

National Cheng Kung University

Department of Electrical Engineering

Introduction to VLSI CAD (Spring 2025)

Lab Session 3

Design of ALU and Multiplication Using Verilog Coding

Name	Student ID	
Practical Sections:	Points	Marks
Lab in class	10	
Prob A	30	
Prob B	25	
Prob C	20	
Report	15	
Notes		

Due Date: 15:10, March 12, 2025 @ moodle

Deliverables

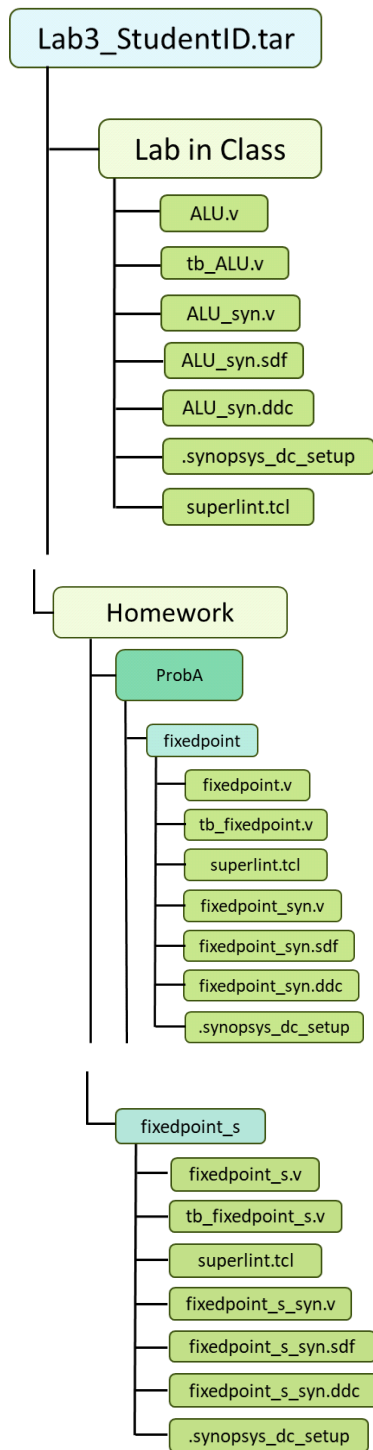
- All Verilog codes including testbenches for each problem should be uploaded.

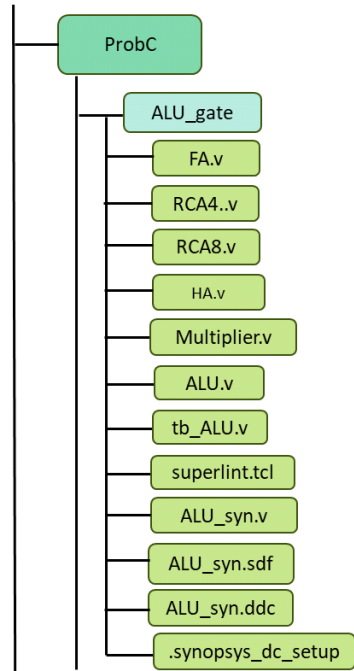
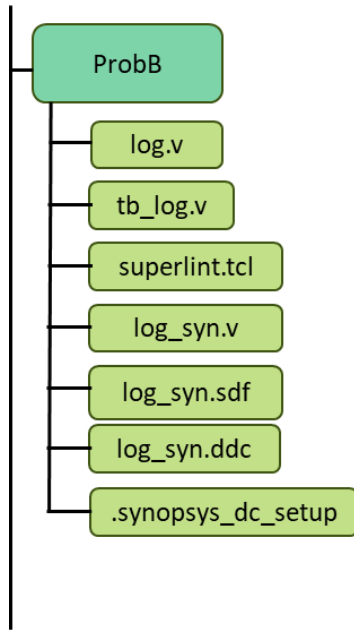
NOTE: Please **DO NOT** include source code in the paper report!

- All homework requirements should be uploaded in this file hierarchy or you will not get the full credit.

NOTE: Please **DO NOT** upload waveforms!

- 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.
- If you upload a dead body which we can't even compile you will get **NO** credit!
- All Verilog file should get at least **90%** superLint Coverage.
- File hierarchy should not be changed; it may cause your code can not be recompiled by TA successfully using the autograding commands





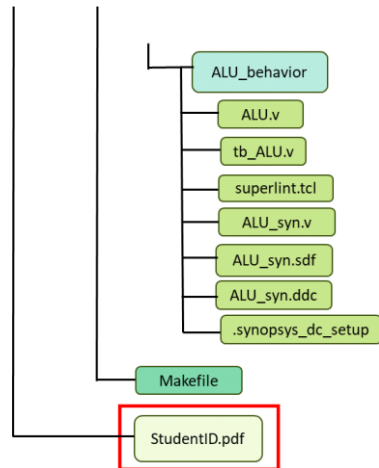


Fig.1 File hierarchy for Homework submission

Lab in class: Arithmetic Logic Unit

- Design a Arithmetic Logic Unit based on the reference code, please implement the following operations.

Signal	I/O	Bits	Description
alu_op	input	5	Select which operation to be executed
src1	input	32	ALU source1
src2	input	32	ALU source2
alu_out	output	32	ALU result
alu_overflow	output	1	When overflow occurs in alu_out during addition or subtraction, alu_overflow1 should be 1, otherwise 0

alu_op	operation	Description
00000	ADD	$alu_out = src1_{signed} + src2_{signed}$
00001	SUB	$alu_out = src1_{signed} - src2_{signed}$
00010	OR	$alu_out = src1 \text{ or } src2$
00011	AND	$alu_out = src1 \text{ and } src2$
00100	XOR	$alu_out = src1 \text{ xor } src2$
00101	NOT	$alu_out = \text{inversion of } src1$
00110	NAND	$alu_out = src1 \text{ nand } src2$
00111	NOR	$alu_out = src1 \text{ nor } src2$
01000	SLT	$alu_out = (src1_{signed} < src2_{signed}) ? 32'd1 : 32'd0$
01001	SLTU	$alu_out = (src1_{unsigned} < src2_{unsigned}) ? 32'd1 : 32'd0$
01010	ABS	$alu_out = (src1 < 0) ? -src1 : src1$
01011	BITREV	$alu_out = \text{bitwise reversal of } src1$

- | Timing (slack) | Area (total cell area) | Power (total) |
|----------------|------------------------|---------------|
| 19.11 | 260.755 | 9.1911e-02 mW |

- ```

Your simulation result on the terminal.
ppppp ppppppppp aaaaaaaaaaaaaa sssssssssss sssssssssss
p:::ppp:::~::~:p a:::~::~:a ss::~::~:s ss::~::~:s
p:::~::~:p aaaaaaaa:::ass:::~::~:s ss:::~::~:s
pp:::~::~:pppp:::~::~:p a:::as:::~::~:sss:::~::~:sss:::~::~:sss:::~::~:s
p:::~::~:p p:::~::~:p aaaaaa:::~::~:a s:::~::~:s ssssss s:::~::~:s ssssss
p:::~::~:p p:::~::~:p aa:::~::~:a s:::~::~:s s:::~::~:s
p:::~::~:p p:::~::~:p a:::aaaa:::~::~:a s:::~::~:s s:::~::~:s
p:::~::~:p p:::~::~:p a:::assssss s:::~::~:s ssssss s:::~::~:s
p:::~::~:ppppp:::~::~:ppa:::~::~:a a:::as:::~::~:sss:::~::~:sss:::~::~:sss:::~::~:s
p:::~::~:~::~:pp:::~::~:p a:::~::~:aaaa:::~::~:as:::~::~:s s:::~::~:s s:::~::~:s
p:::~::~:~::~:ppp a:::~::~:aa:::~::~:as:::~::~:ss s:::~::~:s
p:::~::~:pppppppp aaaaaaaa aaaa sssssssssss sssssssssss
p:::~::~:p
p:::~::~:p
pppppppppp
,((((((((((0))))))))),
,((((((((((0))))))))),
,(((((((((\|//\))))))))),
,((((((((((//\))))))))),
((((((((/////^\\))))))))
(((((' _"- -"- _.'))))))
((((_.-. _.-/ ^ _))))
((((_=(0)) (0) _=- _))))
'((((_/' _.' _.'\ _))))'
'((((_.' A _.- /)))')
'((((_./ _./ /)))')
'((((_.' _.-o _.' _.')))'
' _.-, __, _.-'

$finish called from file "tb_ALU.v", line 304.
$finish at simulation time 23000
V C S S i m u l a t i o n R e p o r t
Time: 230000 ps
CPU Time: 0.450 seconds; Data structure size: 0.0Mb
Thu Mar 6 14:34:29 2025
CPU time: .565 seconds to compile + .204 seconds to elab + .257 seconds to link + .475 seconds in simulation

```

The image displays two views of the ALU circuit: RTL (Right-Hand Side) and Synthesis (Syn). The RTL view shows the circuit logic with inputs src1[31:0] (7fff\_ffff), src2[31:0] (2), alu\_op[4:0] (0), and output alu\_out1[31:0] (8000\_0001). The Synthesis view shows the same circuit after synthesis, with inputs src1[31:0] (F), src2[31:0] (2), alu\_op[4:0] (0), and output alu\_out1[31:0] (XXXX\_XXXX). The alu\_overflow signal is shown as 0 in RTL and x in Synthesis.

Input1:(src1, src2, alu\_op) = (f, 2, 0)，做 sign 相加，結果為  $f+2 = 11(\text{hex})$ ，overflow = 0,

Input2(src1, src2, alu\_op) = (ffffff, 2, 0) , fffffff 在 sign 看成-1 ,  
 結果為 fffffff+2 = 1(hex) , overflow = 0 。  
 Input3(src1, src2, alu\_op) = (7fff\_ffff, 2, 0),  
 7fff\_ffff 換成二進位是 0111 1111...(省略號)..1111 , 再加上 2 後產生 overflow 。  
 結果與預期相符 。  
 合成後的波型會多 delay 。

### SuperLint Coverage

Coverage = 100 %

### Prob A-1: Practice fixed point(unsigned)

- Design an **unsigned** fixed-point multiplier based on the reference code, The decimal part should be rounded(round to nearest, ties to even).

| Signal     | I/O    | Bits | Description                         |
|------------|--------|------|-------------------------------------|
| <b>in1</b> | input  | 8    | unsigned integer[7:5], decimal[4:0] |
| <b>in2</b> | input  | 8    | unsigned integer[7:5], decimal[4:0] |
| <b>out</b> | output | 8    | unsigned integer[7:2], decimal[1:0] |

- Simulate your design with the following test pattern in sample testbench.
- Verify your design by comparing the simulation results with the results you predicted. If the results are not the same, please go back 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.
- Follow the PPT file to **synthesize** your code.
- After you synthesize your design, you may have some information about the circuit. Fill in the following form.

| Timing (slack) | Area (total cell area) | Power (total) |
|----------------|------------------------|---------------|
| 19.47          | 80.09                  | 5.8227e-02mW  |

- You only need to upload your v-code to moodle, **do not** paste your code here.

Your simulation result on the terminal.

```

** **
** Congratulations !! **
** **
** Simulation PASS!! **
** **

 _ _ _
 / \
 / \
 / ^ ^ ^ \
 / ^ ^ ^ \
 | ^ ^ ^ ^ |w|
 | ^ ^ ^ ^ |
 \m m_ | _|

$finish called from file "tb_fixedpoint.v", line 100.
$finish at simulation time 10100
 V C S S i m u l a t i o n R e p o r t
Time: 101000 ps
CPU Time: 0.260 seconds; Data structure size: 0.0Mb
Thu Mar 6 15:02:56 2025
CPU time: .270 seconds to compile + .208 seconds to elab + .250 seconds to link + .295 seconds in sim

```

### Your waveform (RTL & Synthesis) :

#### RTL



#### SYN



### Explanation of your waveform :

Input1: in1 = 1110\_0000(7), in2 = 1110\_0000(7), 7\*7 應等於 49, 換成 binary 表示, 前 6bit 整數, 後 2bit 小數 = 1100\_0100(49)。

Input2: in1 = 0111\_1000, in2 = 0011\_0000, 相乘得 0001\_0110\_1000\_0000, 判斷第 bit[7:6], 因為是 10, 剛好在中間, 所以要取最接近偶數, 接著判斷 bit[9:8], 因為是 10, 已經是偶數, 故不須進位。

答案最後為 0001\_0110

### SuperLint Coverage

Violation Messages View

Category: RACES (1)

Tag: REG\_NRTTRC (1)

"A trigger-propagation race exists between 'rounded\_product[15:8]' and 'rounded\_product[8]'"

Analysis Browser

```

1 module fixedpoint(
2 input [7:0] in1, // unsigned integer[7:5], decimal[4:0]
3 input [7:0] in2, // unsigned integer[7:5], decimal[4:0]
4 output reg [7:0] out // unsigned integer[7:2], decimal[1:0]
5);
6 // put your design here
7
8 reg [15:0] product, rounded_product;
9
10 always @(*)
11 begin
12 product = in1 * in2;
13 case(product[7:6])
14 2'b00: rounded_product = product;
15 2'b01: rounded_product = product;
16 2'b10: begin
17 if(product[9:8] == 2'b01 || product[9:8] == 2'b11)
18 rounded_product = product + 16'b00000000100000000;
19 else rounded_product = product;
20 end
21 2'b11: rounded_product = product + 16'b00000000100000000;
22 end
23 out = rounded_product[15:8];
24 end

```

Coverage = (26-1/26) \* 100 = 96%

### Prob A-2: Practice fixed point (signed)

- Design a **signed** fixed-point multiplier based on the reference code, The decimal part should be rounded(round to nearest, ties to away).

| Signal | I/O   | Bits | Description                       |
|--------|-------|------|-----------------------------------|
| in1    | input | 8    | signed integer[7:5], decimal[4:0] |
| in2    | input | 8    | signed integer[7:5], decimal[4:0] |



|     |        |   |                                   |
|-----|--------|---|-----------------------------------|
| out | output | 8 | signed integer[7:2], decimal[1:0] |
|-----|--------|---|-----------------------------------|

- Simulate your design with the following test pattern in sample testbench.
- Verify your design by comparing the simulation results with the results you predicted. If the results are not the same, please go back 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.
- Follow the PPT file to **synthesize** your code.
- After you synthesize your design, you may have some information about the circuit. Fill in the following form.

| Timing (slack) | Area (total cell area) | Power (total) |
|----------------|------------------------|---------------|
| 19.43          | 71.74                  | 5.7478e-02 mW |

- You only need to upload your v-code to moodle, do not paste your code here.

### Your simulation result on the terminal.

```

** **
** Congratulations !! **
** **
** Simulation PASS!! **
** **

finish called from file "tb_fixedpoint_s.v", line 100.
finish at simulation time 10100
 V C S S i m u l a t i o n R e p o r t
time: 101000 ps
PU Time: 0.270 seconds; Data structure size: 0.0Mb
hu Mar 6 16:32:22 2025
PU time: .286 seconds to compile + .218 seconds to elab + .256 seconds to link + .296 seconds in simulation

```

### Your waveform (RTL & Synthesis) :

RTL:

SYN:

### Explanation of your waveform :

Input2 in1 =1010\_1000, in2 = 0110\_0000 ,  
 $in1 * in2 = -0010\_0001\_0110\_0000 = 1101\_1110\_1010\_0000$  。  
 判斷 bit[7:6] = 10 , 剛好在一半 , 所以要進位變成 1101\_1111

### SuperLint Coverage

Description (Order by Category)

Category: RACES (1)

Tag: REG\_NR\_TRRC (1)

"A trigger-propagation race exists between 'rounded\_product[15:8]' and 'rounded\_product[8]'"

Module for fixedpoint\_s

```

1 module fixedpoint_s
2 input signed [7:0] in1, // signed integer[7:5], decimal[4:0]
3 input signed [7:0] in2, // signed integer[7:5], decimal[4:0]
4 output reg [7:0] out // 8 bit signed integer[7:2], decimal[1:0]
5);
6 // put your design here
7
8 reg [15:0] product, rounded_product;
9
10 always @(*)
11 begin
12 product = in1 * in2;
13 case(product[7:6])
14 2'b00: rounded_product = product;
15 2'b01: rounded_product = product;
16 2'b10: rounded_product = product + 10'b0000000100000000;
17 2'b11: rounded_product = product + 10'b0000000100000000;
18 endcase
19 out = rounded_product[15:8];
20 end
21 endmodule
22

```

Coverage = (21-2) / 21\* 100 % = 90.4%

### Prob B : Practice linear interpolation on logarithmic scale

- Design a **unsigned** fixed-point linear logarithmic interpolation based on the reference code, The decimal part should be rounded(round to nearest, ties to even).

| Signal   | I/O    | Bits | Description                         |
|----------|--------|------|-------------------------------------|
| <b>x</b> | input  | 8    | unsigned integer[7:5], decimal[4:0] |
| <b>y</b> | output | 8    | unsigned integer[7:2], decimal[1:0] |

- Simulate your design with the following test pattern in sample testbench.
- Verify your design by comparing the simulation results with the results you predicted. If the results are not the same, please go back 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.
- Follow the PPT file to **synthesize** your code.
- After you synthesize your design, you may have some information about the circuit. Fill in the following form.

| Timing (slack) | Area (total cell area) | Power (total) |
|----------------|------------------------|---------------|
| 19.87          | 8.13                   | 2.6897e-03 mW |

- You only need to upload your v-code to moodle, **do not** paste your code here.

Your simulation result on the terminal.

```

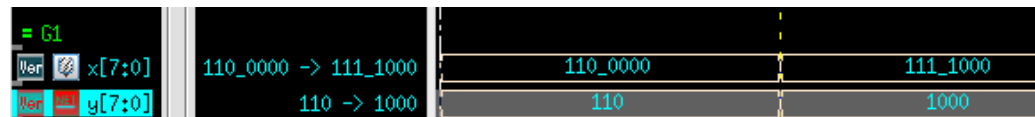
** **
** Congratulations !! **
** **
** Simulation PASS!! **
** **

$finish called from file "tb_log.v", line 100.
$finish at simulation time 10100
VCS Simulation Report
Time: 101000 ps
CPU Time: 0.270 seconds; Data structure size: 0.0Mb
Thu Mar 6 16:59:48 2025
CPU time: .299 seconds to compile + .252 seconds to elab + .280 seconds to link + .293 seconds in simul

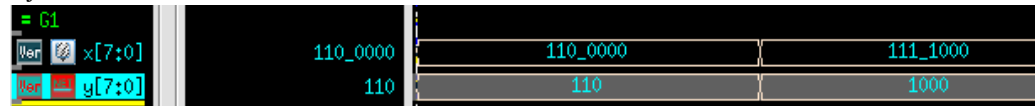
```

### Your waveform (RTL & Synthesis) :

RTL:



Syn:



### Explanation of your waveform :

經過簡化計算， $y = (x-2)/2 + 1$ 。

Input2 x = 0111\_1000(bit[7:5]為整數)， $x-2 = 0011\_1000$ ， $(x-2)/2 = 0001\_1100$ ，

$(x-2)/2 + 1 = 0011\_1100$ ，再開始做 rounding。

看 bit[2:1]，因為是 10，需再判斷 bit[4:3]，因為 bit[4:3]為 11 奇數，須盡位。

變成 01000，因為需要整數 6 位，補齊後得 0000\_1000

### SuperLint Coverage

Category CODINGSTYLE (1)

Tag: ASG\_RHS\_RTRN (1)

Unequal length operands in assignment in module/design-unit 'log'. Length of RHS is 1, but LHS is 2.

Category RACES (1)

Tag: RES\_NP\_TRAC (1)

A trigger-propagation race exists between 'round\_result[7:3]' and 'round\_result[3]'.

```

1 module log(
2 input [7:0] x, //unsigned integer(7:0),decimal(4:0)
3 output reg [7:0] y //unsigned integer(7:2),decimal(1:0)
4);
5
6 // put your design here
7 //more bit to keep fraction
8 reg [7:0] result, round_result;
9
10 always @(*)
11 begin
12 result = ((x - 8'h01000000) >> 1) + 8'h00100000;
13 case(result[2:1])
14 2'b00: round_result = result;
15 2'b01: round_result = result;
16 2'b10: begin
17 if(result[4:3] == 2'b01 || result[4:3] == 2'b11)
18 round_result = result + 8'h00001000;
19 else round_result = result;
20 end
21 2'b11: round_result = result + 8'h00001000;
22 endcase
23 y = {{0{0}}, round_result[7:3]};
24 end

```

Coverage =  $(26-2)/26 * 100 \% = 92.3\%$

### Prob C: Synthesize

- Design the ALU from Lab2 using behavior modeling.
- Follow the PPT file to **synthesize** your code.
- Synthesize the gate-level ALU implemented in Lab2 problem A, and the ALU you design in step(1).
- Determine the **lowest** achievable clock period, along with the corresponding

area and power consumption.

|                         | Clock period | Timing (slack) | Area (total cell area) | Power (total) |
|-------------------------|--------------|----------------|------------------------|---------------|
| <b>ALU (gate-level)</b> | 5.8 ps       | 0              | 120.47                 | 7.4588e-02 mW |
| <b>ALU (behavior)</b>   | 4.3 ps       | 0              | 87.03                  | 6.0211e-02 mW |

- Considering clock period and area, which structure has the better performance.

| Explain of the performance comparison                                                                            |
|------------------------------------------------------------------------------------------------------------------|
| Behavior 有更高的 frequency, 較小的 area 和消耗較少功率, 整體優於 gate level 寫法。因為用 Behavior 撰寫後, tool 會自動進行優化, 所以 performance 較佳。 |

### Lesson Learned

- please write the lessons learned from this lab session, or some suggestions for this lab session. Thank you.

| Lesson Learned from students                                                                         |
|------------------------------------------------------------------------------------------------------|
| 這是第一次在數位電路對小數進行操作, 同時讓我認識到不同的 rounding 方法。此外, 我對於 signed bit 的運算較不熟悉, 這次作業讓我能知道電腦運算的方式, 和知道如何求得正確答案。 |

### Appendix A : Commands we will use to check your homework

|              | Problem |     | Commands                                                     |
|--------------|---------|-----|--------------------------------------------------------------|
| Lab in class | ALU     | RTL | % vcs -R tb_ALU.v -debug_access+all -full64 +define+FSDB     |
|              |         | SYN | % vcs -R tb_ALU.v -debug_access+all -full64 +define+FSDB+syn |
| Homework     | ProbA   | RTL | % make probA_1<br>% make probA_2                             |
|              |         | SYN | % make probA_1_syn<br>% make probA_2_syn                     |
|              | ProbB   | RTL | % make probB                                                 |
|              |         | SYN | % make probB_syn                                             |
|              | ProbC   | RTL | % make probC_1<br>% make probC_2                             |
|              |         | SYN | % make probC_1_syn<br>% make probC_2_syn                     |

