National Sun Yat-sen University
Department of Electrical Engineering

112-1 Practical Digital System Design

HW2 Voting and Median Circuit Design

Name: 胡庭翊

Student ID: B103015006

I. Architecture

A. Voting Circuit Design

1. Characteristics

The characteristic of a voting circuit is that, it can accept five three-bit one-hot (i.e., only one bit will have the logic-1 value) input, and outputs the three bit one-hot number that occurred most often on the inputs.

2. Advantage

The advantage of using three bit one-hot as our input is that it is easier in comparison, we only need to compare the number of who chose the specific bit. If we have multiple chosen voting or if our input is in different types of representation, the circuit must be more complicated.

We can apply this circuit in many different places, for example, we can adapt this voting circuit when we are designing a circuit that may receive multiple inputs by time, however, there is only one input needed. It can be a filter that filters out the disturbance in some way.

3. Disadvantage

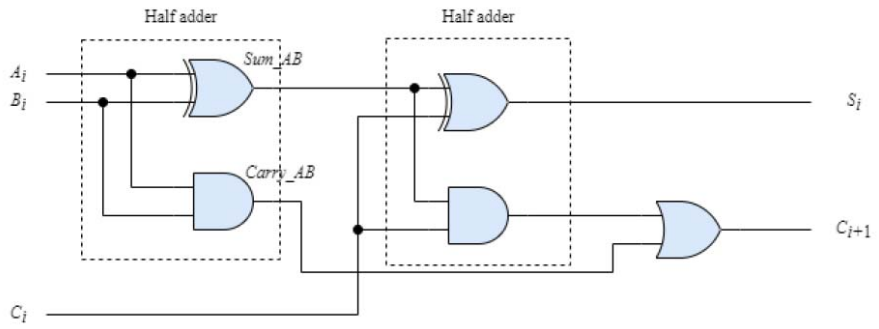
Since we have 5 inputs, it means that we may encounter selection problems while there are inputs of the same number of numbers (e.g. 100, 010, 001, 010, 100) . Hence, we set 100 as the first priority, 010 as the second and 001 as the last to solve this problem. For instance, if there are the same numbers of 100 and 010, we will select 100 as our output.

4. Operation Method

To implement the circuit, I designed a 1-bit to 3-bits ripple carry adder for the use of calculating how many people select a candidate first. I use the 1-bit to 3-bit ripple carry adder to add all the first, second, and last bit of inputs, so here we need a total of three 1-bit to 3-bit ripple carry adders.

A 1-bit to 3-bits ripple carry adder consists of 4 full adders,

and each full adder is made by 2 half adders. Hence, we need 12 full adders, that is, 24 half adders in total.



And as for how to generate the output, we can build a K-MAP simplified logic gate that includes all the cases (e.g. when the first bit has the biggest selected number, the output would be 100, when the second bit has the biggest selected number, the output would be 010…).

B. Median Circuit Design

1. Characteristics

The characteristic of a median circuit is that it accepts three 8-bit one-hot numbers: a2 [7:0], a1 [7:0], a0 [7:0], and outputs the input with the middle of the three values.

For example, if the inputs are a2 =10000000, a1=00010000, a3=00000001, then the output should be 00010000, which is the median of the three one-hot values.

2. Advantage

We can apply this circuit when there is a need to select the middle number of all the inputs. If we have multiple inputs and are needed to compare and select its middle value, applying this circuit can help.

3. Disadvantage

Since we have 3 inputs, if there are equivalent inputs, then the output might be confusing. Here, by algorithm, if we have equivalent inputs, the output would be the one which is

repeated.

4. Operation Method

To implement this circuit, I first designed an 8-bit ripple carry adder/subtractor and subtract one of the outputs with another. If the MSB is 0, that means the subtractor is smaller than the minuend (no overflow), else if the MSB is 1, it means that the subtractor is larger than the minuend. Let the three inputs be A, B, and C, we should thus make three 8-bit ripple carry adder/subtractor in order to compare A with B, B with C, and C with A.

After we get the relationship of A, B and C, we can then simply select the Median number, and by using K-MAP, we can then get the simplified circuit to restore the result.

Having the result, we then need to generate the output. I use two stages of multiplexer to select one bit, for instance, to generate output [0], I assigned a0[0] and a1[0] into the first MUX with a selection bit that knows if the median number is a1 or not. The output of the first MUX would be a1[0] if the selection bit is 1, else it will be a0[0]. Continuing, we assigned the first stage MUX output into the second MUX with a2[0] in comparison, the selection bit now will be the one that can verify if the median number is a2 or not. Following the same pattern, the output would be a2 if the selection bit is 1, and would maintain a0 if the selection bit is 0, this gives the result of our output[0].

Using the same algorithm, we need 8 groups of MUX, that is, 16 MUX in order to implement the circuit.

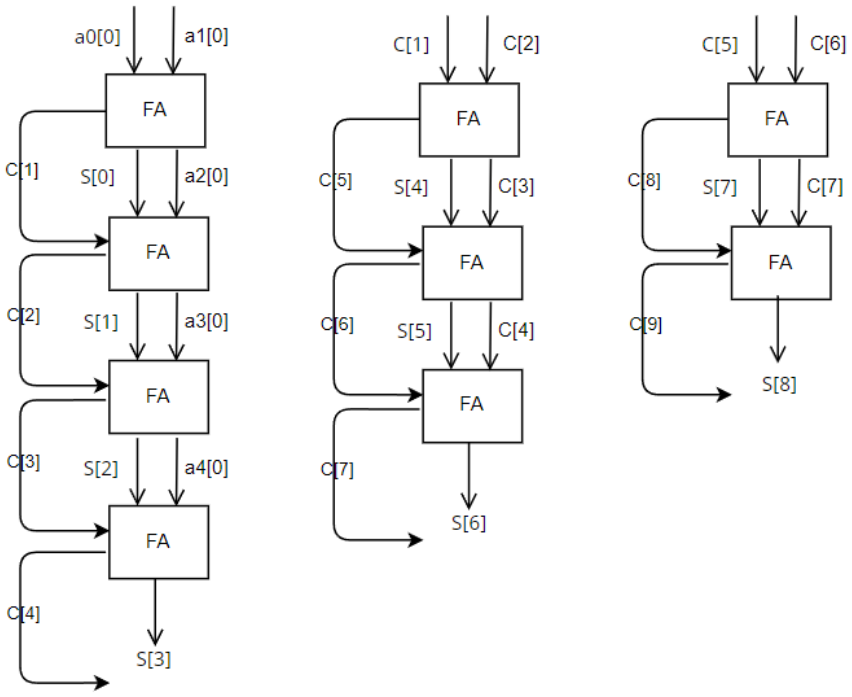
II. Algorithm

A. Voting Circuit Design

1. 1-bit to 3-bit Ripple Carry Adder

For each output bit i, we have to add all the components, that is, a0[i], a1[i], a2[i], a3[i], a4[i] together. We first add a0[i] and

a1[i], it will generate a sum[0] and a carry[1]. Sum a2[i] with sum[0], we can get sum[1] and carry[2] …, following this pattern, we will lastly get the sum of all the added bits, which requires a 4-bit ripple carry adder.



However, the output result is only one bit which is the LSB of the output. In order to generate the needed result (0 <= total sum <= 5), we would require at least 3 bits for representation.

Hence we continue to add those carry generated previously. We have carry [1], carry [2], carry [3], carry [4], following the same pattern, we can get the second bit by using a 3-bit ripple carry adder.

And as for the MSB, we sum the carry that the previous adder generated, that is carry[5], carry[6], and carry[7] together and we will receive the MSB.

We can keep adding the carry until there is no more carry waiting to sum, however, since the biggest number in our case is 5, that is 101, we only need three bits to represent. And hence we get our five 1-bit to one 3-bit ripple carry adder.

2. Logic Gates of Comparison

After implement three 1-bit to 5-bit ripple carry adder, we will get the total sum of 001, 010, and 100, here assume the

total sum of 001 is X, 001 is Y, and 100 I Z, we can accordingly get the pattern:

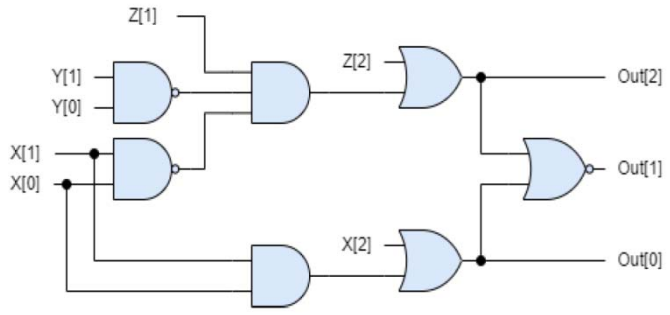
X	Y	Z	Out
5	0	0	
4	1	0	
4	0	1	001
3	1	1	
3	2	0	
3	0	2	
0	5	0	
0	4	1	
1	4	0	
1	3	1	010
0	3	2	
2	3	0	
2	2	1	
0	0	5	
1	0	4	
0	1	4	
1	1	3	100
2	0	3	
0	2	3	
1	2	2	
2	1	2	

According to the form, we can hence get the results:

Out[0] = X[0] & X[1] | X[2]

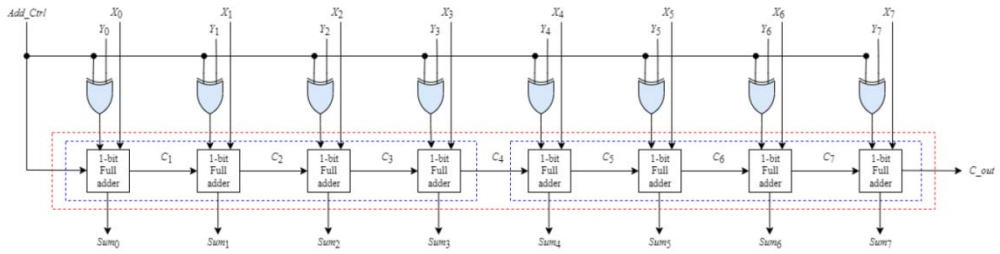
Out[2] = { ~(X[0] & X[1]) & ~(Y[0] & Y[1]) & Z[1] } | Z[2]

Out[1] = ~(Out[0] | Out[2])



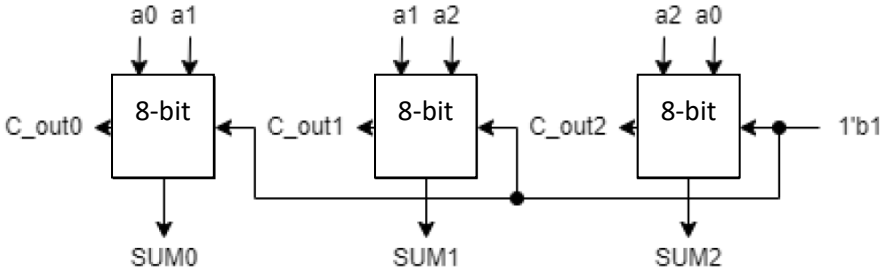
B. Median Circuit Design

1. 8-bit ripple carry adder/subtractor



Since we have three 8-bit inputs, we'll need three 8-bit ripple carry adder/subtracter to implement our function.

An 8-bit ripple carry adder/subtracter consists of 8 FA, while a FA is build with 2 HA. Because the ripple carry adder/subtracter here is used to comparison, we here set the mode to 1, that is, subtraction.



2. The median selection logic gates

Since we compare the size of the input by its MSB, we can thus assign A= SUM0[7], B=SUM1[7], C=SUM2[7]. The relation of _a1 (output select a1) and _a2 (output select a2) follows the chart below:

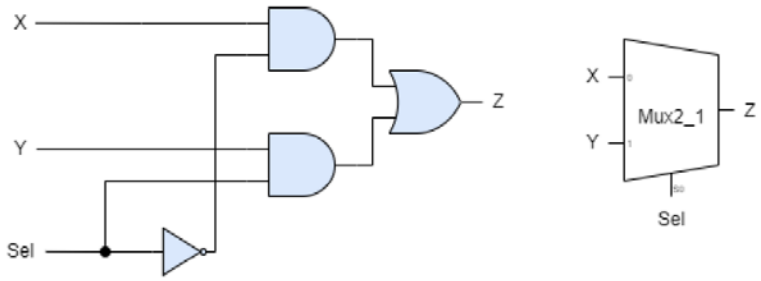
A	B	C	_a0	_a1	_a2
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	0	1
1	0	0	0	0	1
1	0	1	1	0	0

1	1	0	0	1	0
1	1	1	0	0	0

By K-MAP, we know that:
_a0 = A’ BC’ + AB’ C
_a1 = ABC’ +A’ B’ C
_a2 = A’ BC + AB’ C’

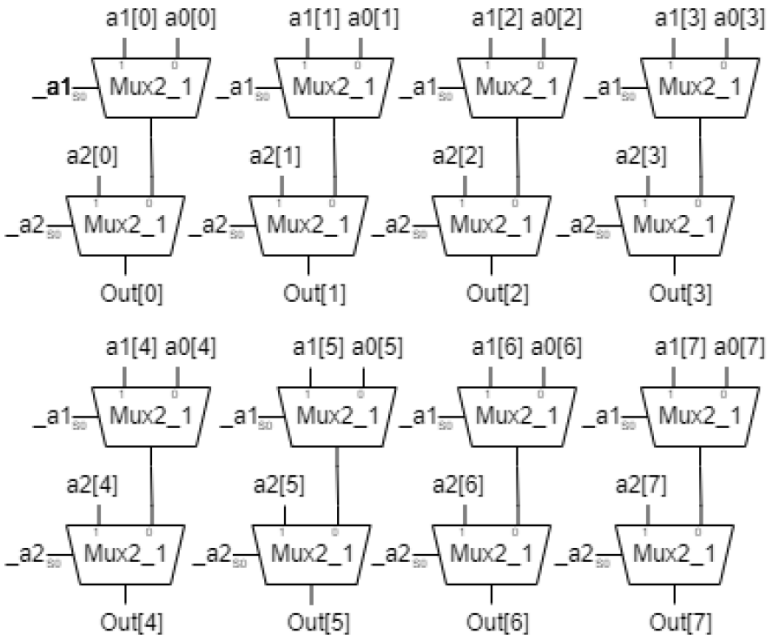
Since we can only use two select bits to justify our result, here we choose _a1 and _a2 as our select bits.

3. 2 to 1 MUX



The function of a 2 to 1 MUX is that, it can use a select bit to determine which of the two inputs would be the output. Its function follows the circuit above.

Here, since we have three inputs, we have to use two stages of MUX to implement our circuit to determine one bit. Hence, 16 MUX are needed to implement our function.

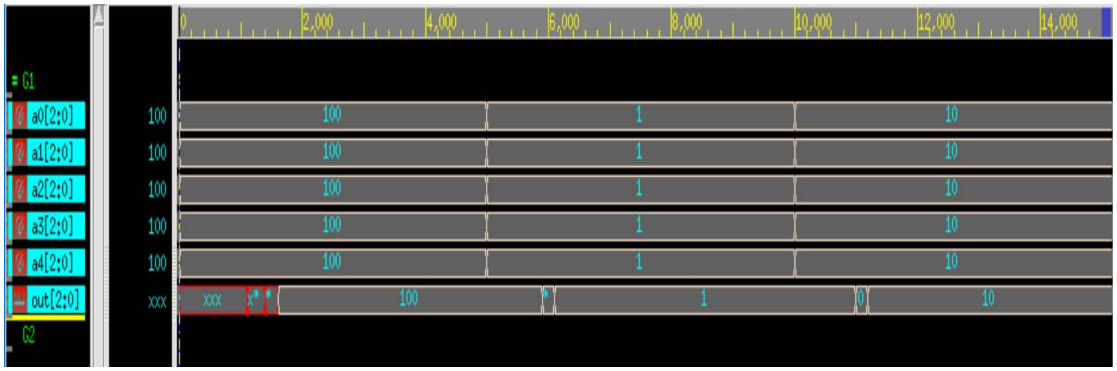


III. Test bench and verification

A. Voting Circuit

1. 5 Same Value

Timescale: 1ns/10ps
Unit Test bench delay: 50 time period



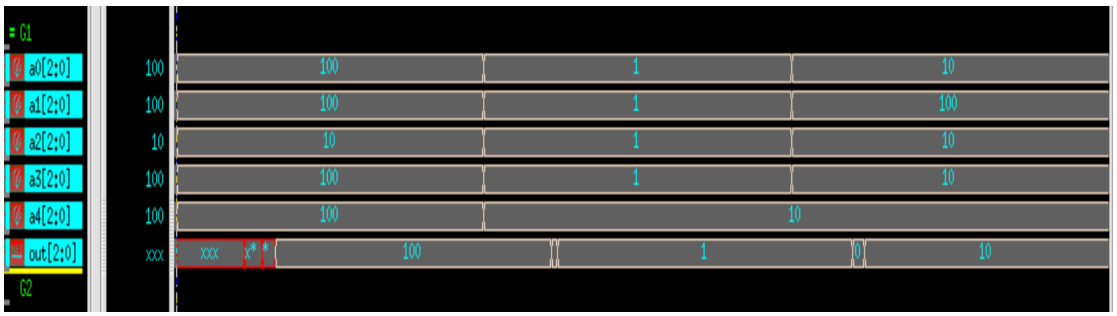
a. Case1: Correct
Input: a0=100; a1=100; a2=100; a3=100; a4=100;
Output: out=100

b. Case2: Correct
Input: a0=001; a1=001; a2=001; a3=001; a4=001;
Output: out=001

c. Case3: Correct
Input: a0=010; a1=010; a2=010; a3=010; a4=010;
Output: out=010

2. 4 Same Value

Timescale: 1ns/10ps
Unit Test bench delay: 50 time period



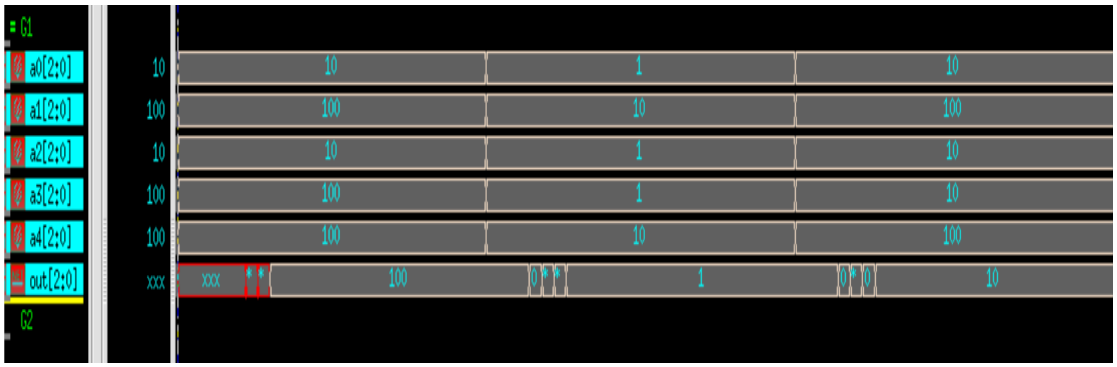
a. Case1: Correct
Input: a0=100; a1=100; a2=010; a3=100; a4=100;
Output: out=100

b. Case2: Correct
Input: a0=001; a1=001; a2=001; a3=001; a4=010;
Output: out=001

c. Case3: Correct
Input: a0=010; a1=100; a2=010; a3=010; a4=010;
Output: out=010

3. 3 Same Value and 2 Same Value

Timescale: 1ns/10ps
Unit Test bench delay: 50 time period



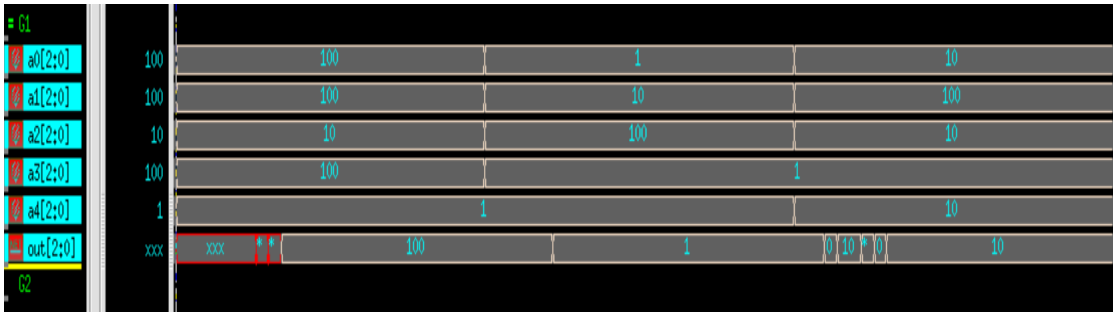
a. Case1: Correct
Input: a0=010; a1=100; a2=100; a3=010; a4=100;
Output: out=100

b. Case2: Correct
Input: a0=001; a1=010; a2=001; a3=001; a4=010;
Output: out=001

c. Case3: Correct
Input: a0=010; a1=100; a2=010; a3=010; a4=100;
Output: out=010

4. 3 Same Value and 2 Different Value

Timescale: 1ns/10ps
Unit Test bench delay: 50 time period



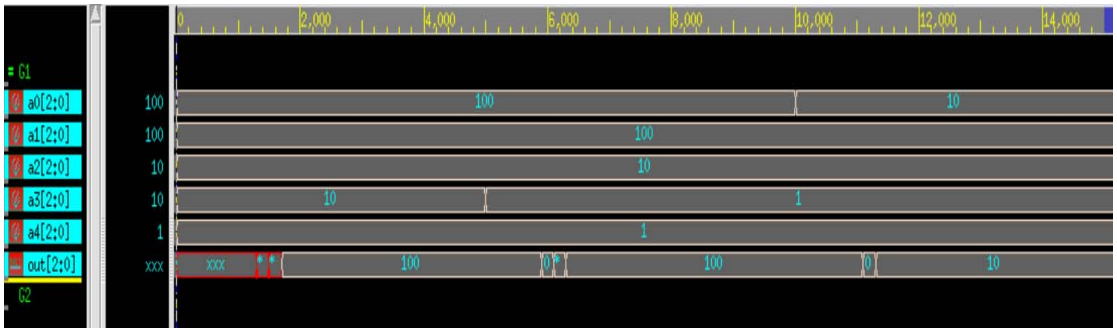
a. Case1: Correct
Input: a0=100; a1=100; a2=010; a3=100; a4=001;
Output: out=100

b. Case2: Correct
Input: a0=001; a1=010; a2=100; a3=001; a4=010;
Output: out=001

c. Case3: Correct
Input: a0=010; a1=100; a2=010; a3=001; a4=010;
Output: out=010

5. (2, 2) Same Value and 1 Different Value

Timescale: 1ns/10ps
Unit Test bench delay: 50 time period



a. Case1: Correct
Input: a0=100; a1=100; a2=010; a3=010; a4=001;
Output: out=100

b. Case2: Correct
Input: a0=a0=100; a1=100; a2=010; a3=001; a4=001;
Output: out=100

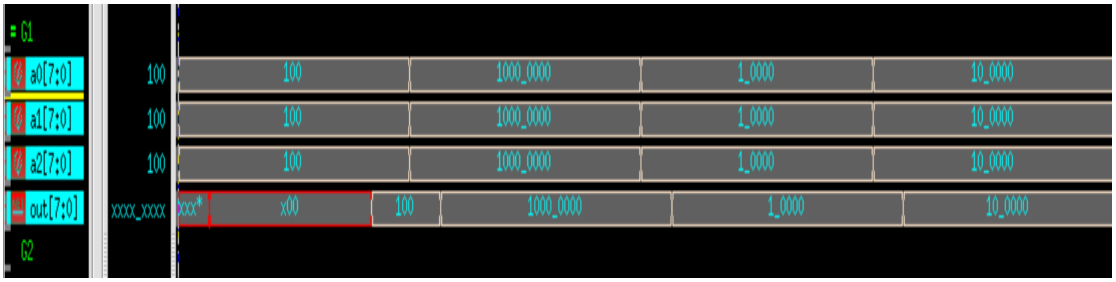
c. Case3: Correct

Input: a0=010; a1=100; a2=010; a3=001; a4=001;
Output: out=010

B. Median Circuit

1. 3 Same Value

Timescale: 1ns/10ps
Unit Test bench delay: 50 time period



a. Case1: Correct
Input: a0=8'b00000100;
a1=8'b00000100;
a2=8'b00000100;
Output: out=00000100

b. Case2: Correct
Input: a0=8'b10000000;
a1=8'b10000000;
a2=8'b10000000;
Output: out=10000000

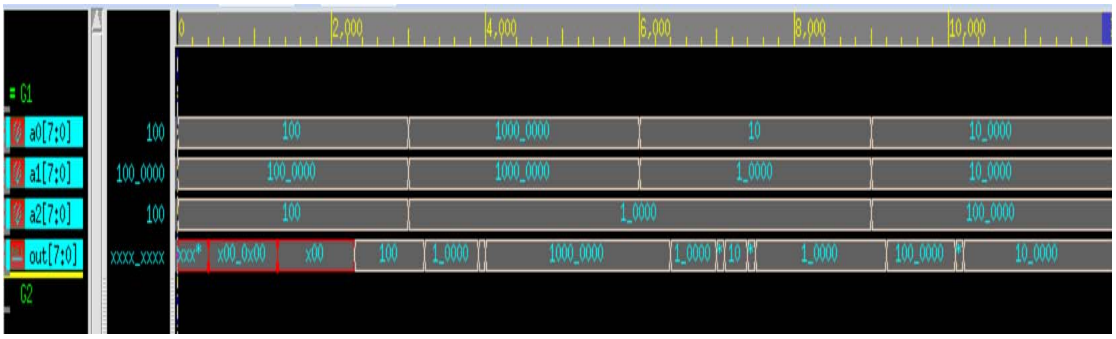
c. Case3: Correct
Input: a0=8'b00100000;
a1=8'b00100000;
a2=8'b00100000;
Output: out=00100000

d. Case4: Correct
Input: a0=8'b00100000;
a1=8'b00100000;
a2=8'b00100000;

Output: out=00100000

2. 2 Same Value

Timescale: 1ns/10ps
Unit Test bench delay: 50 time period



a. Case1: Correct
Input: a0=8'b00000100;
a1=8'b01000000;
a2=8'b00000100;
Output: out=00000100

b. Case2: Correct
Input: a0=8'b10000000;
a1=8'b10000000;
a2=8'b00010000;
Output: out=10000000

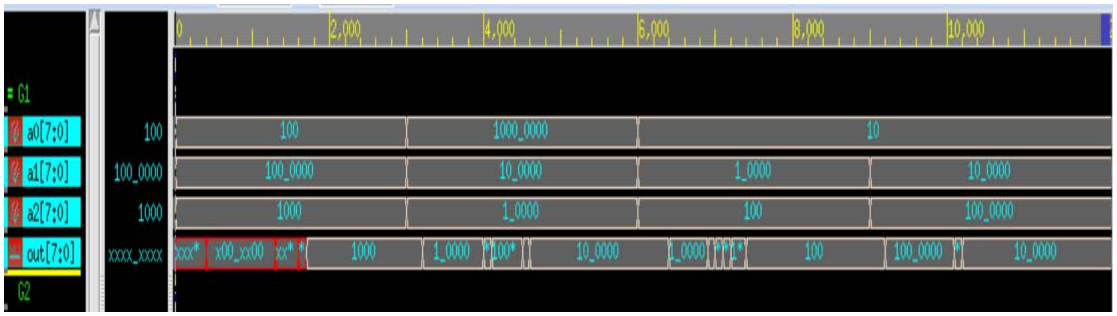
c. Case3: Correct
Input: a0=8'b00000010;
a1=8'b00100000;
a2=8'b00100000;
Output: out=00100000

d. Case4: Correct
Input: a0=8'b00100000;
a1=8'b00100000;
a2=8'b01000000;
Output: out=00100000

3. 3 Different Value

Timescale: 1ns/10ps

Unit Test bench delay: 50 time period



a. Case1: Correct
Input: a0=8'b00000100;
a1=8'b01000000;
a2=8'b00001000;
Output: out=00001000

b. Case2: Correct
Input: a0=8'b10000000;
a1=8'b00100000;
a2=8'b00010000

Output: out=10000000

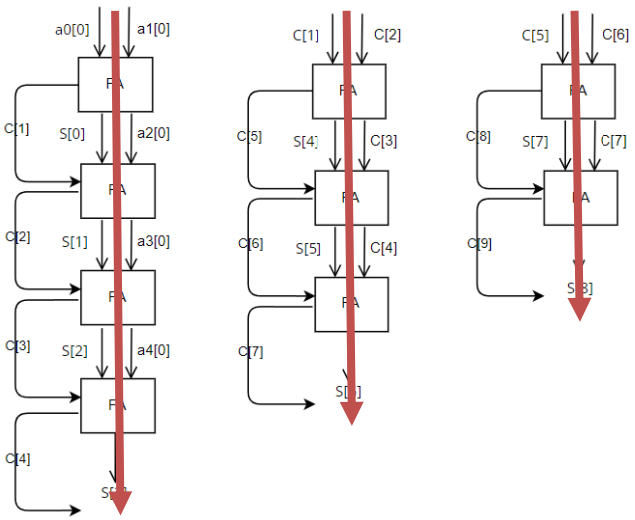
c. Case3: Correct
Input: a0=8'b00000010;
a1=8'b00010000;
a2=8'b00000100
Output: out=00100000

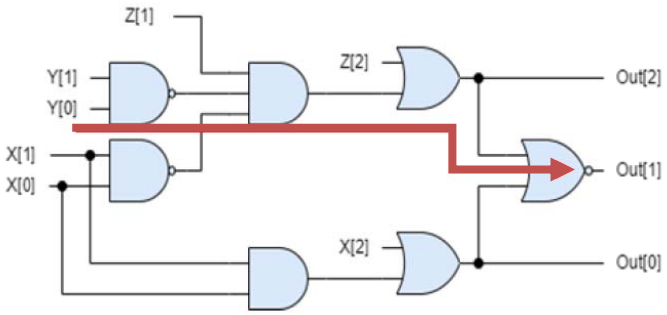
d. Case4: Correct
Input: a0=8'b00000010;
a1=8'b00100000;
a2=8'b01000000;
Output: out=00100000

IV. Circuit Analysis

A.Critical Path and Propagation Delay

1. Voting Circuit





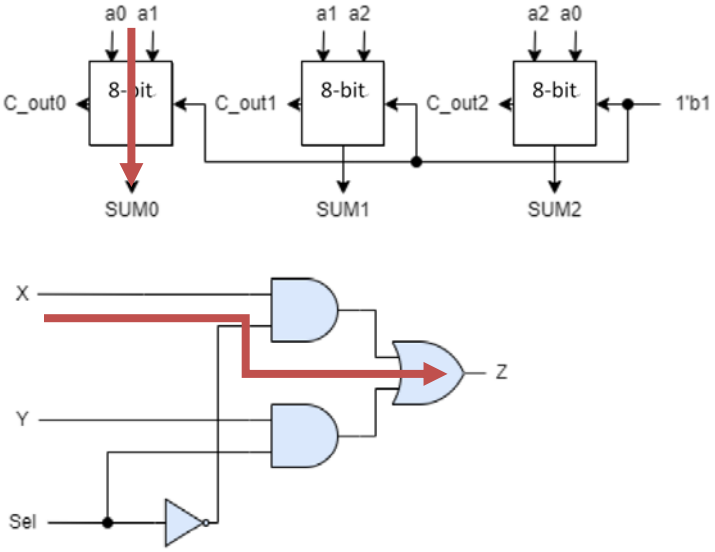
A 1-bit to 3-bit adder consists of 9 FA, and each of the FA has 10 Δ, so the delay of a 1-bit to 3-bit adder is 90 Δ.

After compiling the result of the three bits, the output results connect to a set of simplified logic gates. The critical path of the circuit passes an and gates, a or gates, a nor gate and a nor gate, that is, 2+2+2=8 Δ.

Hence a voting circuit here would have 98 Δ of delay.

In my testbench, the timescale is set as 1ns/10ps, so the maximum delay is approximately equal to 98*1=98ns.

2. Median Circuit



An 8-bit ripple carry adder consists of 8 FA, and each delay of a FA is 10Δ, so the delay of 8-bit ripple carry adder is 80Δ.

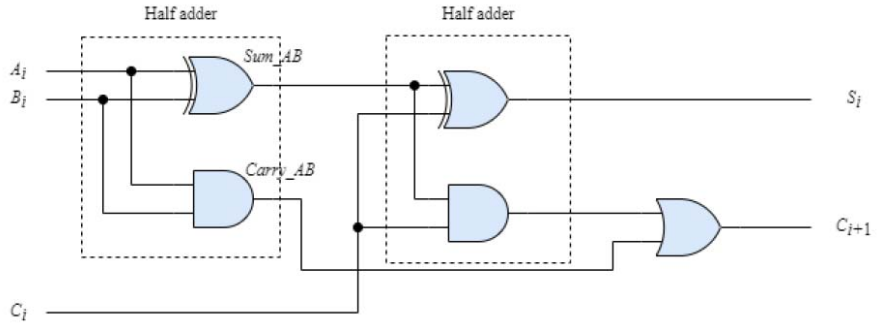
After addition, we go to the median selection logic gates. The critical selection circuit passes a nor gates, an and gate and a or gate, which is 1+2+2=5Δ.

As for a MUX, its critical path passes a not gate, an and gate and an or gate, so its delay is also 1+2+2=5Δ.

The total delay of this circuit is hence equal to 80+5+5=90Δ, and since the timescale of my testbench is 1ns/10ps, the maximum selay is approximately equal to 90*1=90ns.

B. Complexity (Logic Gates and Layers)

1. Voting Circuit



There are a total of 5 logic gates in a FA, and we have 9 FA consist in a 1-bit to 3-bit adder, so three 1-bit to 3-bit adder consist of 3*9=27 gates.

Additionally, there are 7 gates in a comparison circuit, hence there are a total of 27+7=34 gates.

2. Median Circuit

There are a total of 5 logic gates in a FA, and we have 8 FA consist in an 8-bit ripple carry adder, so three 8-bit ripple carry adder consist of 3*8*5=120 logic gates.

As for the median selection logic gate, we have 3 not gates, 4 and gates, and 2 and gates, which is a total of 9 logic gates.

Additionally, we have 4 gates in a single MUX, it means, we would have 4*2*8=64 gates of MUX in or circuit.

Hence, in a total, we have 120+9+64=193 gates.

V. Discussion

A.Problems encountered and solutions

When I'm testing the median circuit, I found that the input value from the monitor always show the different value from my inputs in testbench. After rechecking my code, I add a "8'b" ahead of my value (e.g. 00100000 to 8'b00100000) to insure my inputs won't be converted to decimal number and hence causes different value in binary form.

Apart from this, when deciding what bit to assign in MUX, I mixed up the order of inputs and causes a mixed up of my output, I rethink of the hardware structure again and recheck my code to

solve the problem.

As for the voting circuit design, it cost me a lot of time stoking in how to convert 5 1-bit inputs to an output of 3-bit. After several times of try and errors, I finally design a circuit that can implement this function.

Lastly, I write this homework with data flow modeling at first, but after checking the criteria of this assignment in cyber university, I changed my coding style into gate level modeling. Which should not but unfortunately causes errors to my output results. I solved the problem by following the instruction of the error messages, it shows that I still need to improve not only my grammar but also my ability of coding.

B. Reflection

The complexity of hw2 is much more complex than our first homework. It not only require the ability of converting hardware structure into verilog code but also require some ability of designing. Designing spent time and is always a difficult process, however, after successfully design a circuit, the sense of achievement is precious. Also, I'm grateful that we had already designed ripple carry adder in our previous homework, so when ripple carry adder is needed in my circuit, I only need to copy paste the code, which is very convenient and efficient. After this homework, I'll tried to separate different functions of circuit in different module more often in the future. The advantage of doing so is not only easier for debugging but also more convenient for future using.