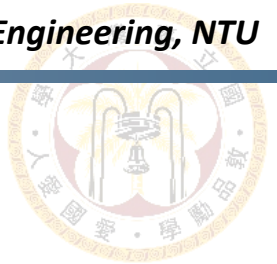
Computer-Aided VLSI System Design

Homework 1: Arithmetic Logic Unit

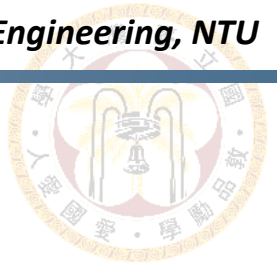
Graduate Institute of Electronics Engineering, National Taiwan University

NTU GIEE



Goal

- In this homework, you will learn
  - How to design ALU with simple operations
  - Differences between combinational circuit and sequential circuit
  - How to define registers and wires
  - How to read spec

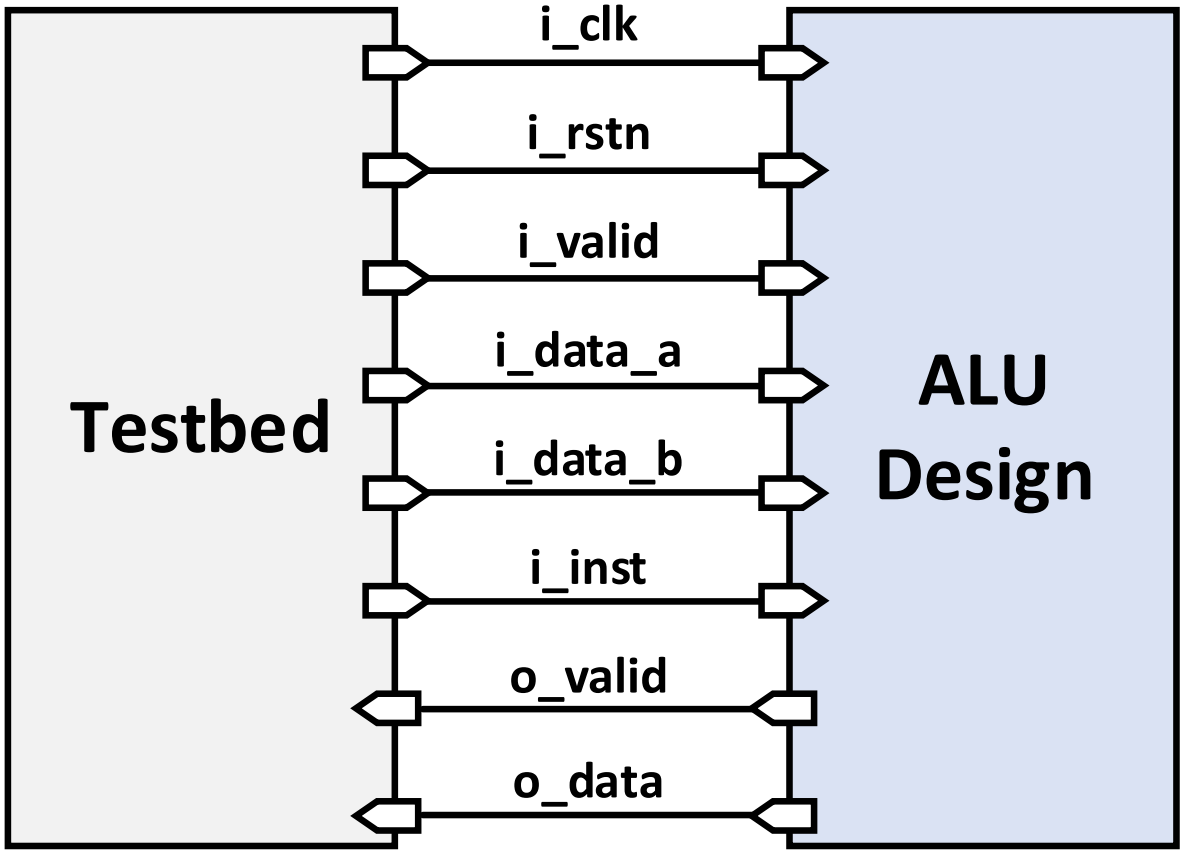


Introduction

- The Arithmetic logic unit (ALU) is one of the components of a computer processor

- In this homework, you are going to design an ALU being able to compute special operations

Block Diagram

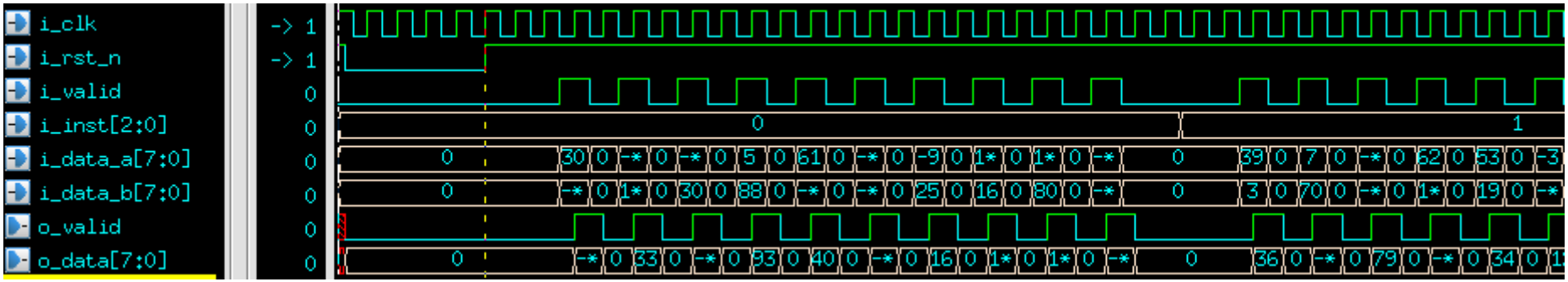


Input/Output

Signal	I/O	Width	Simple Description
i_clk	I	1	Clock signal in the system
i_rst_n	I	1	Active **low** asynchronous reset
i_valid	I	1	The signal is **high** if input data is ready
i_data_a	I	8	Signed input data with 2's complement representation (3b integer + 5b fraction)
i_data_b	I	8	
i_inst	I	3	Instruction for ALU to operate
o_valid	O	1	Set **high** if ALU is ready to output result
o_data	O	8	Result after ALU processing with 2's complement representation (3b integer + 5b fraction)

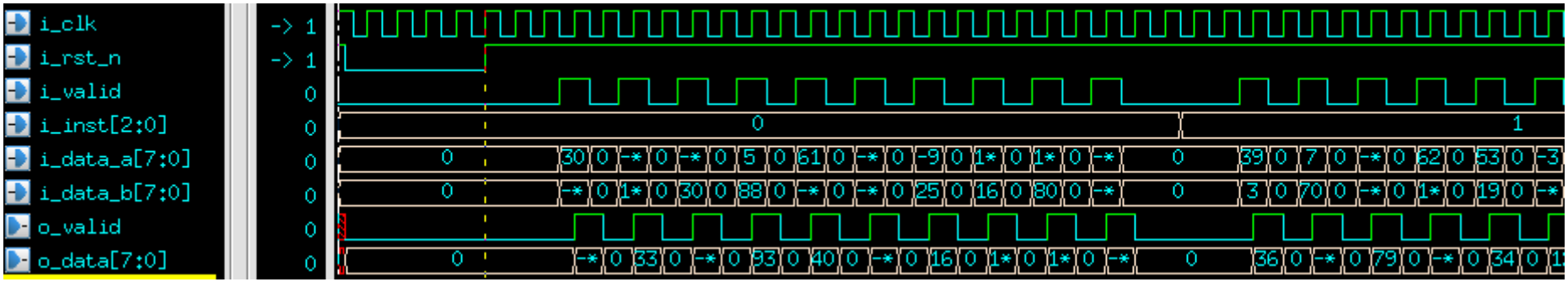
Specification (1)

- All inputs are synchronized with the negative edge clock
- All outputs should be synchronized at clock **rising** edge (Flip-flops are added at outputs)
- Active low asynchronous reset (You should set all your outputs to be zero when i_rst_n is **low**)



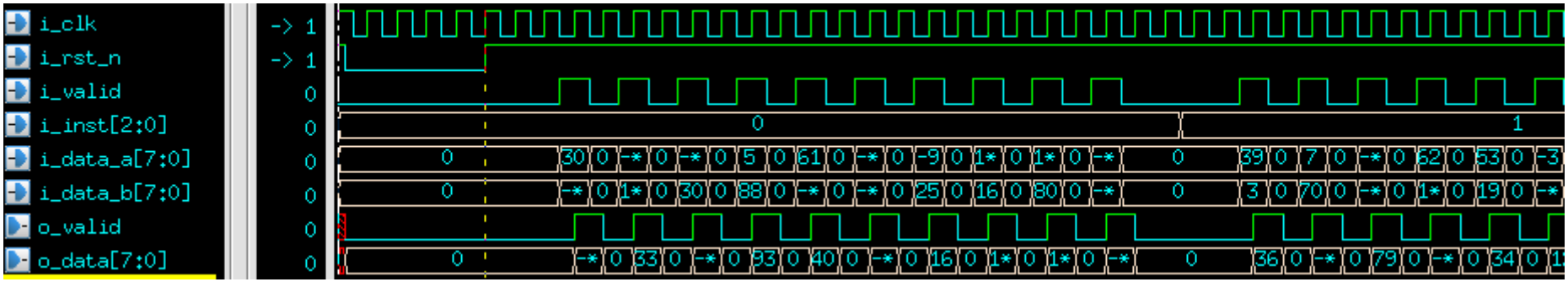
Specification (2)

- i_valid will be pulled **high** for one cycle for ALU to get i_data_a, i_data_b and i_inst
- i_valid will be pulled **high** anytime
- o_valid should be pulled **high** for only **one cycle** for every o_data



Specification (3)

- When o_valid is **high**, the testbench will get your output at negative clock edge to check the answer
- You can raise your o_valid anytime. TA will check your answer when o_valid is high



Instructions

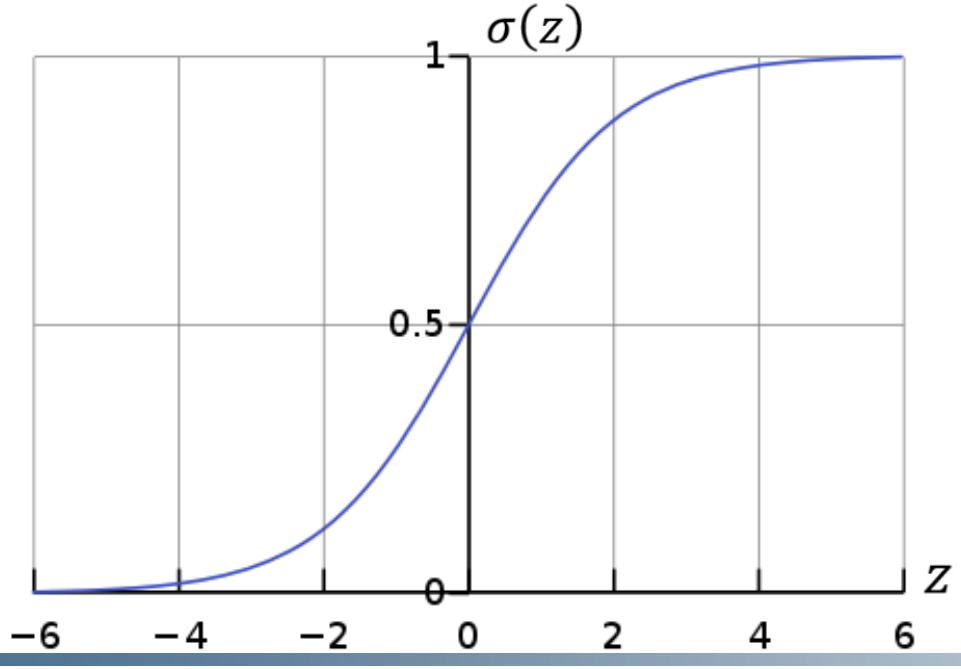
Operation	i_inst [2:0]	Meaning	Note
Signed Addition	2'b000	o_data = i_data_a + i_data_b	Output might saturate
Signed Subtraction	2'b001	o_data = i_data_a - i_data_b	Output might saturate
Signed Multiplication	2'b010	o_data = i_data_a * i_data_b	Output might saturate
NAND	2'b011	o_data = (i_data_a · i_data_b)'	Bit-wise
XNOR	2'b100	o_data = (i_data_a ⊕ i_data_b)'	Bit-wise
Sigmoid	2'b101	o_data = σ(i_data_a)	Use piece-wise linear approximation
Right Circular Shift	2'b110	see p.15	View i_data_b as shift amount (8b integer)
Min	2'b111	o_data = min (i_data_a, i_data_b) If i_data_a = i_data_b, o_data = i_data_a	

Output Saturation & Rounding

- For instruction 000, 001 and 010, output value might exceed the range of 8b representation (3b integer + 5b fraction)
- For instruction 000, 001 and 010, if the output value exceeds the maximum value of 8b representation, use the **maximum value** as output, and vice versa
- For instructions 010, the result need to be **rounded to nearest number** before output

Sigmoid Function

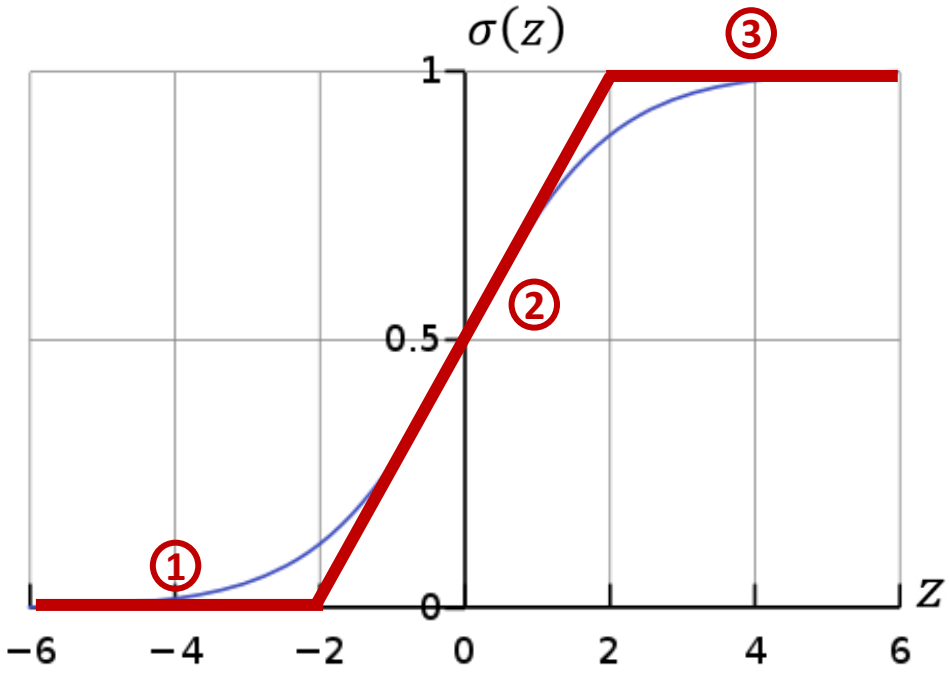
- For instructions 101, you need to implement a sigmoid function which is a popular activation function in in an artificial neural network
- However, it's not intuitive to implement an exponential operation on hardware directly



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

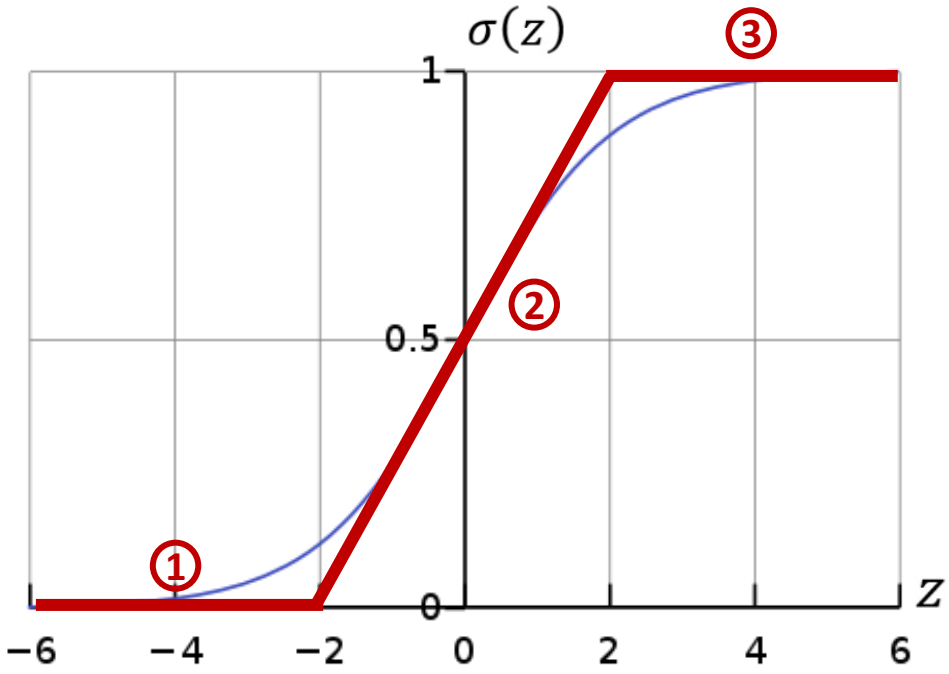
Sigmoid Function Approximation

- In order to have a easier implementation, we use **piecewise linear approximation** to compute sigmoid function
- Divide the curve to **3 straight-line segments** to compute the output



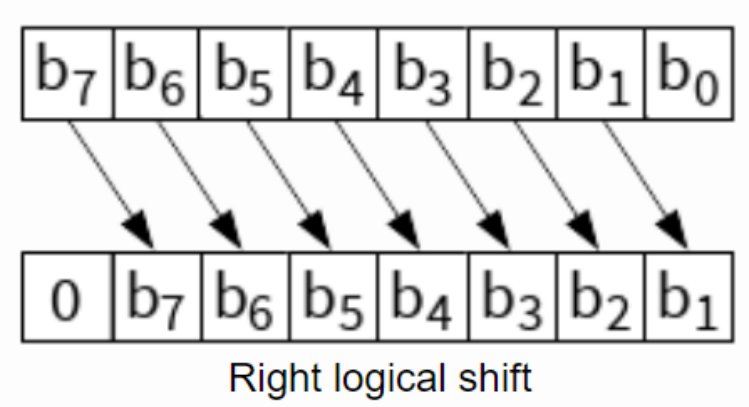
Sigmoid Function Approximation

- Make the appropriate use of the **slope** of 2nd segment
- The 3 segments have 2 intersections **at z=2 & z=-2** respectively
- Output needed to be **rounded down**

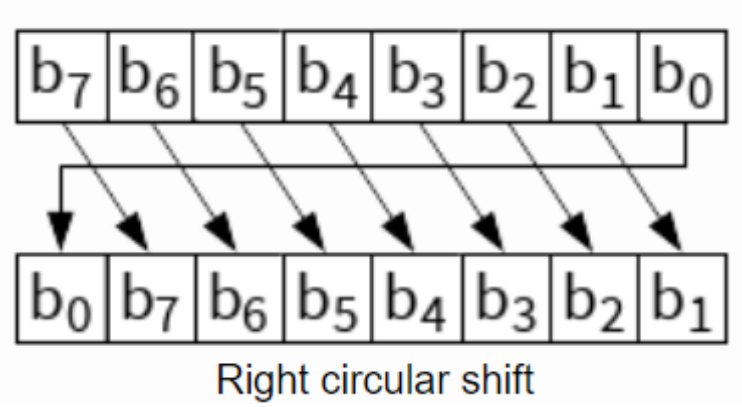


Shift Operations

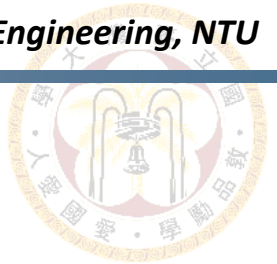
- In **logical shift** the new bits that are shifted in always get the value zero
- In **circular shift**, also called rotation, use the bit that got shifted out at one end and inserts it back as the new bit value at the other end
- For instruction 110, you should implement **right circular shift** function and view i_data_b as rotation amount (8b integer)



Right logical shift



Right circular shift

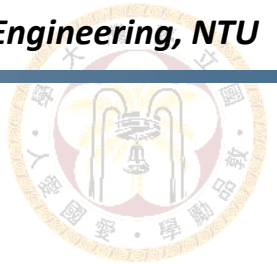


Other Considerations

- Bit-wise operation for instruction 011 and 100
- For instruction 111, if i_data_a = i_data_b, just output i_data_a
- Before and after the operation, if the total number of bits of the variable (reg or wire) is inconsistent, the system presets to complement the zero extension. Please use sign extension for the signal at appropriate time
- For fixed-point operation, please pay attention to the position of separation between integer and fraction after ALU operation

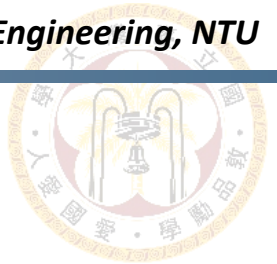
alu.v

```
1   module alu #(
2       parameter INT_W  = 3,
3       parameter FRAC_W = 5,
4       parameter INST_W = 3,
5       parameter DATA_W = INT_W + FRAC_W
6   )(
7       input                    i_clk,
8       input                    i_rst_n,
9       input                    i_valid,
10      input signed [DATA_W-1:0] i_data_a,
11      input signed [DATA_W-1:0] i_data_b,
12      input        [INST_W-1:0] i_inst,
13      output                   o_valid,
14      output       [DATA_W-1:0] o_data
15  );
16
17  // ---------------------------------------------------------------------------
18  // Wires and Registers
19  // ---------------------------------------------------------------------------
20  reg  [DATA_W:0] o_data_w, o_data_r;
21  reg             o_valid_w, o_valid_r;
22  // ---- Add your own wires and registers here if needed ---- //
23
24
```



alu.v

```
25  // ---------------------------------------------------------------------------
26  // Continuous Assignment
27  // ---------------------------------------------------------------------------
28  assign o_valid = o_valid_r;
29  assign o_data = o_data_r;
30  // ---- Add your own wire data assignments here if needed ---- //
31
32
33  // ---------------------------------------------------------------------------
34  // Combinational Blocks
35  // ---------------------------------------------------------------------------
36  // ---- Write your combinational block design here ---- //
37  always@(*) begin
38      o_data_w = ;
39      o_valid_w = ;
40  end
```



alu.v

```
42  // ---------------------------------------------------------------------------
43  // Sequential Block
44  // ---------------------------------------------------------------------------
45  // ---- Write your sequential block design here ---- //
46  always@(posedge i_clk or negedge i_rst_n) begin
47      if(!i_rst_n) begin
48          o_data_r  <= 0;
49          o_valid_r <= 0;
50      end else begin
51          o_data_r  <= o_data_w;
52          o_valid_r <= o_valid_w;
53      end
54  end
55  endmodule
```

testbench.v

```
1   `timescale 1ns/100ps
2   `define CYCLE       10.0
3   `define HCYCLE      (`CYCLE/2)
4   `define MAX_CYCLE   100000
5   `define RST_DELAY   5
6
7   `define Inst0_I "./pattern/INST0_I.dat"
8   `define Inst0_O "./pattern/INST0_O.dat"
9   `define Inst1_I "./pattern/INST1_I.dat"
10  `define Inst1_O "./pattern/INST1_O.dat"
11  `define Inst2_I "./pattern/INST2_I.dat"
12  `define Inst2_O "./pattern/INST2_O.dat"
13  `define Inst3_I "./pattern/INST3_I.dat"
14  `define Inst3_O "./pattern/INST3_O.dat"
15  `define Inst4_I "./pattern/INST4_I.dat"
16  `define Inst4_O "./pattern/INST4_O.dat"
17  `define Inst5_I "./pattern/INST5_I.dat"
18  `define Inst5_O "./pattern/INST5_O.dat"
19  `define Inst6_I "./pattern/INST6_I.dat"
20  `define Inst6_O "./pattern/INST6_O.dat"
21  `define Inst7_I "./pattern/INST7_I.dat"
22  `define Inst7_O "./pattern/INST7_O.dat"
```

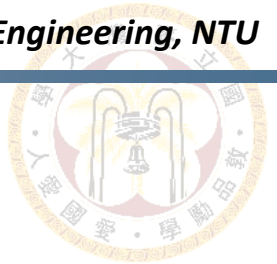
Commands

- 01_run
  - Run all instructions

```
ncverilog testbench.v alu.v +define+ALL +access+rw
```

  - Run specific instrcution

```
ncverilog testbench.v alu.v +define+I0 +access+rw
```



Commands

- 09_clean

```
rm -rf *.history
rm -rf *.key
rm -rf novas.rc
rm -rf novas.fsdb
rm -rf *.log
rm -rf INCA_libs nWaveLog
rm -rf *~
rm -rf nWaveLog
rm -rf BSSLib.lib++
rm -rf novas.conf
```

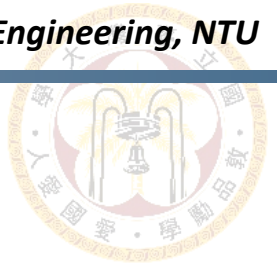
Pattern (Input Data)

	i_data_b	i_data_a
1	11001000	00011110
2	01100110	10111011
3	00011110	10010011
4	01011000	00000101
5	11101011	00111101
6	11001111	11101111
7	00011001	11110111
8	00010000	01111111
9	01010000	01110000
10	10111111	10100101

Pattern (Golden Output)

o_data

1	11100110
2	00100001
3	10110001
4	01011101
5	00101000
6	10111110
7	00010000
8	01111111
9	01111111
10	10000000



Grading Policy (1)

- TA will run your code with following command

```
ncverilog testbench.v alu.v +define+ALL +access+rw
```

Grading Policy (2)

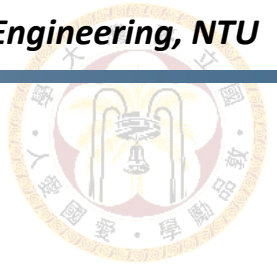
- Released pattern **80%**

Operation	i_inst [2:0]	Score
Signed Addition	2'b000	10%
Signed Subtraction	2'b001	10%
Signed Multiplication	2'b010	10%
NAND	2'b011	5%
XNOR	2'b100	5%
Sigmoid	2'b101	20%
Right Circular Shift	2'b110	10%
Min	2'b111	10%

- Hidden pattern: **20%**
  - Hidden pattern contains 9 different instructions
  - Only if you pass all patterns will you get full 20% score

Grading Policy (3)

- Delay submission
  - In one day: (original score)***0.6**
  - In two days: (original score)***0.3**
  - More than two days: **0 point** for this homework
- Lose **3 point** for any wrong naming rule or format for submission



Submission

- Create a folder named **studentID_hw1**, and put all below files into the folder
  - **alu.v**
- Compress the folder **studentID_hw1** in a tar file named **studentID_hw1.tar**
  - Use **lower case** for the letter in your student ID. (Ex. r09943115_hw1)
- Submit to NTU cool

Discussion

電腦輔助積體電路系統設計 (EEE5022) › 討論 › [HW1]Discussion

111-1

首頁
課程資訊
課程內容
公告
作業
成績
討論
文件
頁面
成員
線上測驗
設定

[HW1]Discussion

陳定揚 (CHEN, TING-YANG)

所有班別

HW1相關問題在此討論，並請以下列格式發問，方便助按照每個問題回答

1. 問題一

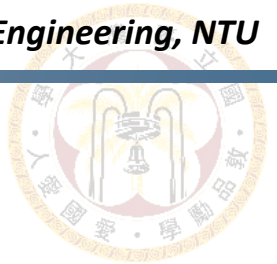
2. 問題二

...

另外，若需要截圖，請勿把自己的code截圖上傳，變成大家的參考答案，若違反將扣本次作業總分10分。

祝同學們學習順心

TA



References

- Reference for 2’complement:
  - https://en.wikipedia.org/wiki/Two%27s_complement
- To understand fixed-point representation with fraction number:
  - https://reurl.cc/D3xQnR