National Cheng Kung University

Department of Electrical Engineering

Introduction to VLSI CAD

Lab Session 2

Design and Simulation of K-map simplifier, Multiplier and Ripple Carry Adder

Objectives:

Help students get familiar with the CAD tools (NC-Verilog & Verdi) for digital logic design. Please go through the hands-on exercise step-by-step.

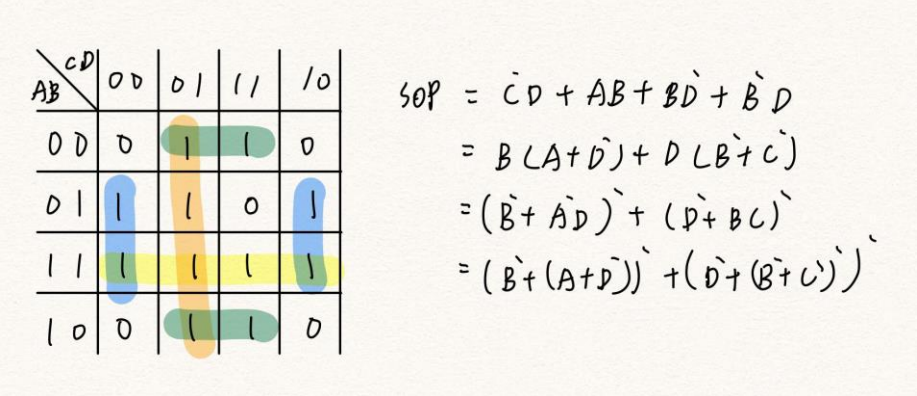
Prob A: Design Steps

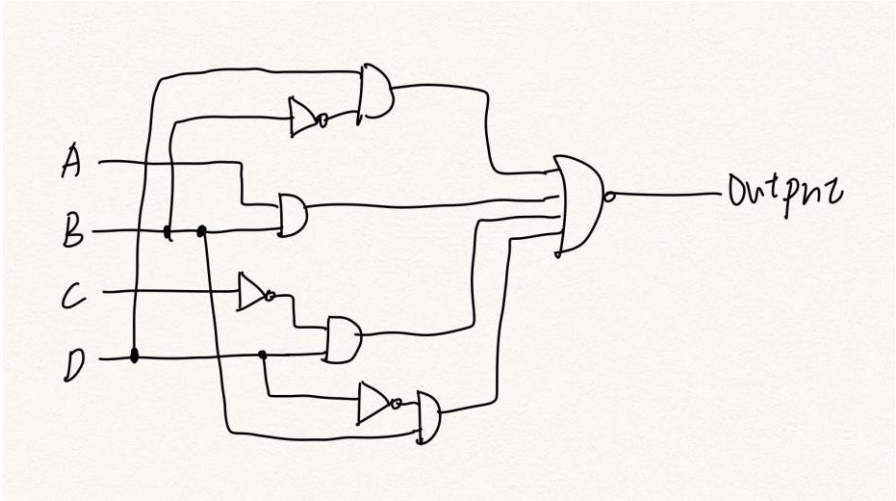
---

1) **Understand the K-map method and gate level.**

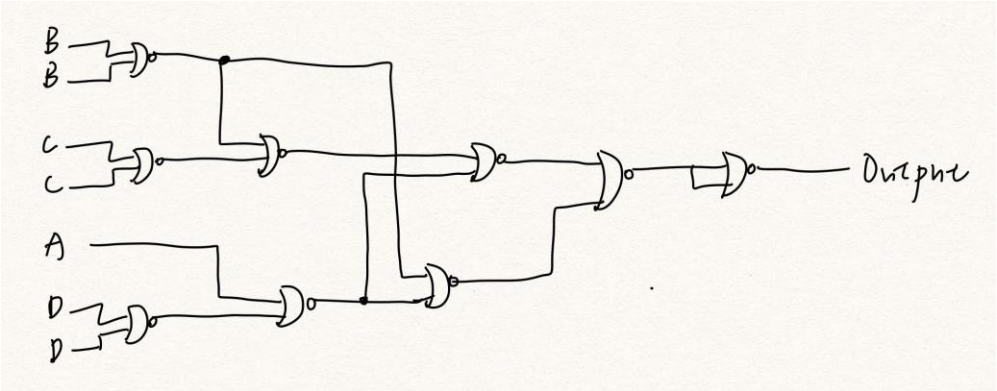
   Exercise 1-1: Simplifying the giving truth table using K-map method(Please describe/draw your procedure clearly). Draw a Schematic View of your simplified circuit (gate-level diagram)

A	B	C	D	Out
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1





2) **Design your circuit(simplified by K-map method) with only NOR gate in it.**



3) **Implement your simplified circuit based on the gate-level diagram you drawn in 2) by verilog code. (The module name should be kmap)**
   ➔ You only need to upload your v-code file to moodle, **do not** paste your code here.

4) **Read the testbench for a kmap you implemented in 4). (The module is named kmap_tb)**
   ➔ You only need to upload your v-code to moodle, **do not** paste your code here.

5) **Compile the code you implemented in 4) using NC-Verilog. If there are errors, please go back to 3) to debug your code and re-compile again.**
   (Hint : The command for compiling is %ncverilog kmap.v)

6) **Simulate your design in 3) with the testbench in 4).**
   (Hint: The command for simulation is
   %ncverilog kmap_tb.v kmap.v +access+r +define+FSDB)

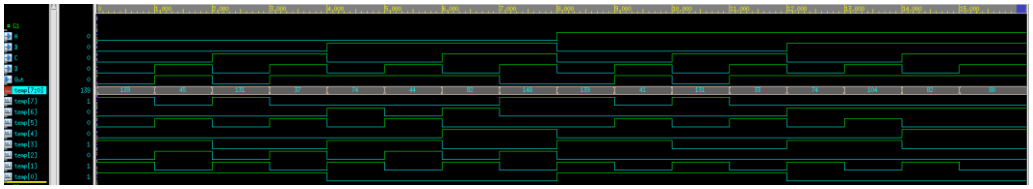
7) **Verify your design by comparing the simulation results with the results you predicted. If the results are not the same, please go back to 3) and revise your code. If the simulation results are correct, please snapshot the simulation result on the terminal and the waveform you dumped and *explain your waveform*. (Hint : The command to open nWave is %nWave &)**

In addition, you should check your coding style, there are no error messages and over 90% coverage with Superlint. Snapshot the result and *calculate Superlint coverage*. (Hint: The command to open Superlint is %jg -superlint superlint.tcl)

Your simulation result on the terminal.

```
Verdi* : End of traversing.
                    0ns, A=0, B=0, C=0 , D=0 , Out=0
                   10ns, A=0, B=0, C=0 , D=1 , Out=1
                   20ns, A=0, B=0, C=1 , D=0 , Out=0
                   30ns, A=0, B=0, C=1 , D=1 , Out=1
                   40ns, A=0, B=1, C=0 , D=0 , Out=1
                   50ns, A=0, B=1, C=0 , D=1 , Out=1
                   60ns, A=0, B=1, C=1 , D=0 , Out=1
                   70ns, A=0, B=1, C=1 , D=1 , Out=0
                   80ns, A=1, B=0, C=0 , D=0 , Out=0
                   90ns, A=1, B=0, C=0 , D=1 , Out=1
                  100ns, A=1, B=0, C=1 , D=0 , Out=0
                  110ns, A=1, B=0, C=1 , D=1 , Out=1
                  120ns, A=1, B=1, C=0 , D=0 , Out=1
                  130ns, A=1, B=1, C=0 , D=1 , Out=1
                  140ns, A=1, B=1, C=1 , D=0 , Out=1
                  150ns, A=1, B=1, C=1 , D=1 , Out=1
Simulation complete via $finish(1) at time 160 NS + 0
./kmap_tb.v:41   #`INTERVAL $finish;
```

Your waveform :

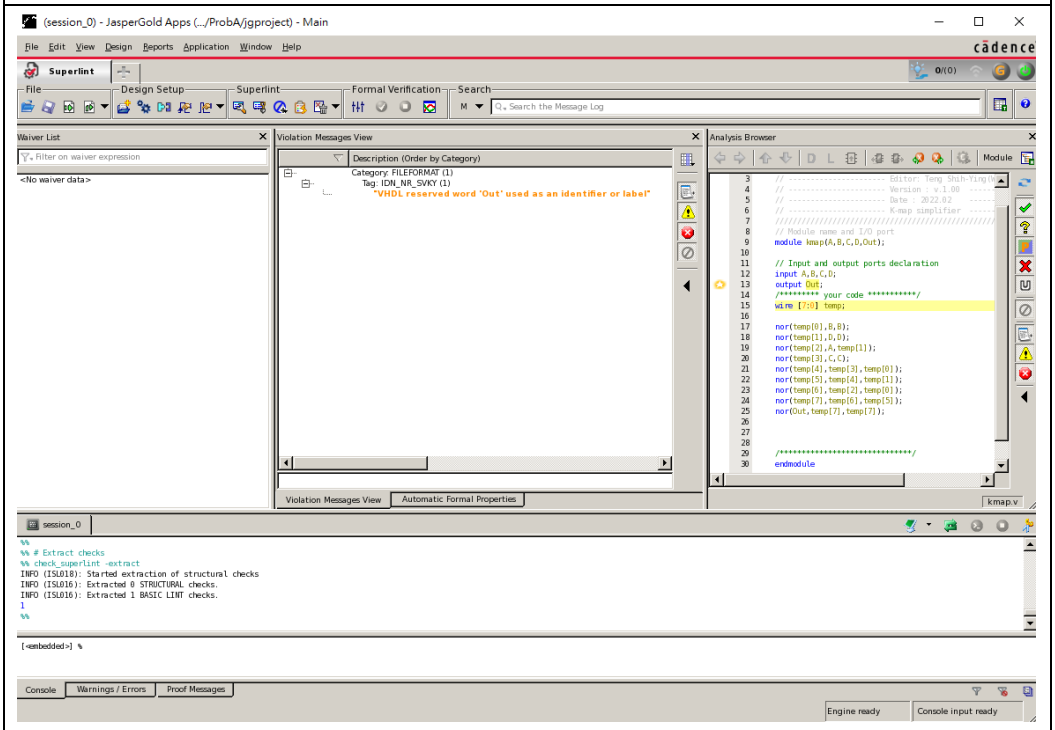


Explanation of your waveform :

Temp[0] is used to store B', Temp[1] is used to store D', Temp[2] is used to store (A+D')', Temp[3] is used to store C', Temp[4] is used to store (C'+B')', Temp[5] is used to store (D'+(C'+B')')', Temp[6] is used to store (B'+(A+D')')', Temp[7] is used to XOR [5] and [6]. Finally, two [7] values are XORed and used as a NOT operation. Removing the outermost NOT gives the final answer.

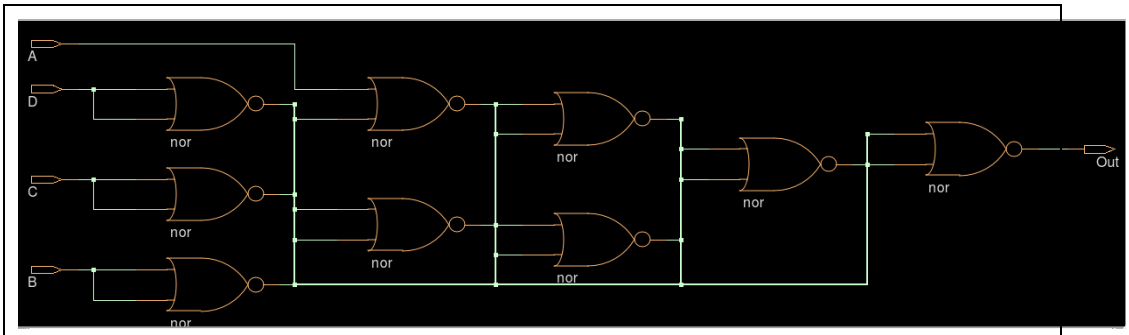
It can be seen that at the first time point, since ABCD are all 0, temp[0][1][3] are all 1. Temp[2] is (0+1)' = 0, temp[6] is (1+0)' = 0. By analogy, using the initial values, the final answer can be obtained.

Superlint screenshot and coverage



Coverage = 1

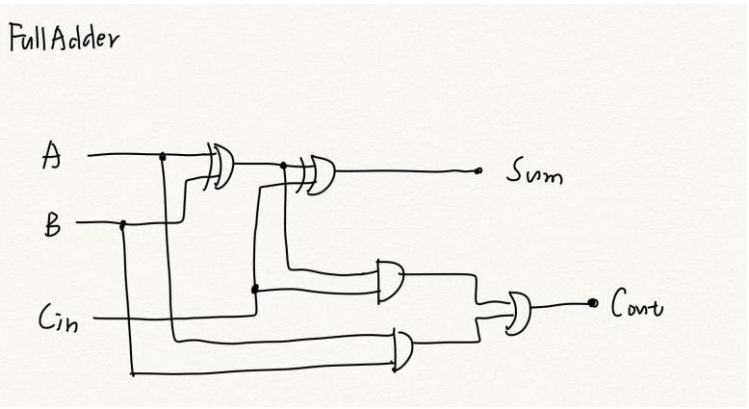
8) snapshot the schematic view of your design via *nSchema* (Hint : the command is $Debussy)



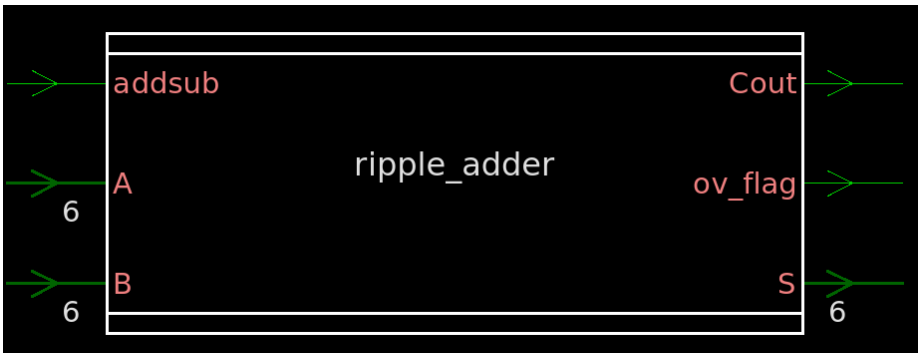
Prob B: hieratical coding

1) **An adder is a digital circuit that performs addition of number. Please design a full adder in gate level.**
2) **Draw a full adder in gate level.**

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



3) **Design a full adder in Structral coding (The module name should be FullAdder. And the file you include should be FullAdder.v)**
4) You only need to upload your v-code to moodle, **do not** paste your code here.
5) **Design a 6-bit add/sub ripple carry adder in hierarchical coding.**



Signal	Type	Bits	Description
A	Input	6	First operand
B	Input	6	Second operand

addsub	Input	1	If addsub is 0, operator is addition. If addsub is 1, operator is subtraction.
S	Output	6	Result after calculating
Cout	Output	1	Carry bit
ov_flag	Output	1	If result overflows, ov_flag pull to 1. Otherwise, ov_flag is 0.

6) **Your design should be named as ripple_adder.v** and module names in the file should be the same as file name (**ripple_adder**).

7) **Simulate your design in 2) with the following test pattern in sample testbench.**
(Hint: The command for compiling is %ncverilog ripple_adder.v)
(Hint: The command for simulation is %ncverilog ripple_adder_tb.v ripple_adder.v +define+FSDB +access+r)

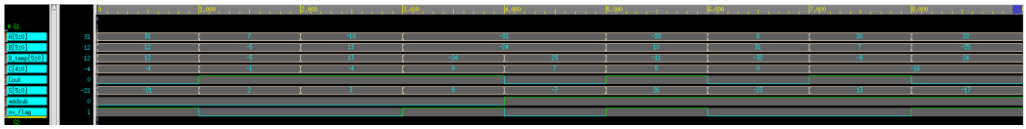
   I. **A = 31 ; B = 12 ; addsub = 0;**
   II. **A = 7 ; B = -5; addsub = 0;**
   III. **A = -10 ; B = 13 ; addsub = 0;**
   IV. **A = -31; B = -24 ; addsub = 0;**
   V. **A = -31; B = -24 ; addsub = 1;**
   VI. **A = -28; B = 10 ; addsub = 1;**
   VII. **A = 8; B = 31; addsub = 1;**
   VIII. **A = 20 ; B = 7 ; addsub = 1;**
   IX. **A = 22 ; B = -25 ; addsub = 1;**

8) **Verify your design by comparing the simulation results with the results you predicted. If the results are not the same, please go back to 2) and revise your code. If the simulation results are correct, please snapshot the simulation result on the terminal and the waveform you dumped and explain your waveform.**
(Hint : The command to open nWave is %nWave &)
In addition, you should check your coding style, there are no error messages and over 90% coverage with Superlint. Snapshot the result and *calculate Superlint coverage*. (Hint: The command to open Superlint is %jg -superlint superlint.tcl)

Your simulation result on the terminal.

```
*Verdi* : End of traversing.
                   0ns, A=011111, B=001100, addsub=0, S=43, Cout=0, ov_flag=1
                  10ns, A=000111, B=111011, addsub=0, S= 2, Cout=1, ov_flag=0
                  20ns, A=110110, B=001101, addsub=0, S= 3, Cout=1, ov_flag=0
                  30ns, A=100001, B=101000, addsub=0, S= 9, Cout=1, ov_flag=1
                  40ns, A=100001, B=101000, addsub=1, S=57, Cout=0, ov_flag=0
                  50ns, A=100100, B=001010, addsub=1, S=26, Cout=1, ov_flag=1
                  60ns, A=001000, B=011111, addsub=1, S=41, Cout=0, ov_flag=0
                  70ns, A=010100, B=000111, addsub=1, S=13, Cout=1, ov_flag=0
                  80ns, A=010110, B=100111, addsub=1, S=47, Cout=0, ov_flag=1
Simulation complete via $finish(1) at time 90 NS + 0
```

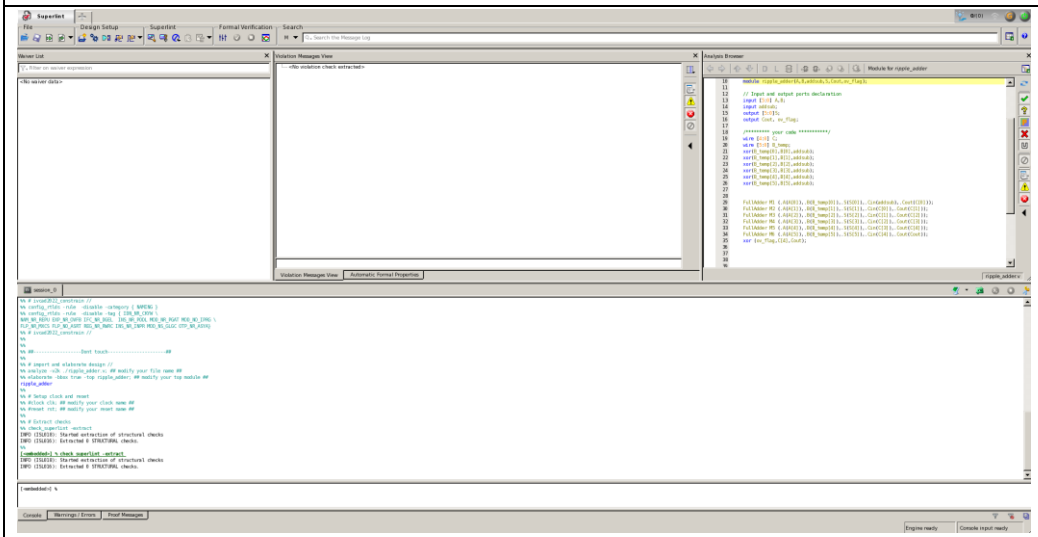
Your waveform :



Explanation of your waveform :

When between 0 and 1, since addsub = 0, addition is performed. The value of B is placed into B_temp without any operation. Then, FullAdders are used for addition. Finally, ov_flag is calculated by XORing the Cout of the last two bits to determine if there is an overflow. When between 4 and 5, since addsub = 1, subtraction is performed. The value of B is converted to a negative number using 1's complement and stored in B_temp for calculation. Also, Cin = 1 in the first FA, which can be like 2's complement form to get the correct answer. The overflow judgment method is the same as above, and so on.

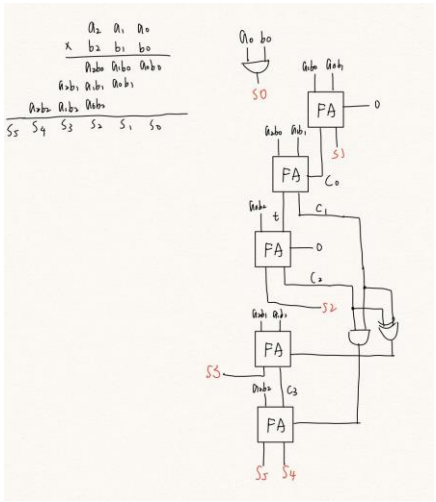
Superlint screenshot and coverage



Coverage = 1

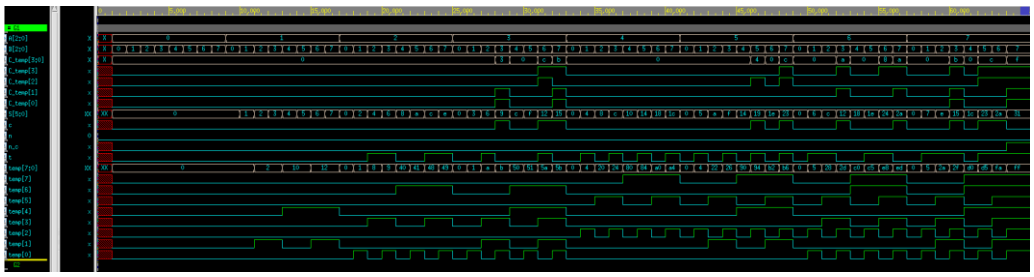
Prob C: Application

1) **Design a 3x3 bits multiplier using RCAs of Prob B, also designed at gate level rather than behavior level. (The module name should be multi. And the file you include should be multi.v)**
2) **Draw your hieratical structure.**
3) **You only need to upload your v-code to moodle, do not paste your code here.**
4) **Simulate your design in 2) with the testbench which includes all case of input. (Hint: The command for compiling is %ncverilog multi.v)**
   (Hint: The command for simulation is %ncverilog multi _tb.v multi.v +define+FSDB +access+r)
9) **Verify your design by comparing the simulation results with the results you predicted. If the results are not the same, please go back to 2) and revise your code. If the simulation results are correct, please snapshot the simulation result on the terminal and the waveform you dumped and explain your waveform. (Hint : The command to open nWave is %nWave &)**
   In addition, you should check your coding style, there are no error messages and over 90% coverage with Superlint. Snapshot the result and *calculate Superlint coverage*. (Hint: The command to open Superlint is %jg -superlint superlint.tcl)

Draw your hieratical structure.

Your simulation result on the terminal.

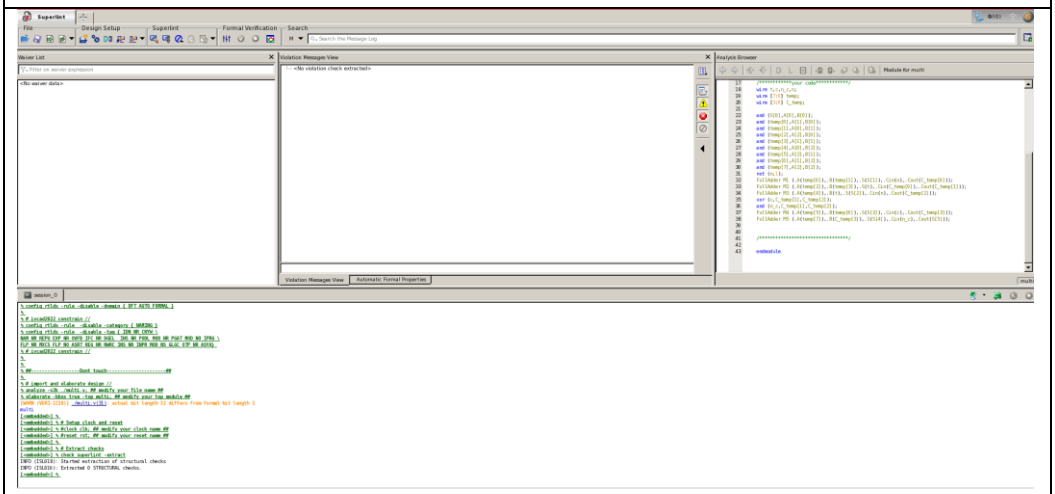
```
70ns, A=0, B=6, S in decimal= 0, S in binary=000000
80ns, A=0, B=7, S in decimal= 0, S in binary=000000
90ns, A=1, B=0, S in decimal= 0, S in binary=000000
00ns, A=1, B=1, S in decimal= 1, S in binary=000001
10ns, A=1, B=2, S in decimal= 2, S in binary=000010
20ns, A=1, B=3, S in decimal= 3, S in binary=000011
30ns, A=1, B=4, S in decimal= 4, S in binary=000100
40ns, A=1, B=5, S in decimal= 5, S in binary=000101
50ns, A=1, B=6, S in decimal= 6, S in binary=000110
60ns, A=1, B=7, S in decimal= 7, S in binary=000111
70ns, A=2, B=0, S in decimal= 0, S in binary=000000
80ns, A=2, B=1, S in decimal= 2, S in binary=000010
90ns, A=2, B=2, S in decimal= 4, S in binary=000100
00ns, A=2, B=3, S in decimal= 6, S in binary=000110
10ns, A=2, B=4, S in decimal= 8, S in binary=001000
20ns, A=2, B=5, S in decimal=10, S in binary=001010
30ns, A=2, B=6, S in decimal=12, S in binary=001100
40ns, A=2, B=7, S in decimal=14, S in binary=001110
50ns, A=3, B=0, S in decimal= 0, S in binary=000000
60ns, A=3, B=1, S in decimal= 3, S in binary=000011
70ns, A=3, B=2, S in decimal= 6, S in binary=000110
80ns, A=3, B=3, S in decimal= 9, S in binary=001001
90ns, A=3, B=4, S in decimal=12, S in binary=001100
00ns, A=3, B=5, S in decimal=15, S in binary=001111
10ns, A=3, B=6, S in decimal=18, S in binary=010010
20ns, A=3, B=7, S in decimal=21, S in binary=010101
30ns, A=4, B=0, S in decimal= 0, S in binary=000000
40ns, A=4, B=1, S in decimal= 4, S in binary=000100
50ns, A=4, B=2, S in decimal= 8, S in binary=001000
60ns, A=4, B=3, S in decimal=12, S in binary=001100
70ns, A=4, B=4, S in decimal=16, S in binary=010000
80ns, A=4, B=5, S in decimal=20, S in binary=010100
90ns, A=4, B=6, S in decimal=24, S in binary=011000
00ns, A=4, B=7, S in decimal=28, S in binary=011100
10ns, A=5, B=0, S in decimal= 0, S in binary=000000
20ns, A=5, B=1, S in decimal= 5, S in binary=000101
30ns, A=5, B=2, S in decimal=10, S in binary=001010
40ns, A=5, B=3, S in decimal=15, S in binary=001111
50ns, A=5, B=4, S in decimal=20, S in binary=010100
60ns, A=5, B=5, S in decimal=25, S in binary=011001
70ns, A=5, B=6, S in decimal=30, S in binary=011110
80ns, A=5, B=7, S in decimal=35, S in binary=100011
90ns, A=6, B=0, S in decimal= 0, S in binary=000000
00ns, A=6, B=1, S in decimal= 6, S in binary=000110
10ns, A=6, B=2, S in decimal=12, S in binary=001100
20ns, A=6, B=3, S in decimal=18, S in binary=010010
30ns, A=6, B=4, S in decimal=24, S in binary=011000
40ns, A=6, B=5, S in decimal=30, S in binary=011110
50ns, A=6, B=6, S in decimal=36, S in binary=100100
60ns, A=6, B=7, S in decimal=42, S in binary=101010
70ns, A=7, B=0, S in decimal= 0, S in binary=000000
80ns, A=7, B=1, S in decimal= 7, S in binary=000111
90ns, A=7, B=2, S in decimal=14, S in binary=001110
00ns, A=7, B=3, S in decimal=21, S in binary=010101
10ns, A=7, B=4, S in decimal=28, S in binary=011100
20ns, A=7, B=5, S in decimal=35, S in binary=100011
30ns, A=7, B=6, S in decimal=42, S in binary=101010
40ns, A=7, B=7, S in decimal=49, S in binary=110001
```

Your waveform :



Explanation of your waveform :

It can be seen from the hierarchical structure above what values are stored in C_temp and temp, respectively. It can be seen that at 3*6, which is 011*110, there is a carry when operating S[2], so there will also be a carry when operating S[3]. Among them, c and n_c are used when operating S[2] because the three temps are added together. Therefore, it is possible that C_temp[1]

and [2] are both 1, which will directly enter the next two bits. Therefore, n_c = and(temp[1],temp[2]) is used as the Cin of S[4], and when there is only one 1, it can directly enter S[3] as Cin. When both are = 0, there is no need to carry. Therefore, c = xor(c_temp[1],c_temp[2]). It can be seen that at this time c_temp[1] = 0 and c_temp[2] = 1, which means that only one bit will be carried. Therefore, c = 1 and n_c = 0, which means that the cin of s[3] = 1. Then the general FA is done later, and so on.
Superlint screenshot and coverage

Coverge = 1

Appendix A : Commands we will use to check your homework

Problem		Command
ProbA	Compile	% ncverilog kmap.v
	Simulate	% ncverilog kmap_tb.v kmap.v +access+r +define+FSDB
ProbB	Compile	% ncverilog ripple_adder.v
	Simulate	% ncverilog ripple_adder_tb.v ripple_adder.v +access+r +define+FSDB
ProbC	Compile	% ncverilog multi.v
	Simulate	% ncverilog multi_tb.sv multi.v +access+r +define+FSDB