National Sun Yat-sen University
Department of Electrical Engineering

112-1 Practical Digital System Design

HW1-1 Combinational 8-bit adder/subtractor

Name: 胡庭翊

Student ID: B103015006

I. Characteristic

A. Ripple Carry Adder

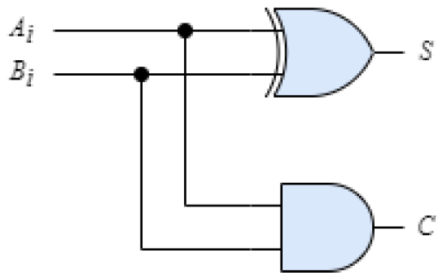
A ripple carry adder/subtractor is a digital circuit that can perform both addition and subtraction of binary numbers. It's called a "ripple carry" because the carry bit ripples through the individual full adders from the least significant bit (LSB) to the most significant bit (MSB).

1. Half-adder

A half adder is a basic digital circuit used in logic design to perform simple addition of two binary numbers.

It has two inputs and two outputs. The inputs are the two binary digits to be added, and the outputs are the sum and the carry.

A half adder can handle the addition of two bits, but it does not account for any carry from a previous addition. It's a fundamental building block for more complex adder circuits like the full adder, which can handle multi-bit addition and carry propagation.



Here, the two input , A and B, are 1-bit. And the output $S = A \oplus B; C = A \cdot B$.

$S=A \oplus B$

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

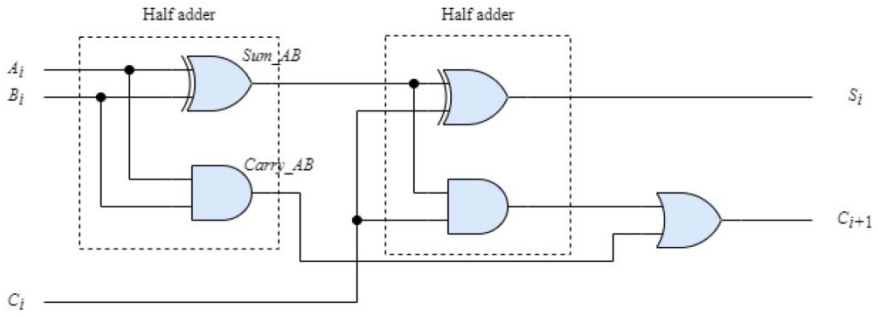
$C=A \cdot B$

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

2. Full-Adder

A full adder is a more complex digital circuit used in digital logic design to perform binary addition of three inputs: A, B, and an incoming carry (Ci). It has two outputs: the sum (S) and an outgoing carry (Ci+1).

A full adder can handle the addition of two binary digits along with a carry from a previous addition, making it suitable for multi-bit binary addition.

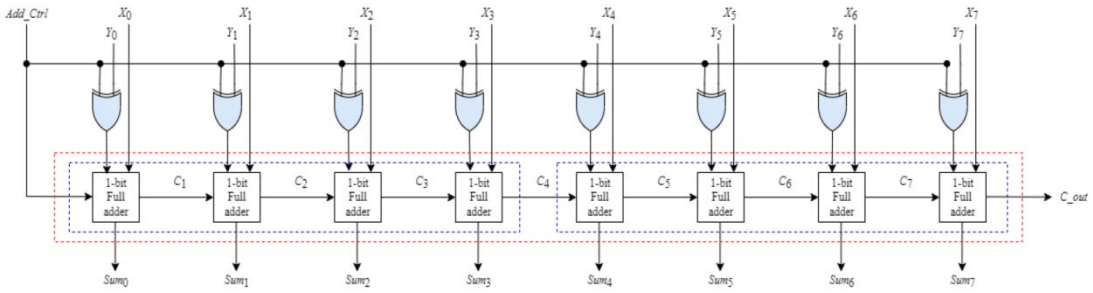


A full adder is consisted of two half adder, the output of the first adder will go to the second half adder and be operated with the previous carry (Ci).

3. 8-bit unsigned without overflow

An 8-bit adder/subtractor is built with eight Full-adders (FA) and a bit of Add_ctrl to conduct 8 bits addition and subtraction.

A Full-adder is made of two Half-adder, and here, we defined Add_ctrl=0 as addition and Add_ctrl=1 as subtraction.



Since $X - Y = X + (1's\ compliment\ of\ Y)$, so the adder/subtractor can adapt the characteristic of XOR gate: when Add_Ctrl = 1, Y will be the compliment of 1, and when

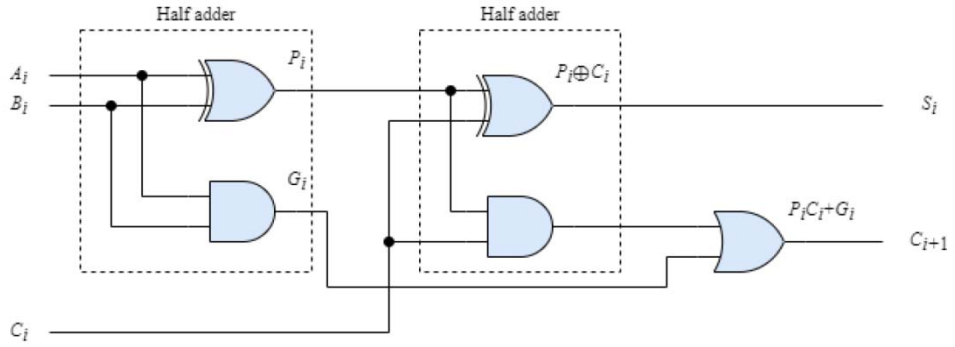
Add_Ctrl = 0, Y will not be changed.

Assign Add_Ctrl into Cin(C0) of the logic circuit, and the function of addition and subtraction will be completed.

B. Carry-lookahead adder

A carry-lookahead adder is a type of digital circuit used for binary addition in computer and digital logic design. It's designed to improve the speed of addition operations compared to the ripple carry adder.

1. Full-adder donates as PG



Adapting the concept of half-adder and full-adder, we now assign A, B as the input of the first half-adder, and substitute the output to P, G, we can get Carry propagate: $P_i = A_i \oplus B_i$, Carry generate: $G_i = A_i \cdot B_i$.

And the output result of the full=adder is $S_i = P_i \oplus C_i$, $C_{i+1} = P_i \cdot C_i + G_i$.

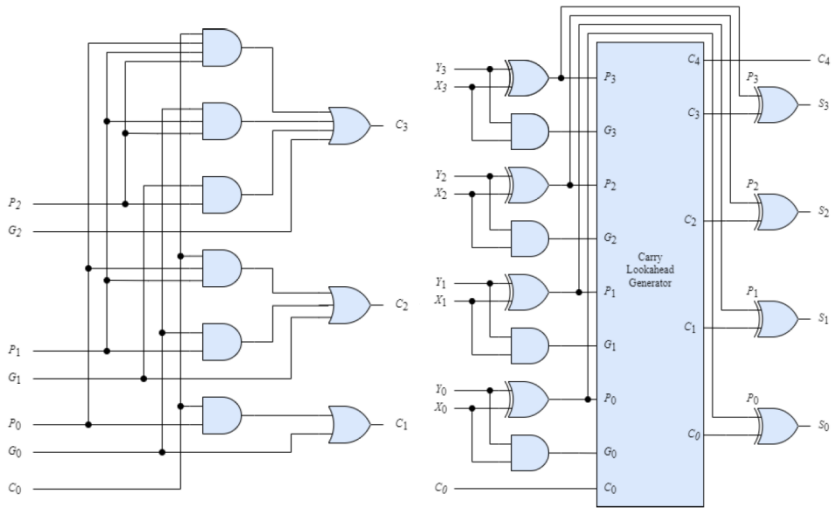
2. Carry-lookahead generator

Since $C_{i+1} = P_i \cdot C_i + G_i$, we can get :

$$C_1 = P_0 \cdot C_0 + G_0$$

$$C_2 = P_1 \cdot C_1 + G_1 = P_1 \cdot (G_0 + P_0 \cdot C_0) + G_1$$
$$= P_1G_0 + P_1P_0C_0 + G_1$$

$$C_3 = P_2 \cdot C_2 + G_2 = P_2 \cdot (G_1 + P_1 \cdot C_1) + G_2$$
$$= P_2G_1 + P_2P_1C_1 + G_2 = P_2G_1 + P_2P_1(G_0 + P_0 \cdot C_0) + G_2$$
$$= P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 + G_2$$

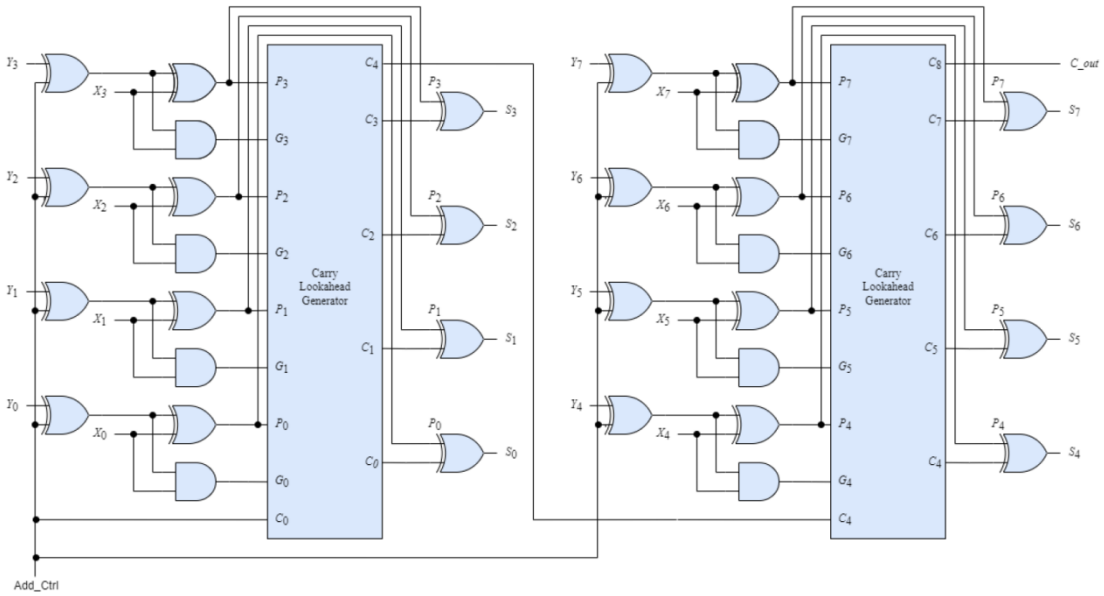


Following the same regular, we can convert these patterns into the above circuit, which is, a 4-bit carry-lookahead adder.

3. 8-bit unsigned without overflow

Here, we defined Add_ctrl=0 as addition and Add_ctrl=1 as subtraction.

Since $X - Y = X + (1's\ compliment\ of\ Y)$, so the adder/subtractor can adapt the characteristic of XOR gate: when Add_Ctrl = 1, Y will be the compliment of 1, and when Add_Ctrl = 0, Y will not be changed.



We can separate the 8-bit carry-lookahead adder/subtractor into three parts.

First is the PG Generator, which is on the left hand side of the adder/subtractor. The function of the PG Generator is that, it can convert the input X, Y into the needed value P and G.

Second is the carry-lookahead generator, the function inside this box has already been described above. It can convert the P and G into the needed carry(C).

Last but not least is the Sum Generator. According to the relationship of $S_i = P_i \oplus C_i$, we can generate the output sum by using XOR gates.

II. Comparison

A. Propagation Delay

1. Ripple Carry Adder

In a ripple carry adder, the carry bit ripples through the stages, meaning that the carry-out of one stage depends on the carry-in from the previous stage. As a result, there's a significant propagation delay, and the time it takes to complete the addition increases with the number of bits.

2. Carry-Lookahead adder

A carry-lookahead adder reduces propagation delay by computing carry bits in parallel. The carry generation and carry propagation are optimized, so the delay is largely constant and doesn't depend on the number of bits. This results in significantly faster addition operations.

B. Complexity

1. Ripple Carry Adder

Ripple carry adders are relatively simple in terms of the number of components required. However, for a large number of bits, they can become quite complex due to the long chain of carry bits.

2. Carry-Lookahead Adder

Carry-lookahead adders are more complex to design and require additional logic gates to implement the carry lookahead feature. However, this complexity remains manageable even for larger bit sizes.

C. Parallelism

1. Ripple Carry Adder

Ripple carry adders do not take full advantage of parallelism because carry propagation is serial, and each stage must wait for the carry-in from the previous stage.

2. Carry-Lookahead Adder

Carry-lookahead adders are highly parallel, as carry generation is independent for each stage. This allows them to take full advantage of parallel processing, making them faster for large additions.

D. Applications

1. Ripple Carry Adder

Ripple carry adders are suitable for applications where speed is not a critical factor or for simple adder/subtractor circuits.

2. Carry-Lookahead Adder

Carry-lookahead adders are preferred in applications where speed is crucial, such as in high-performance processors, arithmetic logic units (ALUs), and any system requiring rapid addition operations.

III. Design Principle

A. Gate level modeling

In gate-level modeling, the design is expressed in terms of logic gates, such as AND gates, OR gates, and flip-flops, and their interconnections. Each gate-level model corresponds to a physical electronic gate or flip-flop in the actual hardware.

In our first homework, we are requested to use gate-level modeling as our coding style.

B. Ripple Carry Adder

First, by using top-down design we defined three modules: half_adder, full_adder, and adder.

The function of half adder and full adder is simple, and after combining these two modules, we can get a full adder, which is already very close to our target.

After that, since we are making 8-bit unsigned ripple carry adder without overflow, we should define input and output of 8 bits. Nevertheless, using hardware oriented design, we know that we should also define our wires.

I give my input ports as A and B, and set one bit of input mode to design whether my digital circuit is an adder or a subtractor by connection it with B to a XOR gate.

The addition/subtraction result is named as sum with 8 bits, and there is also an output called c_out with 1 bit, which is supposed to be the last bit of the carry of the adder.

C. Carry-Lookahead Adder

First, I define a PG Generator module in order to convert the input A, B into variable P and G, which is easier for the latter compiling.

After that, since we are making 8-bit unsigned carry-lookahead adder without overflow, I here define input and

output of 8 bits, one bit of mode in order to decide whether the digital circuit is an adder or a subtractor, and wires connected to AND gates inside the adder/subtractor.

I gave name to my input ports as A and B, output sum as sum, and the last bit of carry as c_out.

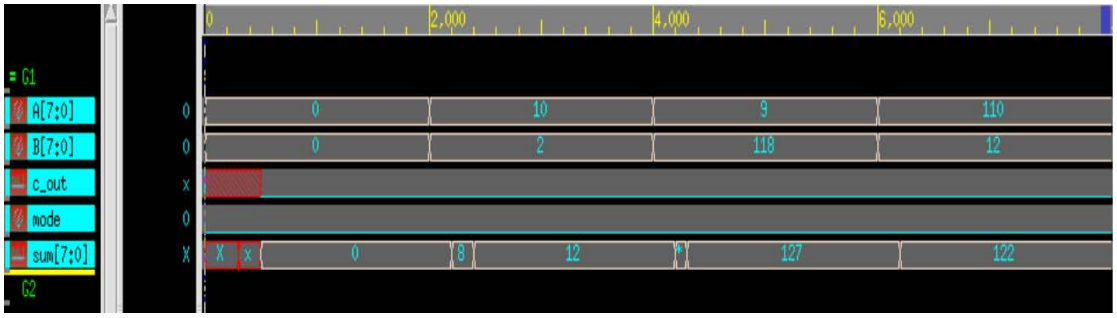
By the principle of carry-lookahead adder mentioned above, we can simply connect the wires and gates, and conduct a carry-lookahead module.

IV. Result

A. Ripple Carry Adder

1. Addition

Timescale: 1ns/10ps
Gate delay: 1 time period
Test bench delay: 20 time period



a. Case1: Correct
Input: A = 0, B = 0 (inside the range of [-127, 127])
mode = 0 (Addition)
Output: c_out = 0 (no carry), sum = 0

b. Case2: Correct
Input: A = 10, B = 2 (inside the range of [-127, 127])
mode = 0 (Addition)
Output: c_out = 0 (no carry), sum = 12

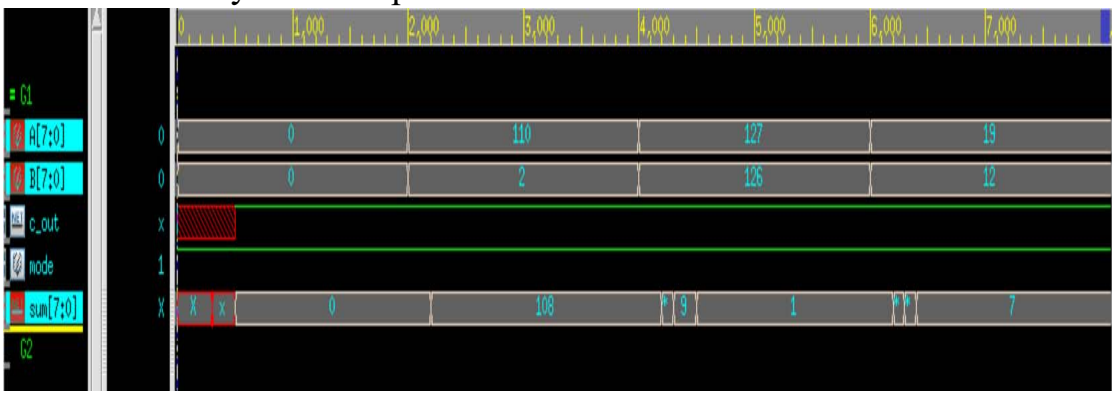
c. Case3: Correct
Input: A = 9, B = 118 (inside the range of [-127, 127])
mode = 0 (Addition)

Output: c_out = 0 (no carry), sum = 127

d. Case2: Correct
Input: A = 110, B = 12 (inside the range of [-127, 127])
mode = 0 (Addition)
Output: c_out = 0 (no carry), sum = 122

2. Subtraction

Timescale: 1ns/10ps
Gate delay: 1 time period
Test bench delay: 20 time period



a. Case1: Correct
Input: A = 0, B = 0 (inside the range of [-127, 127])
mode = 1 (Subtraction)
Output: c_out = 1(has carry), sum = 0

b. Case2: Correct
Input: A = 110, B = 2 (inside the range of [-127, 127])
mode = 1 (Subtraction)
Output: c_out = 1(has carry), sum = 108

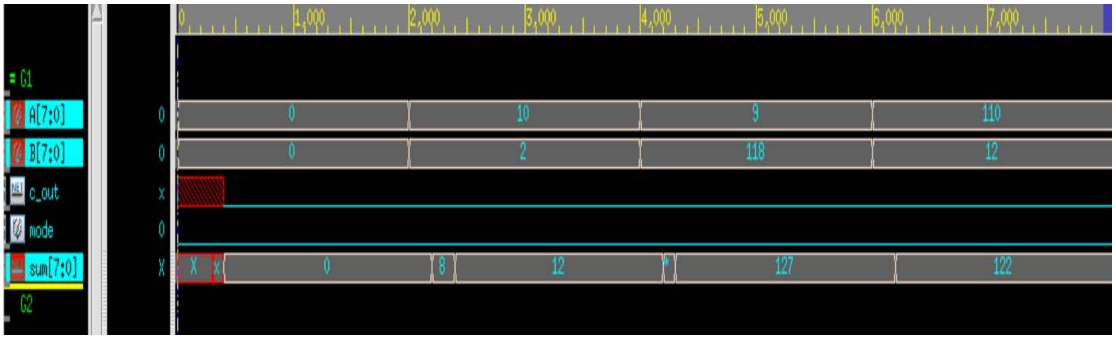
c. Case3: Correct
Input: A = 127, B = 126 (inside the range of [-127, 127])
mode = 1 (Subtraction)
Output: c_out = 1(has carry), sum = 1

d. Case2: Correct
Input: A = 19, B = 12 (inside the range of [-127, 127])
mode = 1 (Subtraction)
Output: c_out = 1(has carry), sum = 7

B. Carry-Lookahead adder/subtractor

1. Addition

Timescale: 1ns/10ps
Gate delay: 1 time period
Test bench delay: 20 time period



a. Case1: Correct
Input: A = 0, B = 0 (inside the range of [-127, 127])
mode = 0 (Addition)
Output: c_out = 0 (no carry), sum = 0

b. Case2: Correct
Input: A = 10, B = 2 (inside the range of [-127, 127])
mode = 0 (Addition)
Output: c_out = 0 (no carry), sum = 12

c. Case3: Correct
Input: A = 9, B = 118 (inside the range of [-127, 127])
mode = 0 (Addition)
Output: c_out = 0 (no carry), sum = 127

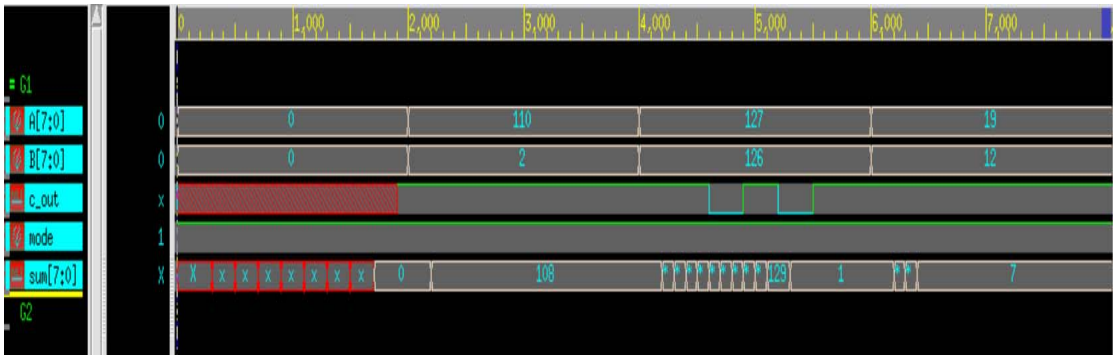
d. Case2: Correct
Input: A = 110, B = 12 (inside the range of [-127, 127])

mode = 0 (Addition)

Output: c_out = 0 (no carry), sum = 122

2. Subtraction

Timescale: 1ns/10ps
Gate delay: 1 time period
Test bench delay: 20 time period



a. Case1: Correct
Input: A = 0, B = 0 (inside the range of [-127, 127])
mode = 1 (Subtraction)
Output: c_out = 1(has carry), sum = 0

b. Case2: Correct
Input: A = 110, B = 2 (inside the range of [-127, 127])
mode = 1 (Subtraction)
Output: c_out = 1(has carry), sum = 108

c. Case3: Correct
Input: A = 127, B = 126 (inside the range of [-127, 127])
mode = 1 (Subtraction)
Output: c_out = 1(has carry), sum = 1

d. Case2: Correct
Input: A = 19, B = 12 (inside the range of [-127, 127])
mode = 1 (Subtraction)
Output: c_out = 1(has carry), sum = 7

II. Circuit Analysis

A. Critical path and the maximum delay

According to the circuit diagram, we know that a 8-bit ripple carry adder/subtractor is consists of 8 FA and mode control XOR gate.

Delay of each a HA is 1, and the delay of a FA module is 3, which is, the delay of connecting the carry to the next stage FA is 2. Hence the total delay inside the ripple carry adder is $3 + 2 \times 7 = 17$. However, we still have a XOR gate to control whether the circuit is an adder or a subtractor, so lastly the longest path delay is 18.

Since the timescale of the test bench is designed as 1ns/10ps, and the delay of each gates #20, so we know that the delay of my ripple carry adder is $18 \times 20 \times 1 = 360\text{ns}$.

On the other hand, the delay of carry-lookahead adder/subtractor is 3 no matter how many bits is designed today. Additionally we have a XOR gate to control whether the circuit is an adder or subtractor, hence the longest path delay is 4.

Since the timescale of the test bench is designed as 1ns/10ps, and the delay of each gates #20, so we know that the delay of my ripple carry adder is $4 \times 20 \times 1 = 80\text{ns}$.

B. Number of logic gates and layers

An 8-bit ripple carry adder/subtractor is consists of 8 FA and 8 XOR gates, inside of the FA, there are 5 logic gates, thus we need $8 \times 5 + 8 = 48$ logic gates, and the layer of RCA is $3 \times 8 + 1 = 25$. (3 layers in each FA plus one XOR gate).

An 8-bit carry-lookahead adder/subtractor needs 52 logic gates according to its circuit diagram, and adding the XOR gates outside the CLA, we have $52 + 8 = 60$ logic gates, and the layer of CLA is 5.

Compare these 2 adder/subtractor, we can tell that the delay of CLA is smaller , however it has a larger area.

III. Discussion

A. Problems and Solutions

The problem I encountered is that, I found that after adding gate delays to my module, the output waveform of nWave is not as accurate as previous result (not adding gate delay). The number is strange and doesn't give the correct answer).

After discussing with my classmates, I noticed that we need some time for the gate to compile after we added the gate delay, so we cannot get the correct answer immediately. Hence, the impact to my design is that, I redesign my test bench and give a larger waiting time for my circuit to compile, after that, I get a normal output result and waveform.

B. Reflection

This is our first homework of PDSD, and the professor said that the homework is simple; however, I think our first homework is enough complicated. I can said that I had learned some sense of verilog coding and noticed the way of using the work station, but the loading of our homework is too huge, I spent a whole weekend and still can't finish my homework… not to mention my other subjects' midterm exam. I can't imagine how it would be like of our following homework and final project.