

National Cheng Kung University

Department of Electrical Engineering

Introduction to VLSI CAD (Spring 2024)

Lab Session 2

Design and Simulation of Carry-Lookahead Adder & Parallel-Prefix Adder & Multiplier

Name	Student ID	
游宗謀	E94106151	
Practical Sections:	Points	Marks
Prob A	15	
Prob B	30	
Prob C	40	
Report	15	
Bonus	10	
Notes		

Due Date: 14:59, March 13, 2024 @ moodle

Deliverables

- 1) All Verilog codes including testbenches for each problem should be uploaded.
NOTE: Please **DO NOT** include source code in the paper report!
- 2) All homework requirements should be uploaded in this file hierarchy or you will not get the full credit.
NOTE: Please **DO NOT** upload waveforms!
- 3) **Important! TA will use the command in Appendix A to check your design under SoC Lab environment, if your code can not be recompiled by TA successfully using the commands, you will not get the full credit.**
- 4) If you upload a dead body which we can't even compile you will get **NO** credit!
- 5) All Verilog file should get at least **90%** superLint Coverage.
- 6) **File hierarchy should not be changed; it may cause your code can not be recompiled by TA successfully using the autograding commands**

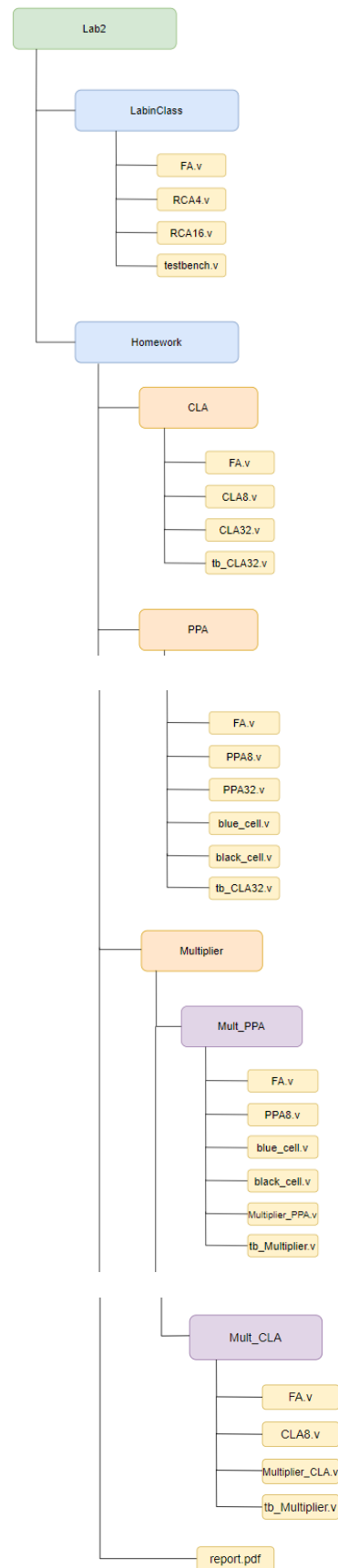


Fig.1 File hierarchy for Homework submission

Objectives:

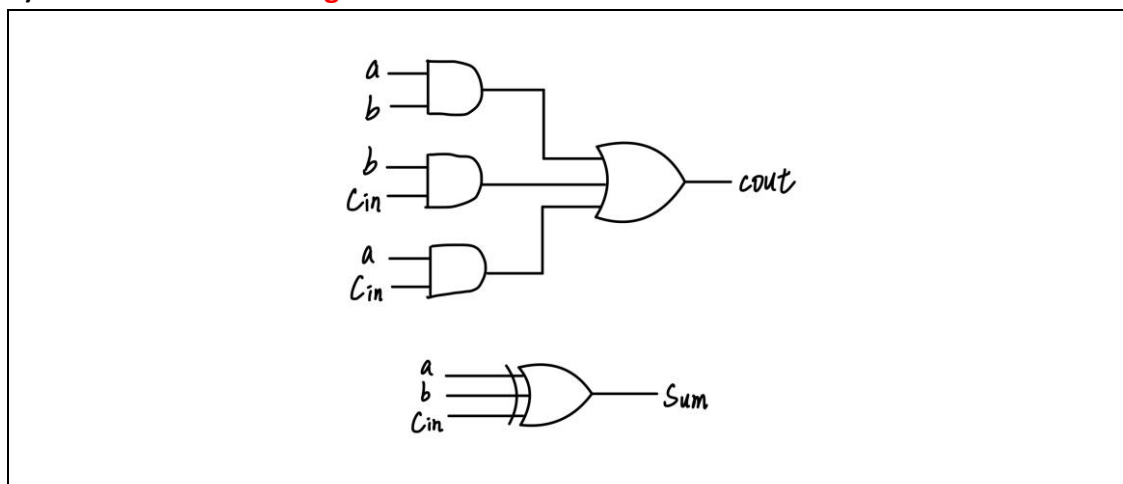
Help students get familiar with the CAD tools (VCS & Verdi) for digital logic design. Introduce different adder architectures and the algorithms behind them. Please go through the hands-on exercise step-by-step.

Prob A: Design Steps & Carry-Lookahead Adder

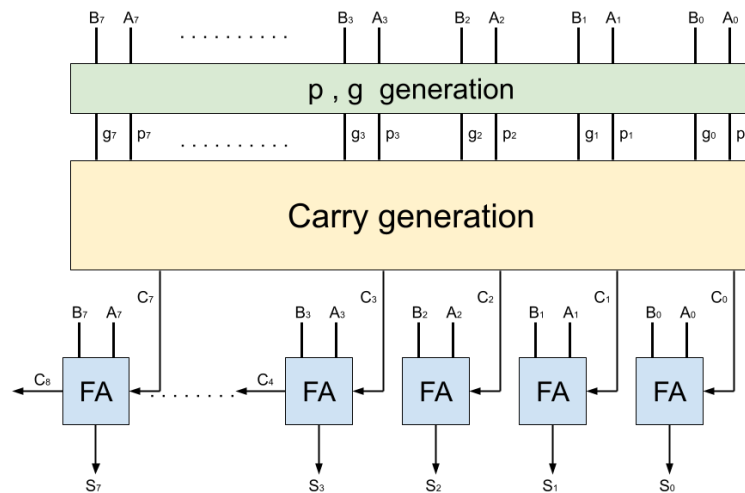
- 1) An adder is a digital circuit that performs addition of number. Please design a full adder in gate level.
- 2) The truth table of full adder

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) Draw a full adder in **gate level**.



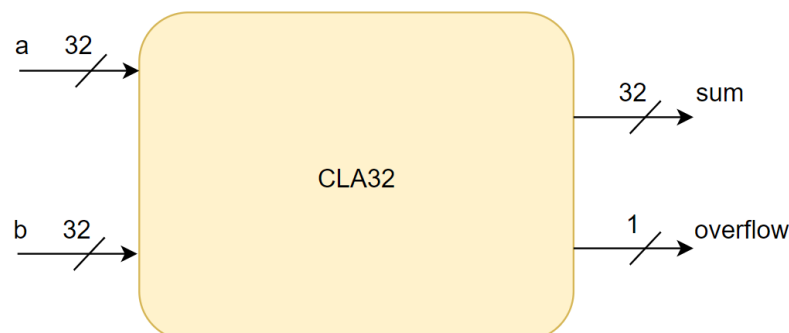
- 4) Design a full adder in **Structural coding** (The module name should be **FA**. And the file you include should be **FA.v**)
- 5) Carry-lookahead-adder uses carry generator to alleviate the lengthy carry propagation procedure in ripple-carry-adder.
- 6) Derive the 8th carry bit C₈(ex. $C_2 = A_1B_1 + (A_1 + B_1)C_1 = g_1 + p_1C_1 = g_1 + p_1g_0 + p_1p_0C_0$)



ex. $C_2 = A_1B_1 + (A_1 + B_1)C_1 = g_1 + p_1C_1 = g_1 + p_1g_0 + p_1p_0C_0$, $g_1 + p_1g_0 + p_1p_0C_0$ is the final result.

$$C_8 = A_7B_7 + (A_7 \oplus B_7)C_7 = g_7 + p_7C_7 = g_7 + p_7g_6 + p_7p_6g_5 + p_7p_6p_5g_4 + p_7p_6p_5p_4g_3 + p_7p_6p_5p_4p_3g_2 + p_7p_6p_5p_4p_3p_2g_1 + p_7p_6p_5p_4p_3p_2p_1g_0 + p_7p_6p_5p_4p_3p_2p_1p_0C_0$$

- 7) Design a 8-bit carry lookahead adder in **Structural coding** (The module name should be **CLA8**. And the file you include should be **CLA8.v**)
- 8) Design a 32-bit carry lookahead adder in **hierarchical coding** using previously designed CLA8 module. (The module name should be **CLA32**. And the file should be **CLA32.v**)



Signal	IO	Bits	Description
a	Input	32	Addend

b	Input	32	Augend
sum	Output	32	Result after calculating
overflow	Output	1	Overflow detection

- 9) Simulate your design with the following test pattern in sample testbench.
 (Hint: The command for compiling is **% vcs -R tb_CLA.32v -full64**)
 (Hint: The command for simulation is **%vcs -R tb_CLA32.v -debug_access+all -full64 +define+FSDB**)
- 10) 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, and make sure that there are no error messages and coverage with Superlint must > 90 %. Snapshot the result and *calculate Superlint coverage*. (Hint: The command to open Superlint is **%jg -superlint superlint.tcl**)
- 11) You only need to upload your v-code to moodle, **do not** paste your code here.

Your simulation result on the terminal.

```

[0]
[File Edit View Search Terminal Help]
Back to file "tb CLA32.v".
Top Level Modules:
  tb
Timescale is 1 ns / 10 ps
Starting vcs inline pass...

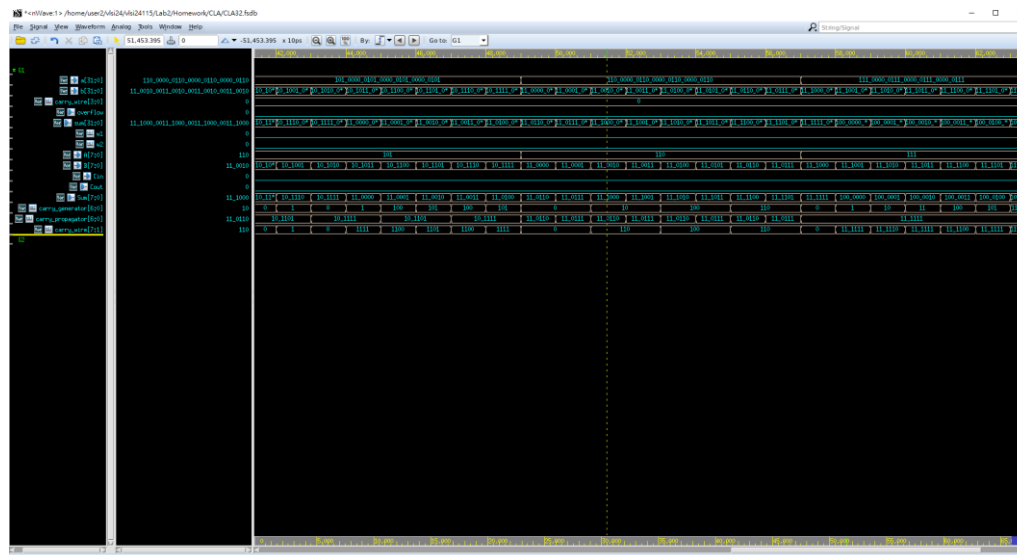
3 modules and 8 UDP read.
recompiling module FA
recompiling module CLA8
recompiling module tb
All of 3 modules done
Warning : License for product VCS-BASE-COMPLETE will expire within 22 days, on: 31-mar-2024.

If you would like to temporarily disable this message, set
the VCS_LIC_EXPIRE_WARNING environment variable to the number of days
before expiration that you want this message to start (the minimum is 0).
rm -f _carcrt.so _crtcr.so pre_vcsobj.*.so share_vcsobj.*.so
if [ -x ../simv ]; then chmod +x ../simv fi
g++ -o ../simv -rdynamic -Wl,-rpath=/usr/cad/synopsys/vcs/cur/linux64/lib -L/usr/cad/synopsys/vcs/cur/linux64/lib -Wl,-rpath-link=/usr/lib64/libnuma.so.1 -obj/libm0w
6.0 -3864 archive 1:so -SIM 1.0 -mpuaps.mps.o mpuaps.o mpar.o mpar.nd.o mpar.llm.9.1.0 mpar.llm.9.0.0 -lsvirsh -lerrorlib -lsgnmlloc -lvsf -lvcshw -lsmgrfile -lucimative /usr/cad/synopsys/vcs/cur/linux64/
lib/vcs.11.0 -Wl,-whole-archive -lvsucli -Wl,-no-whole-archive _vcs.pli_stub_ /usr/cad/synopsys/vcs/cur/linux64/lib/vcs_save_restore_new.o /usr/cad/synopsys/verdi/cur/share/PLI/VCS/LINUX64/pli.a -ldl -lc -lm -pthread
-lld
../simv up to date
Info: [VCS SAVE RESTORE INFO] ASLR (Address Space Layout Randomization) is detected on the machine. To enable save functionality, ASLR will be switched off and simv re-executed.
Please use "no save" simv switch to avoid re-execution or "-suppressASLR_DETECTED_INFO" to suppress this message.
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version U-2023.03-SP2 Full64; Runtime version U-2023.03-SP2 Full64; Mar 18 18:15 2024
Warning : License for product VCS-BASE-RUNTIME will expire within 22 days, on: 31-mar-2024.

If you would like to temporarily disable this message, set
the VCS_LIC_EXPIRE_WARNING environment variable to the number of days
before expiration that you want this message to start (the minimum is 0).
*Verdi* Loading Libcsrc vcs20230310
FSDB Dumper for VCS, Release Verdi U-2023.03-SP2, Linux x86_64/64bit, 08/28/2023
(C) 1998 - 2023 by Synopsys, Inc.
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this file.
*Verdi* : Create FSDB file "CLA32.fsb"
*Verdi* : Begin Traversing the scopes, layer (0).
*Verdi* : End of Traversing.
*****
** Congratulations !! **
** **
** Simulation PASS!! **
** ^ ^ ^ ^ ^ **
** ^ ^ ^ ^ ^ **
*****

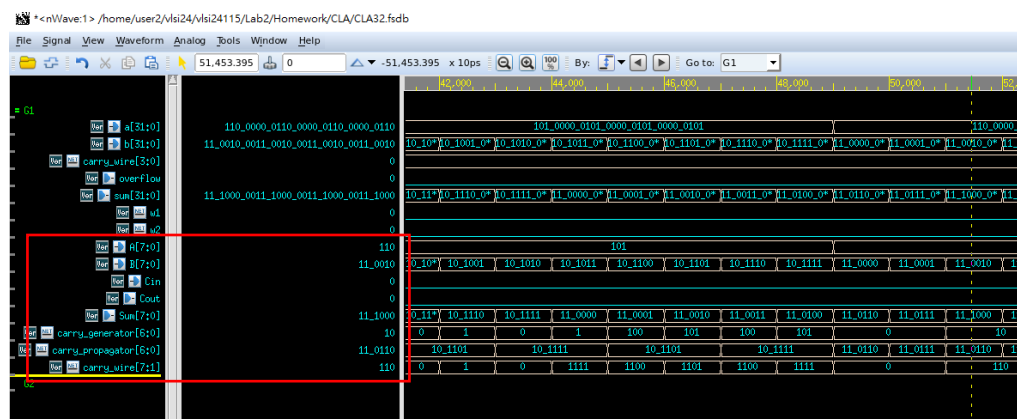
$finish called from file "tb CLA32.v", line 101.
$finish at simulation time
V C S - S i m u l a t i o n R e p o r t
Time: 600000 ps
CPU Time: 6.618 seconds: Data structure size: 0.0MB
Sun Mar 18 18:15:35 2024
CPU time: .334 seconds to compile + .599 seconds to elab + .404 seconds to link + .651 seconds in simulation
VLSIcad: /home/user2/vlsi24/vlsi24115/Lab2/Homework/CLA32.fsb
```

Your waveform



Explanation of your waveform :

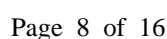
由於此 Adder 是由 4 個 8 bit 的 CLA 串接而成，所以選擇其中一個 CLA 做詳細的波形解釋。



A = 110; B = 110010; Cin = 0

Cout = 0，為 A[7]、B[7]、carry_wire[6]經 Full Adder 運算得到

carry_generator = 10，為 A、B 各個位元作相對應的 and

$$\text{overflow} = A[31]'B[31]'\text{sum}[31] + A[31]B[31]\text{sum}[31]' = w1 + w2 = 0 + 0 = 0$$
$$\text{Coverage} = (1 - (4/199)) * 100\% = 97.99\%$$


Prob B: Parallel-Prefix Adder

- 1) Parallel-prefix-adder reduces the complexity of the carry generation circuitry in carry-lookahead-adder.
- 2) Recursive formulation of carry bits:

$$Q(m, n) = \sum_{i=n}^m (\prod_{r=i+1}^m p_r) g_i$$

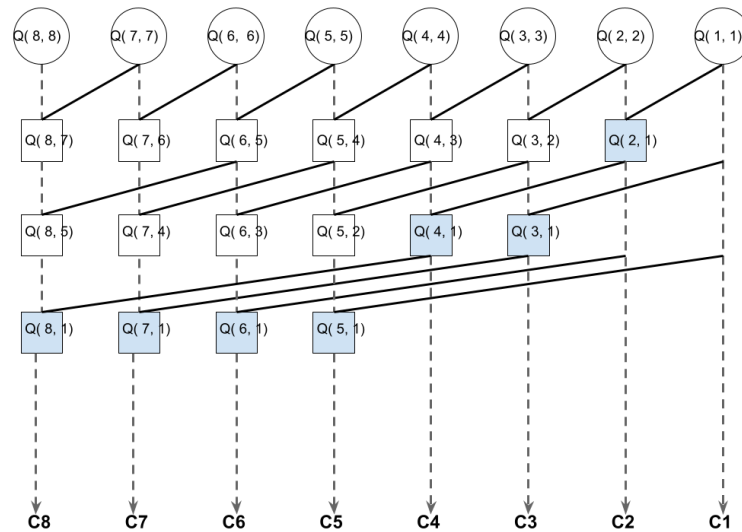
$$C_0 = 0$$

$$C_1 = A_1 B_1 + (A_1 \oplus B_1) C_0 = g_1 + p_1 C_0 = g_1 = Q(1, 1)$$

$$C_2 = A_2 B_2 + (A_2 \oplus B_2) C_1 = g_2 + p_2 C_1 = g_2 + p_2 g_1 = Q(2, 1) = p_2 Q(1, 1) + Q(2, 2)$$

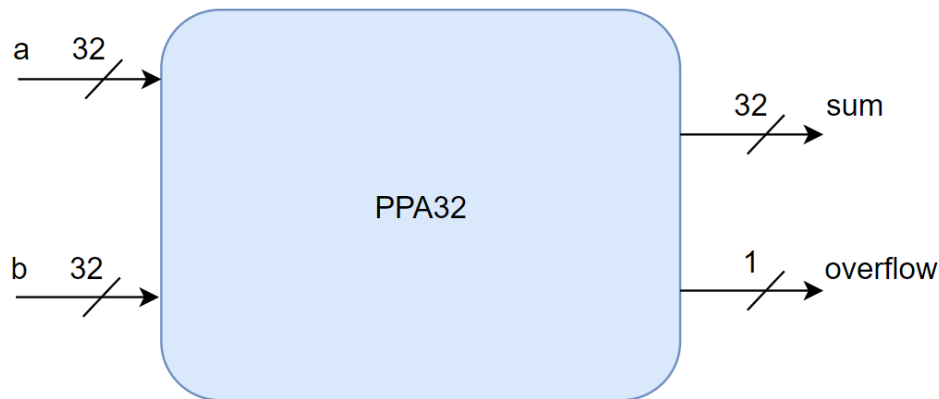
$$C_3 = A_3 B_3 + (A_3 \oplus B_3) C_2 = g_3 + p_3 C_2 = g_3 + p_3 g_2 + p_3 p_2 g_1 = Q(3, 1) = p_3 p_2 Q(1, 1) + Q(3, 2)$$

$$C_4 = A_4 B_4 + (A_4 \oplus B_4) C_3 = g_4 + p_4 C_3 = g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 = Q(4, 1) = p_4 p_3 Q(2, 1) + Q(4, 3)$$



- 3) Design a full adder in **Structural coding** (The module name should be **FA**. And the file you include should be **FA.v**)
- 4) Design black cell & blue cell in **Structural coding** (The module name should be **black_cell** & **blue_cell**. And the file you include should be **black_cell.v** & **blue_cell.v**)
- 5) Design a 8-bit parallel-prefix-adder in **hierarchical coding** using black cell & blue cell as basic unit (The module name should be **PPA8**. And the file you include should be **PPA8.v**)
- 6) Design a 32-bit parallel-prefix-adder in **hierarchical coding** using previously designed PPA8 module. (The module name should be **PPA32**. And the file should

be PPA32.v)



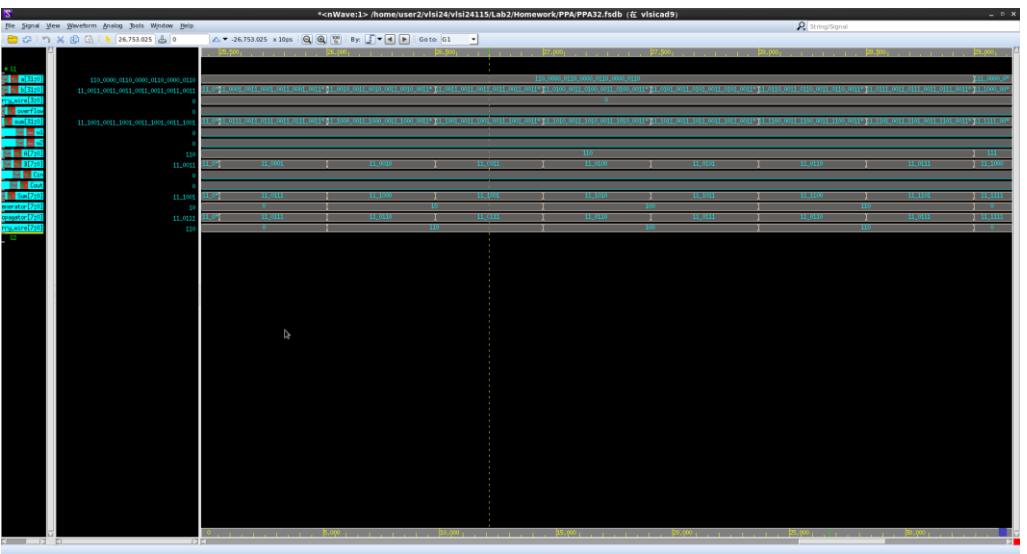
Signal	IO	Bits	Description
a	Input	32	Addend
b	Input	32	Augend
sum	Output	32	Result after calculating
overflow	Output	1	Overflow detection

- 7) Simulate your design with the following test pattern in sample testbench.
 (Hint: The command for compiling is **% vcs -R tb_PPA32.v -full64**)
 (Hint: The command for simulation is **%vcs -R tb_PPA32.v -debug_access+all -full64 +define+FSDB**)
- 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, and make sure that there are no error messages and coverage with Superlint must > 90 %. Snapshot the result and *calculate Superlint coverage*. (Hint: The command to open Superlint is **%jg -superlint superlint.tcl**)
- 9) You only need to upload your v-code to moodle, **do not** paste your code here.

Your simulation result on the terminal.

[illegible]

Your waveform :



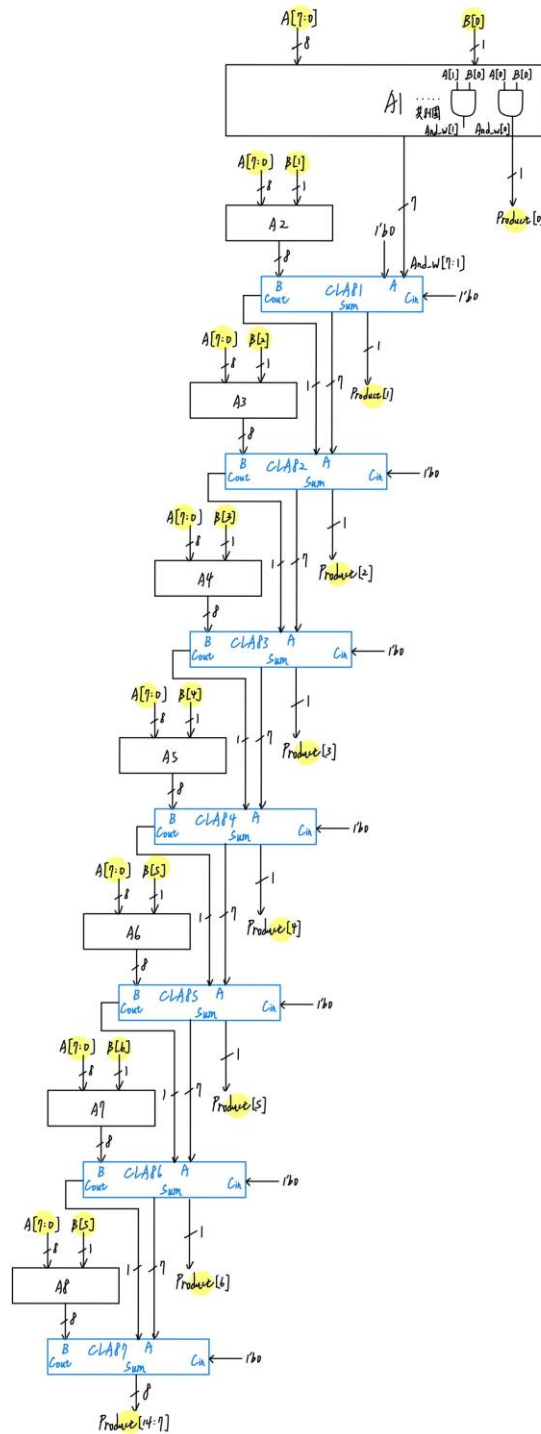
Explanation of your waveform :

由於此 Adder 是由 4 個 8 bit 的 PPA 串接而成，所以選擇其中一個 PPA 做詳細的波形解釋。

Prob C: Application

- 1) Design a 8*8-bit multiplier
 - a. Using CLAs of Prob A
 - b. Using PPAs of Prob B
 - c. Design in gate level rather than behavioural modelling. (The module name should be **Multiplier_CLA & Multiplier_PPA**. And the file you include should be **Multiplier_CLA.v & Multiplier_PPA.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 with the testbench which includes all case of input.
(Hint: The command for compiling is **% vcs -R tb_Multiplier.v -full64**)
(Hint: The command for simulation is **% vcs -R tb_Multiplier.v -debug_access+all -full64 +define+FSDB**)
- 10) 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.

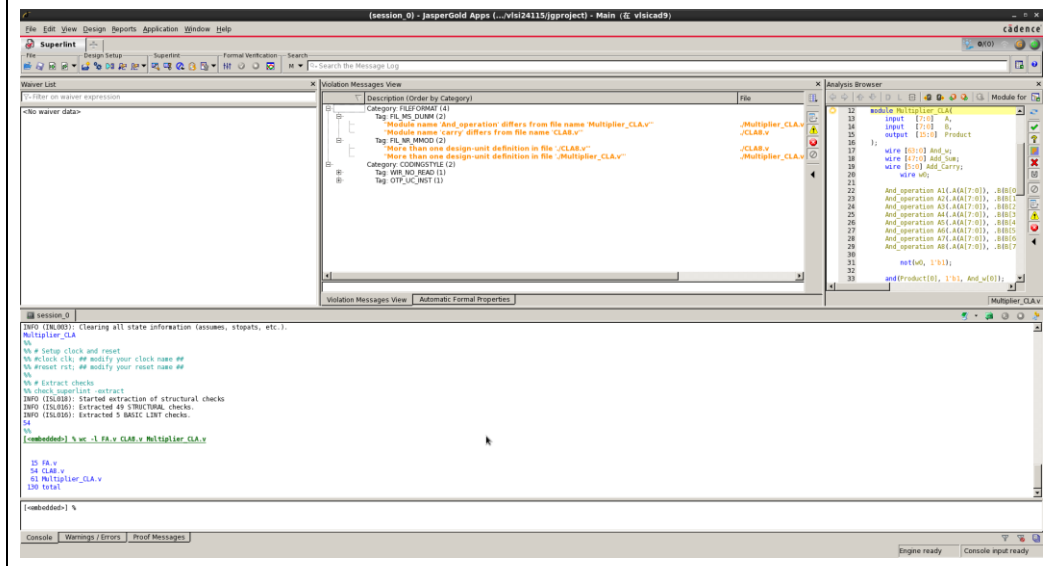


Your simulation result on the terminal.

AND 所得到的結果

Superlint screenshot and coverage

Coverage = $(1 - (6/130)) * 100\% = 95.38\%$



Appendix A : Commands we will use to check your homework

Problem		Commands
ProbA	Compile	% vcs -R tb_CLA32.v -full64
	Simulate	% vcs -R tb_CLA32.v -debug_access+all -full64 +define+FSDB
ProbB	Compile	% vcs -R tb_PPA32.v -full64
	Simulate	% vcs -R tb_PPA32.v -debug_access+all -full64 +define+FSDB
ProbC	Compile	% vcs -R tb_Multiplier.v -full64
	Simulate	% vcs -R tb_Multiplier.v -debug_access+all -full64 +define+FSDB