2018 DSD Machine Test

Student ID :__________________
Name:__________________

Final Outcome: (Write √ if you pass)

A. RTL Simulation:

☐ ALU results are correct! (30%)

☐ DFC results of "ALU+FIFO instruction" are correct! (15%)

☐ DFC results of "ALU+LIFO instruction" are correct! (15%)

☐ Use ALU sub-module in DFC design and results are all correct! (5%)

C. Illustrate Your Finite State Machine of DFC Design (20%)

B. Gate-Level Simulation:

☐ DFC results of "ALU+FIFO instruction" are correct! (5%)

☐ DFC results of "ALU+LIFO instruction" are correct! (5%)

☐ Use ALU sub-module in DFC design and results are all correct ! (5%)

Note: 1. Don't need to synthesize ALU design, Only synthesize DFC design
2. If you only pass one instruction, you can still synthesize and have partial scores.

Total time: 3 hours (14:00 ~ 17:00)

Objectives: RTL coding & simulation; logic synthesis; gate-level simulation

Note:

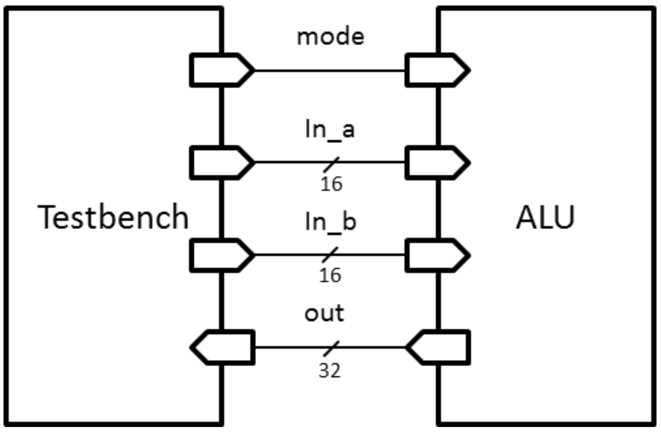
1. Due to limited licenses, please only open **one** NCverilog, nWave, and DC.
2. Please remember to fill you outcome, student ID, and name.
3. Please **return the examination paper**.

Test Overview:

Since some designs are complex, it uses to partition the design into several sub-modules. Therefore, we can separately verify the functions. In this test, you have to finish an ALU sub-module and apply the sub-module to implement Data Flow Control (DFC) design. The test includes two parts, which part I is ALU design and part II is DFC design, respectively.

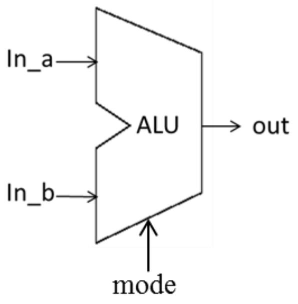
Part I. ALU Design

1-1 Definition



In part I, you have to design a simple ALU. The ALU need to support two operations: **unsigned addition** and **signed addition**.

The two operation modes are defined below:



Mode	Description
0	Unsigned Addition
1	Signed Addition

a. ***Unsigned Addition*** : inputs and output are all **unsigned** value. (in 2's complement form)

$$\mathbf{out = in\_a + in\_b}, \quad (in\_a , in\_b) \in [0, 255]$$

b. ***Signed Addition*** : inputs and output are all **signed** value. (in 2's complement form)

$$\mathbf{out = in\_a + in\_b}, \quad (in\_a, in\_b) \in [-128, 127]$$

1-2 Specification

The input/output pins are defined in the following:

Signal name	**Input/output**	**Bit width**	**Description**
in_a	Input	8	Unsigned Addition: Unsigned Signed Addition: Signed
in_b	Input	8	Unsigned Addition: Unsigned Signed Addition: Signed
mode	Input	1	1: Signed Addition 0: Unsigned Addition
out	Output	9	Unsigned Addition: Unsigned Signed Addition: Signed

The ALU design is pure combinational. When the ALU get inputs, it should give the correct output immediately in RTL simulation. Since the output contains 9 bits, **no overflow condition** will happen. All you need to care is the signed/unsigned of inputs and output (in 2's complement format). Since the ALU is the pure combinational circuit, no timing diagram will be provided.

1-3 Provided Files

File Name	**Description**
ALU.v	I/O declaration template.
ALU_tb.v	Testbench of ALU design.
ALU_a.dat	Test patterns of in_a.
ALU_b.dat	Test patterns of in_b.
golden_ALU.dat	Answer of ALU.

1-4 Requirements: RTL Simulation

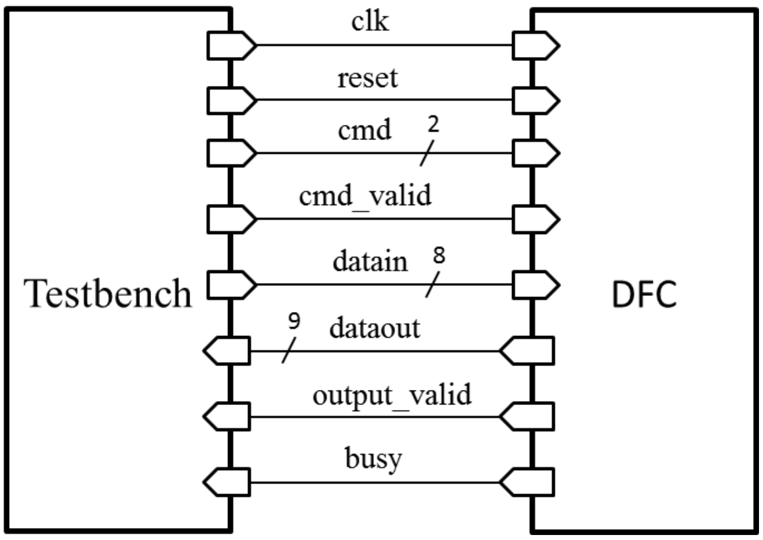
You need to finish Verilog RTL and check its functional correctness. Here are the instructions you need to run: (correct code will see **Congradulation!**)

```
ncverilog    +access+r    ALU_tb.v    ALU.v
```

[Note: You don't need to synthesize the ALU design.]

Part II. Data Flow Controller (DFC) Design

2-1 Definition



In part II, you have to design the data flow control, which will execute **signed addition** to input vector **a** and **b**. Since **a** and **b** vector contains four elements, the process can be formulated as:

$$\mathbf{y_1 = a_1 + b_1\,;}$$
$$\mathbf{y_2 = a_2 + b_2\,;}$$
$$\mathbf{y_3 = a_3 + b_3\,;}$$
$$\mathbf{y_4 = a_4 + b_4\,;}$$

Therefore, you are may use **four** ALU modules (designed in part I) as the components for computation. In other words, you should get the computation result through the submodules ALU. **If you do not use ALU modules in your top module (DFC.v), some scores will not be received.**

On the other hand, FIFO and LIFO methods are accounting techniques used in managing data output. For FIFO (first-in, first-out) operation, the schedule of system output is $\mathbf{y_1 \rightarrow y_2 \rightarrow y_3 \rightarrow y_4}$**.** On the other hand, for LIFO (last-in, first-out) operation, the schedule of system output is $\mathbf{y_4 \rightarrow y_3 \rightarrow y_2 \rightarrow y_1}$**.**

In the DFC, a command signal is applied to identify the instruction. The command includes the following:

Cmd	Description
0	Load Data
1	ALU+FIFO
2	ALU+LIFO

2-2 Specification

The input/output pins are defined in the following:

Signal Name	I/O	Width	Simple Description
clk	I	1	The clock of system, all signal timing are related to the rising edge of clk.
reset	I	1	The reset is an **active high asynchronous signal**.
cmd	I	2	Operation mode. (3 modes) **When cmd_valid=1 and busy=0, cmd is valid.**

cmd_valid	I	1	Input data enable signal:
			cmd_valid=0(disable)
			cmd_valid=1(enable)

datain	I	8	8-bit input data.
busy	O	1	The busy signal of system. **busy=0 (Host can input cmd)** **busy=1 (Host cannot input cmd)**
dataout	O	9	8-bit output data.
output_valid	O	1	Output control signal: **output_valid=0 (invalid output data)** **output_valid=1 (valid output data)**

2-3 Timing Diagram

[Load Data]

Fig. 1 shows the timing diagram of Load Data command. After the testbench sense the "busy=1'b0", the testbench will give command in the following cycle. Signal "cmd" accompany with "cmd_valid" is asserted.

Then the testbench will serially input 8 data in 8 cycles (**Each data takes a cycle**). The input data is **$a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow b_4$**, respectively, and all data is **8bits signed**. All the data should be stored in buffer for calculating other command. After finish load data command after 8 cycles, "busy" should be set to 1'b0. Hence, the testbench will input next command.

[Note: the input data are sent at negative edge and testbench will check your answer also at negative edge.]

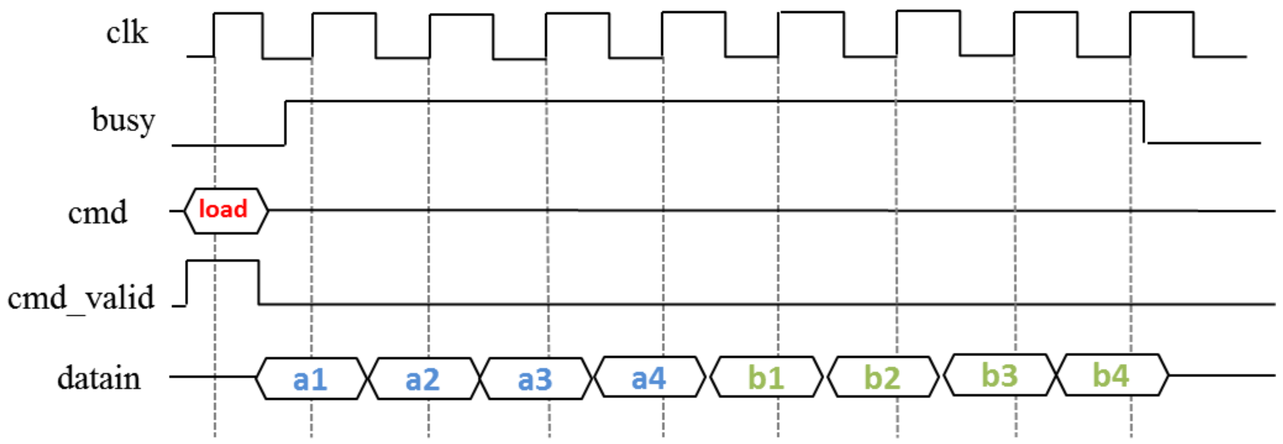


Fig. 1. Timing diagram of Load Data command.

[ALU+FIFO]

Fig. 2 shows the timing diagram of ALU+FIFO command. After the testbench sense the "busy=1'b0", the testbench will give command in the following cycle. Then, the system should calculate y1~y4 base on loaded data.

After calculating process is accomplished, system should **output data in $y_1 \rightarrow y_2 \rightarrow y_3 \rightarrow y_4$ schedule** and each data takes one cycles. Besides, "output_valid" should be set to 1'b1 to identify output of DFC is valid. After finish load data command after 8 cycles, "busy" should be set to 1'b0. Hence, the testbench will input next command.

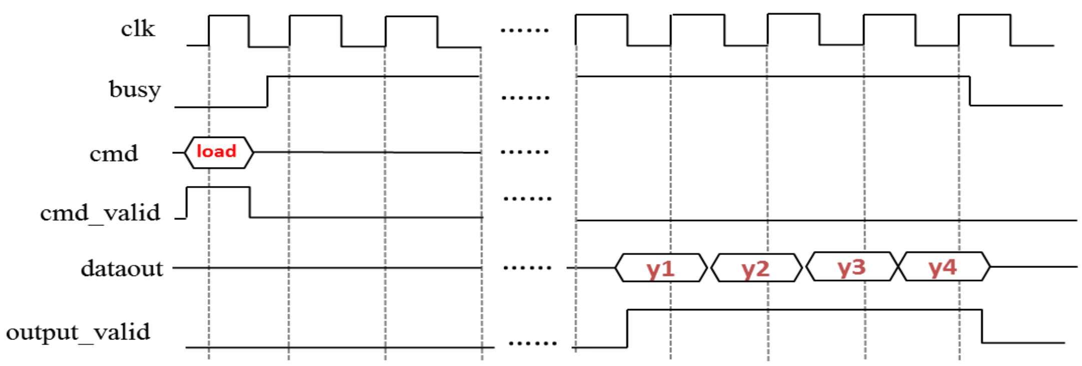


Fig. 2. Timing diagram of ALU+FIFO command.

[ALU+LIFO]

Fig. 3 shows the timing diagram of ALU+LIFO command. All control signals perform the same as ALU+LIFO command, but the results of ALU should **be outputted in $y_4 \rightarrow y_3 \rightarrow y_2 \rightarrow y_1$ schedule**.

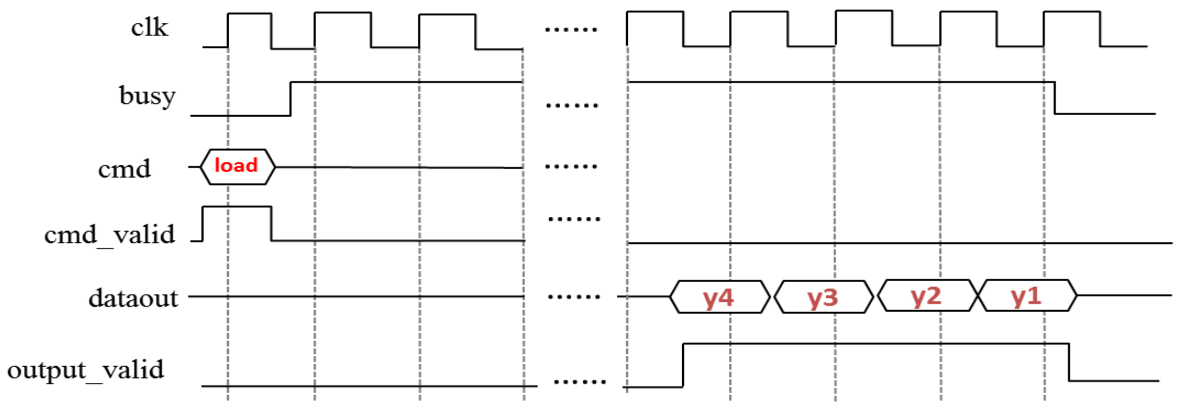


Fig. 3. Timing diagram of ALU+LIFO command.

2-3 Provided Files

File Name	Description
ALU.v	You should **copy ALU.v form Part I**.
DFC.v	I/O declaration with some note template. You can modify the content if you don't need it.
DFC_tb.v	Testbench of DFC design.
datain.dat	Test patterns of input data.
cmd.dat	Test patterns of command.
golden_DFC.dat	Answer of DFC.
DFC_DC.sdc	**Constraints for logic synthesis**
tsmc13.v	Technology file. ***DO NOT download this file !***
.synopsys_dc.setup	Environment settings of DC.

2-4 Requirements

[RTL Simulation]

You need to finish Verilog RTL and check its functional correctness. Here are the instructions you need to run: (correct code will see **Congradulation!**)

ncverilog +access+r DFC_tb.v DFC.v ALU.v (if you use ALU sub-module)

or

ncverilog +access+r DFC_tb.v DFC.v (if you do not use ALU sub-module)

After finishing the simulation, you may check your code with nLint to make sure that all syntax you use is legal and synthesizable. Check-report of the nLint cannot have any errors, but some minor warnings (such as have no driver or fan-out number) are ok!

[Synthesis]

After passing RTL simulation, you have to run the synthesis tool (Design Vision) with provided script:
Step 1: Open Design Compiler (Remember to check the .synopsys_dc.setup file.)
Step 2: Read in design file and set top module being 'DFC'. (**current_design DFC**)
Step 3: Set constraints by DFC_DC.sdc
Step 4: Compile your design
Step 4: Check if timing slack is "MET"
Step 5: **write_sdf -version 2.1 DFC_syn.sdf**
Step 6: **write -f verilog -hierarchy -output DFC_syn.v**
Step 7: **write -f ddc -hierarchy -output DFC_syn.ddc**

[Gate-Level (Netlist) Simulation]

Finally, run the provided testbench to test your netlist:

ncverilog +access+r DFC_tb.v DFC_syn.v -v tsmc13.v +define+SDF

3. Grading Policy (Read this Part Carefully)

[RTL Simulation]

A. ALU simulation results are correct! (30%)
B. DFC simulation results of "ALU+FIFO instruction" are correct! [No error at pattern 1~4] (15%)
C. DFC simulation results of "ALU+LIFO instruction" are correct! [No error at pattern 4~7] (15%)
D. Use ALU sub-module in DFC design and results are all correct! (5%)

[Illustrate Your Finite State Machine of ALU Design] (20%)

A. Draw your FSM of DFC design clearly. If the illustration is not clearly enough or correct, you will achieve partial points.

[Gate-Level Simulation]

A. DFC results of "ALU+LIFO instruction" are correct! (5%)
B. DFC results of "ALU+LIFO instruction" are correct! (5%)
C. Use ALU sub-module in DFC design and results are all correct! (5%)

[Note: You can synthesize the design even if you only accomplish one command and partial points will be given if the gate-level simulation results are correct.]

[Note: DO NOT directly output the result base on golden data. We have other test patterns. You will get 0%]

4. Submission Guideline

Please upload the design to the FTP as follows:

Address : **140.112.18.84**   Port : **1232**

Account : **DSD_STUDENT** Password : **dsd2018**

Please submit your design as follows in one .zip file with the naming convention:

StudentID_machine.zip (e.g., **B04901xxx_machine.zip**)

*Note that the file name must be exactly the same. Otherwise you will get zero score.

File Name	**Description**
ALU.v	RTL of ALU design.
DFC.v	RTL of DFC design.
DFC_syn.v	Netlist of DFC design.
DFC_syn.sdf	Timing information file.

If you want to modify your files, please use the following naming rule and upload a new zip file to the server.   Example: **B04901xxx_machine_v2.zip**

Appendix A --- ALU Test patterns

1. **Unsigned Addition** : 0 (in_a) + 0 (in_b) = 0
2. **Unsigned Addition** : 1 (in_a) + 127 (in_b) = 128
3. **Unsigned Addition** : 127 (in_a) + 127 (in_b) = 254
4. **Unsigned Addition** : 255 (in_a) + 255 (in_b) = 510
5. **Signed Addition** : 0 (in_a) + 0 (in_b) = 0
6. **Signed Addition** : 127 (in_a) + 127 (in_b) = 254
7. **Signed Addition** : -1 (in_a) + 1 (in_b) = 0
8. **Signed Addition** : -128 (in_a) + -128 (in_b) = -256

Appendix B --- DFC Test patterns

(a1, a2, a3, a4) = (0, 127, -1, -128)
(b1, b2, b3, b4) = (0, 127, 1, -128)
(y1, y2, y3, y4) = (0, 254, 0, -256)

Appendix: Tool list

NCverilog:	**source /usr/cadence/cshrc**
Verdi (nWave &, nLint -gui):	**source /usr/spring_soft/CIC/verdi.cshrc**
Design Compiler (dv &):	**source /usr/cad/synopsys/CIC/synthesis.cshrc**