1092 DSD Machine Test

Total time: 3 hours (14:20 ~ 17:20)

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

Note:

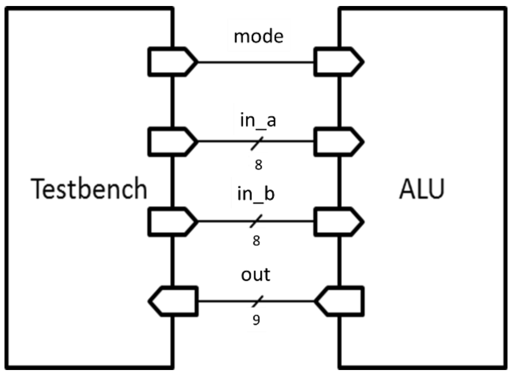
1. Due to limited licenses, please only open **one** NCverilog, nWave, and Design Compiler.
2. Please remember to write your outcome, student ID, and name.
3. Please **return the DSD_MachineTestFinalOutcome.docx**

Test Overview:

Since some designs are complex, it is popular to partition the design into several submodules. By doing so, we can separately and easily verify the functions of those submodules. In this test, you have to finish an ALU submodule and apply the submodule to implement **Data Flow Control (DFC) design**. The test includes two parts. Part I is ALU design and Part II is DFC design.

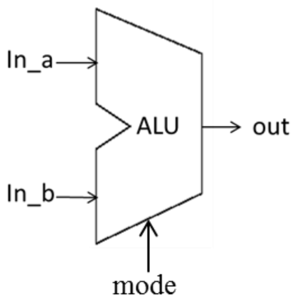
Part I. ALU Design

1-1 Definition



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

The two operation modes are defined in the following:



Mode	Description
0	Unsigned Addition
1	Signed Addition

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

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

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

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

1-2 Specification

The input/output ports 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 gets 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). The ALU is a 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 RTL and check its functional correctness. Here are the instructions you need to run: (correct code will see **Congratulations!**)

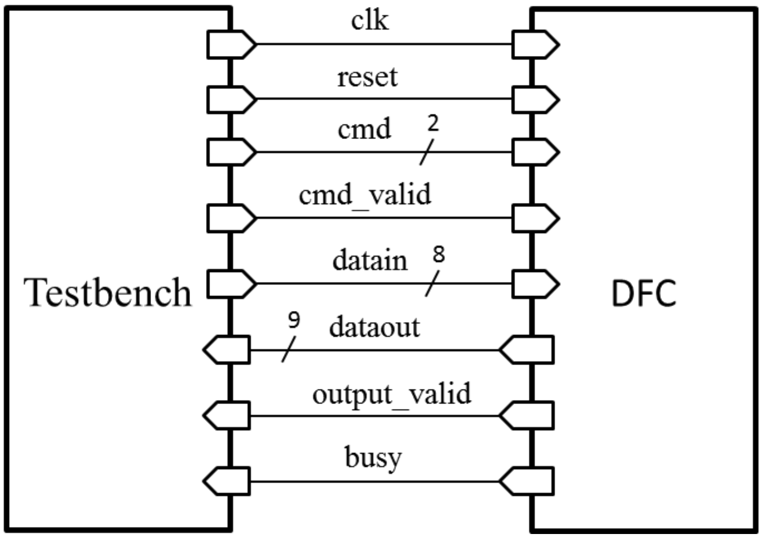
```
ncverilog  +access+r  ALU_tb.v  ALU.v
```

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

[Note: Don't modify the cycle time in ALU_tb.v.]

Part II. Data Flow Controller (DFC) Design

2-1 Definition



In part II, you have to design a data flow controller, which executes **signed addition** to input vector **a** and **b**. Vector **a** and **b** both 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 might need to use four ALU modules (designed in part I) as the components for computation. In other words, you should get the computation result from the submodules ALU. **If you do not use your own designed ALU modules in your top module (DFC.v), you will not get some credits.**

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

In the DFC, a command signal (cmd) 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 ports are defined in the following:

Signal Name	I/O	Width	Simple Description
clk	I	1	The clock of system, timing of all signals is 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	Valid signal for cmd:
			cmd_valid=0(disable)
			cmd_valid=1(enable)

datain	I	8	8-bits input data.
busy	O	1	The busy signal from DFC **busy=0 (Host can input cmd)** **busy=1 (Host cannot input cmd)**
dataout	O	9	9-bits output data.
output_valid	O	1	Valid signal for dataout: **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. When the testbench senses the "busy=1'b0", the testbench will give *Load Data* command in the following cycle. Signal "cmd" accompanies with "cmd_valid" is asserted.

After that, testbench will serially input 8 data in 8 cycles (**each data takes one cycle**). The order of input data is **a_1→a_2→a_3→a_4→b_1→b_2→b_3→b_4**, and all data is **8bits signed**. All the data should be stored in buffer for other commands usage. After 8 cycles finishing *Load Data* command, "busy" should be set to 1'b0 so that the testbench can input next command.

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

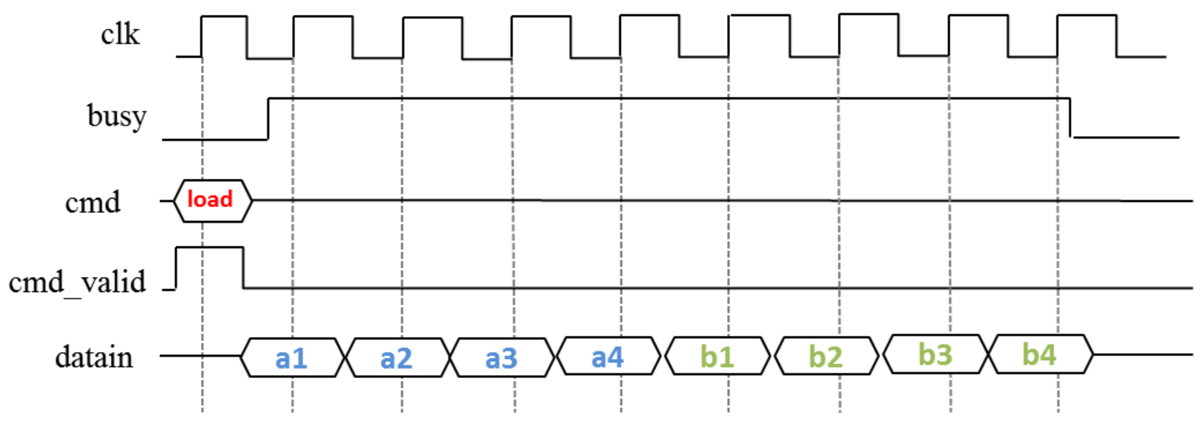


Fig. 1. Timing diagram of Load Data command.

[ALU+FIFO]

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

When calculating process is accomplished, system should **output data in order of y_1→y_2→y_3→y_4** and each data takes one cycle. Besides, "output_valid" should be set to 1'b1 at the same time to identify the output of DFC is valid. After finishing *ALU+FIFO* command, "busy" should be set to 1'b0 so that the testbench can input next command.

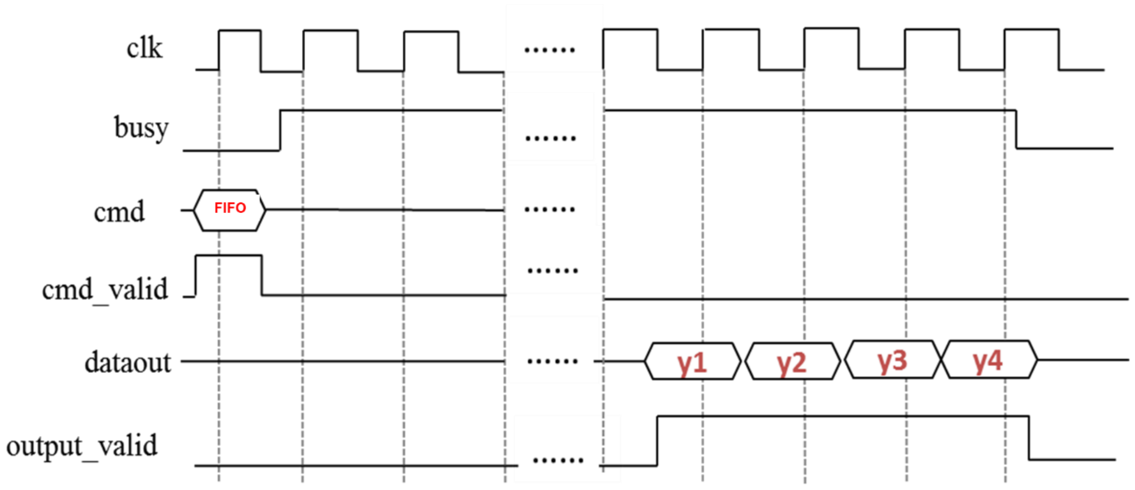


Fig. 2. Timing diagram of ALU+FIFO command.

[ALU+LIFO]

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

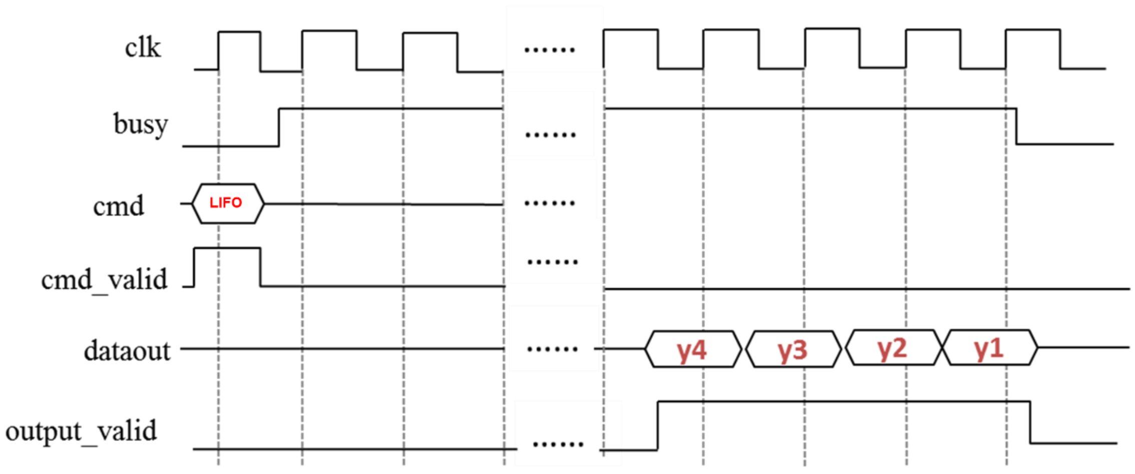


Fig. 3. Timing diagram of ALU+LIFO command.

2-3 Provided Files

File Name	**Description**
ALU.v	You should **copy ALU.v from Part I**.
DFC.v	I/O declaration with some notes. 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 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 RTL and check its functional correctness. Here are the instructions you need to run: (correct code will see **Congratulations!**)

ncverilog +access+r DFC_tb.v DFC.v ALU.v (if you use ALU submodule)

or

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

[Synthesis]
After passing RTL simulation, you have to run the synthesis tool (Design Compiler) to synthesize DFC:

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'

dc_shell > **read_file -format verilog ALU.v** (if you use ALU submodule)

dc_shell > **read_file -format verilog DFC.v**

dc_shell > **current_design DFC**

Step 3: Set constraints by DFC_DC.sdc

dc_shell > **source DFC_DC.sdc**

Step 4: Compile your design

dc_shell > **compile**

Step 5: Make sure your timing slack is "POSITIVE"

dc_shell > **report_timing > DFC_syn.timing_rpt**

Step 6: dc_shell > **write_sdf -version 2.1 DFC_syn.sdf**

Step 7: dc_shell > **write -f verilog -hierarchy -output DFC_syn.v**

Step 8: dc_shell > **write -f ddc -hierarchy -output DFC_syn.ddc**

[Note: You don't need to optimize DFC in this test. What you need to do is pass the given testbench with cycle time 10ns]

[Note: Make sure there is no latch in your design!]

[Note: If your slack is negative, which means your design doesn't fit timing constraint, you will not get any credits in gate-level simulation despite the fact that you pass the gate-level simulation.]

[Gate-Level (Netlist) Simulation]
Run the provided testbench to test your netlist and make sure there is no timing violation:

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 5~8] (15%)

D. Use your own ALU submodule 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 only get partial points.

[Gate-Level Simulation]

A. DFC results of "ALU+LIFO instruction" are correct and no timing violation! (5%)

B. DFC results of "ALU+LIFO instruction" are correct and no timing violation! (5%)

C. Use your own ALU submodule in DFC design and results are all correct and no timing violation! (5%)

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

[Note: DO NOT directly output results based on golden data. We have other test patterns. If you do so, you will get 0% in this test]

4. Submission Guideline

Please upload your **ZIP** file to CEIBA:

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

StudentID_machine.zip (e.g., **b04901xxx_machine.zip**)

[Note: The file name must be exactly the same. Otherwise, you will get 0% in this test]

File Name	**Description**
DSD_MachineTestFinalOutcome.docx	Outcome of Machine Test
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.
DFC_syn.ddc	DDC file

If you want to modify your files, please use the following naming rule and upload a new ZIP file to CEIBA.

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):	**source /usr/spring_soft/CIC/verdi.cshrc**
Design Compiler (dv &):	**source /usr/cad/synopsys/CIC/synthesis.cshrc**