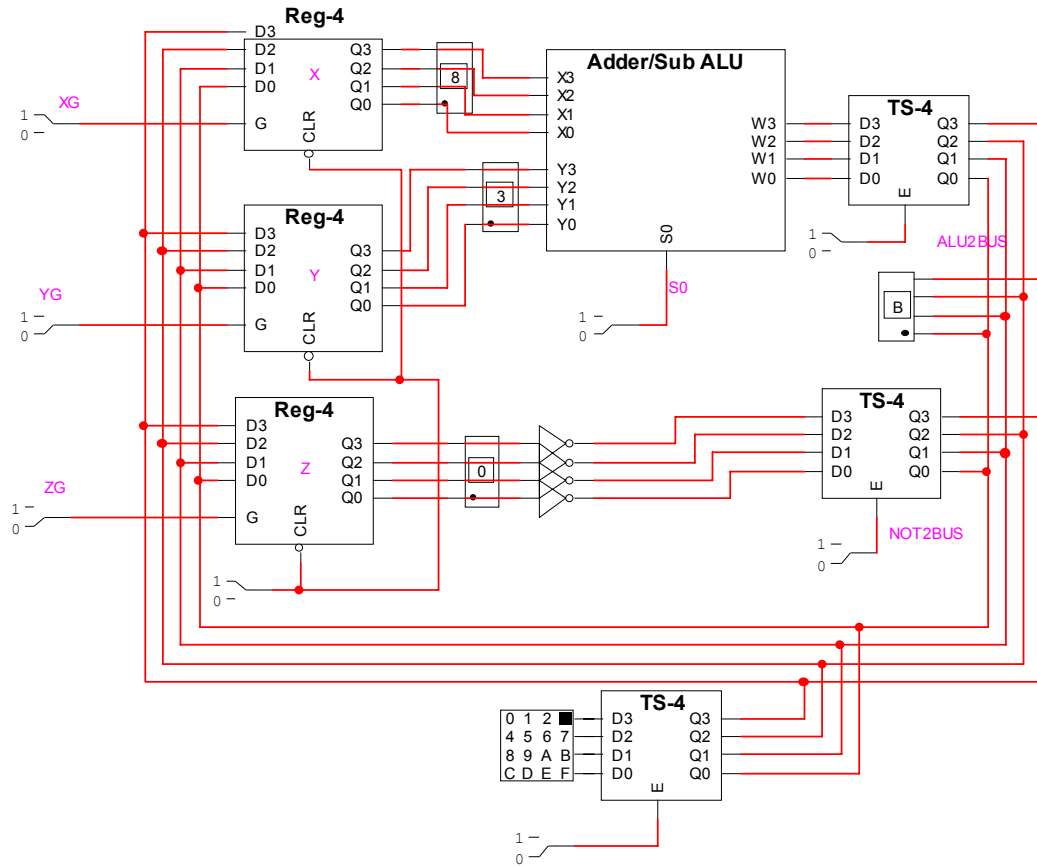


Figure 24
Test 1 Of Datapath

If I keep those two same values that are currently in the the X and Y registers (8 and 3, respectively), when I set $S0 = 1$, the value of 5 should appear in the output and upon the rising edge of YG, the value of 5 should appear in the Y register. **Figure 25** shows that this is true:

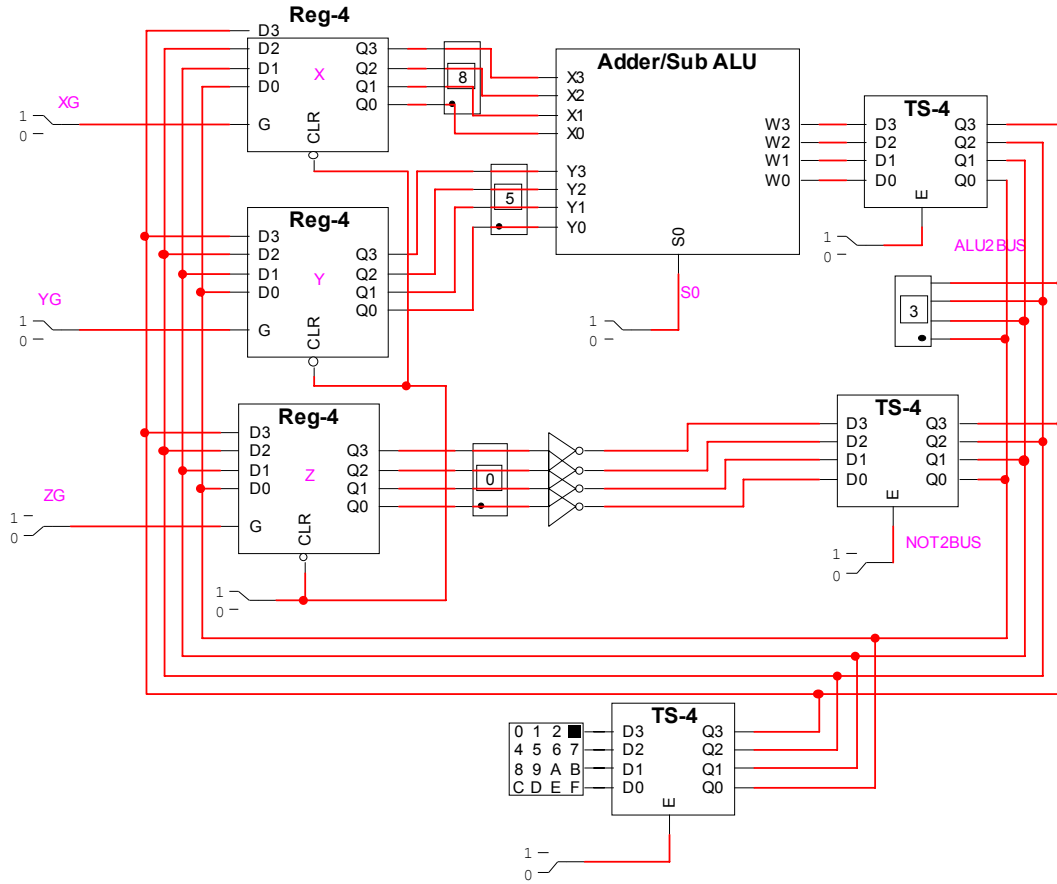


Figure 25
Test 2 Of Datapath

For my third test, I wanted to check and see if the gamma condition worked correctly. I set the value of 3 into the Z register as well as set the NOT2BUS to 1. The value that appeared was C, which is the compliment of 3. Upon the rising edge of XG, the new value in the X register was now C. **Figure 26** shows that this is true:

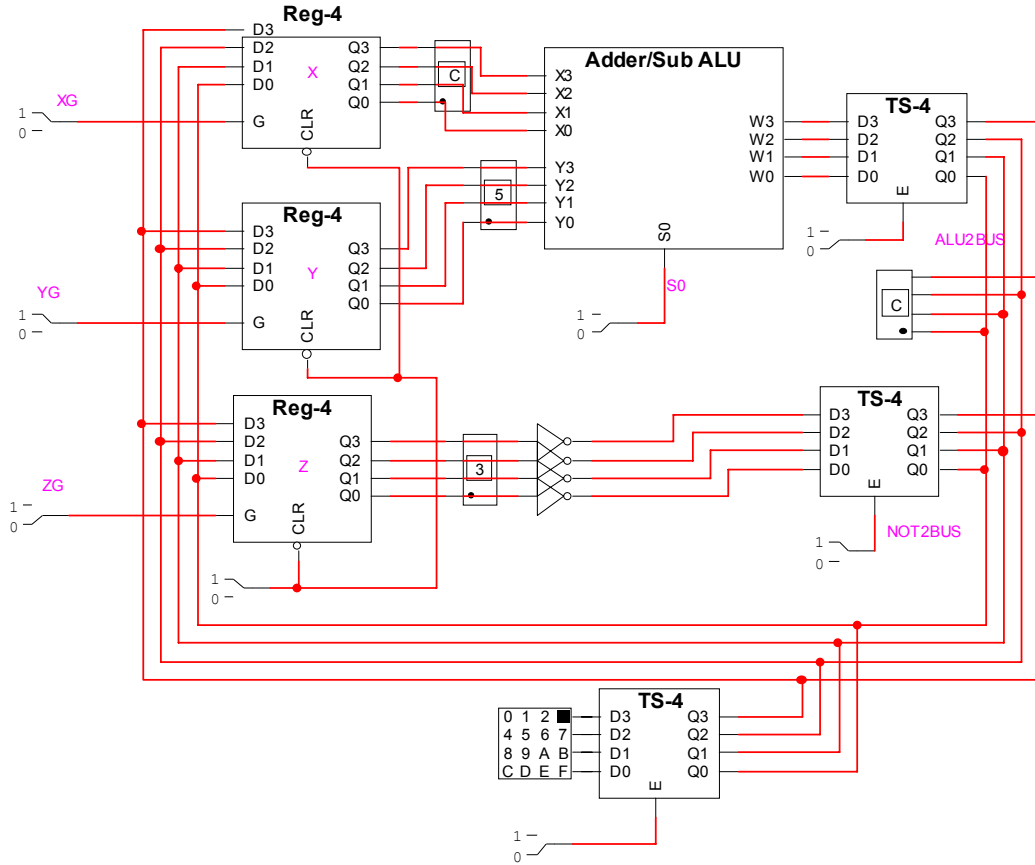


Figure 26
Test 3 Of Datapath

Subtask 5 - Design the control unit activation table

Table 4 shows the control unit activation table I created for task (c).

		ALU2BUS	NOT2BUS	S0	LD-X	LD-Y	LD-Z
Alpha	$X \leftarrow X + Y$	1	0	0	1	0	0
Beta	$Y \leftarrow X - Y$	1	0	1	0	1	0
Gamma	$X \leftarrow Z'$	0	1	0	1	0	0

Table 4
Control Unit Activation Table

Subtask 6 - Form Boolean Equations Based On My Activation Table

Below are the equations I formed based on my activation table. I found these equations by checking each column for 1s and creating an OR condition. For example, the ALU2BUS has a 1 in both the alpha and beta rows; therefore my equation will be $\alpha + \beta$.

Control Point Equations:

$$\text{ALU2BUS} = \alpha + \beta$$

$$\text{NOT2BUS} = \gamma$$

$$\text{SO} = \beta$$

$$\text{XG} = \alpha + \gamma$$

$$\text{YG} = \beta$$

$$\text{ZG} = 0$$

I knew that the load control signals also needed a further AND with G, so I also came up with the following equations for the AND gates that are in my control unit circuit:

Note: * denotes the AND operation.

$$X = (\alpha + \gamma) * G$$

$$Y = \beta * G$$

$$Z = 0 * G$$

Subtask 7 - Design And Incorporate The Control Unit With The Designed Datapath And Do Final Testing On The Register Transfer System

In order for me to complete the complete register transfer system, I first need to get form the equations I discussed in subtask 6. Once I had those formed, it was easy for me to create the appropriate circuit and incorporate it into my final design. **Figure 27** shows what my final design looks like:

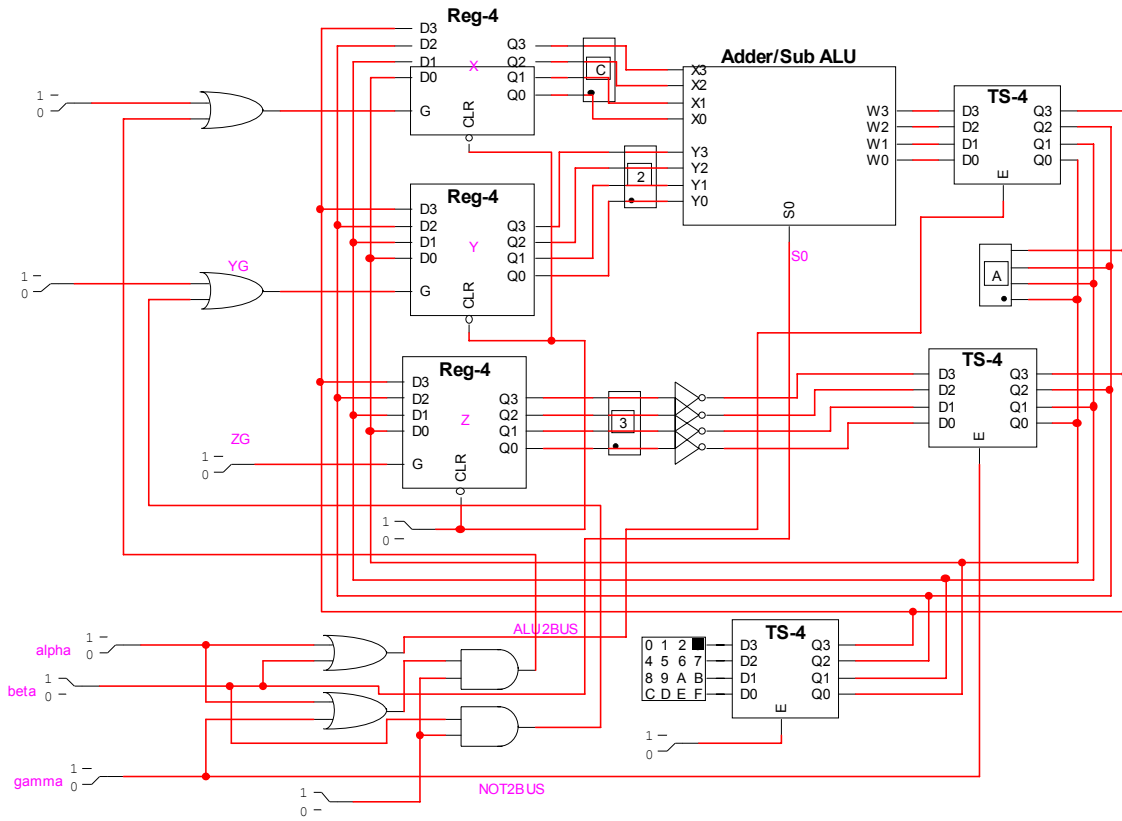


Figure 27
 Final Design

Once I designed my register transfer system, I tested each of the conditions, (alpha, beta, and gamma, respectively) to make sure my complete design worked properly.

For my second test, I kept the same values that are currently in Figure 28, (F and 5 in the X and Y registers, respectively). When I set beta = 1 the new value in the output was A (10), which is F (15) subtracted from 5. Upon the rise of the common control G, the new value in the Y register was A. Figure 29 shows that this is true:

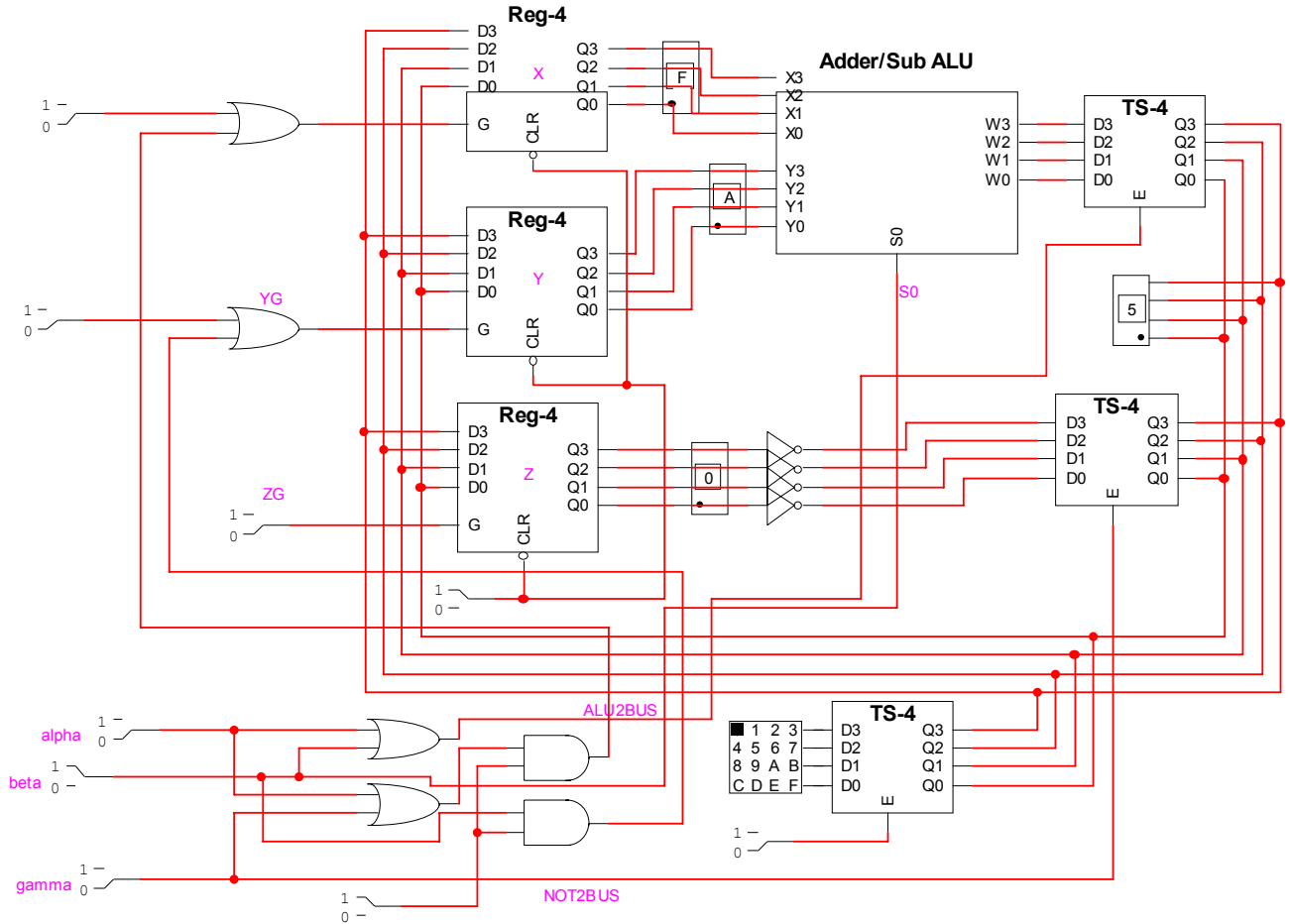


Figure 29
Test 2 Of RTS

For my final test, I put the value of F (15) into the Z register, set the NOT2BUS = 1, and set gamma = 1. When I did that, the new value in the output was 0, which is the compliment of 1111. Upon the rise of the common control G, the value of 0 was transferred to the X register. **Figure 30** shows that this is true:

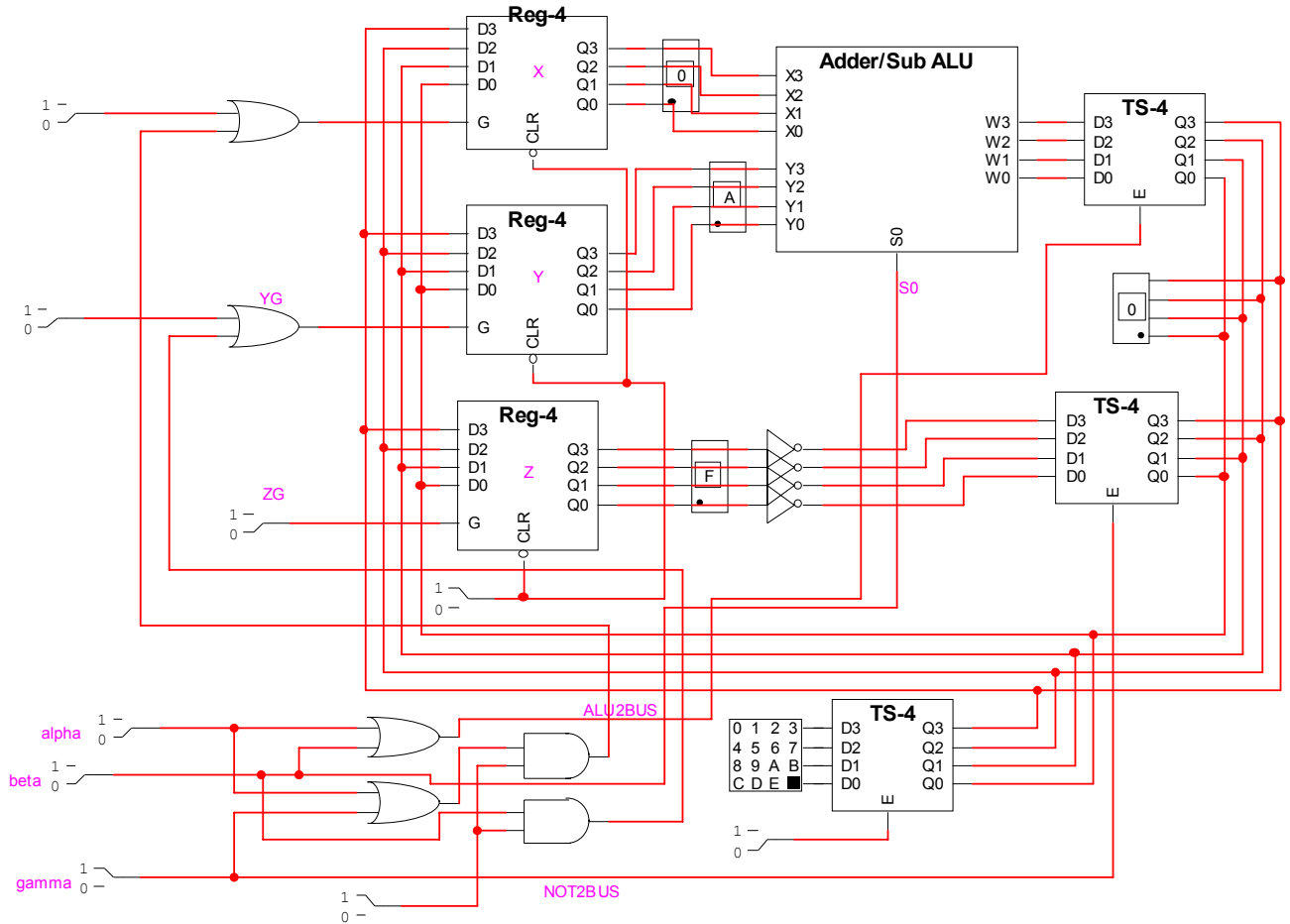


Figure 30
Test 3 Of RTS

This concludes the end of all tasks.

Note: For ease and space, I have only discussed three different tests in this report to show that my RTS works correctly; however, I did more testing than merely three tests to determine the correctness of the complete circuit.