NCTU-EE IC LAB – Fall 2023

Lab02 Practice

Design: Complex Number Calculator

Data Preparation

1. Extract test data from TA's directory:

   % tar xvf ~iclabTA01/Lab02.tar

2. The extracted LAB director contains:
   a. Exercise
   b. Practice

Design Description

Complex numbers are widely used in engineering field, such as communications, circuit designs, or waveform analysis. A complex number contains a real part a and an imaginary part b, which can be donated in a+bi. Now you are going to implement a complex number calculator, which can performs addition, subtraction and multiplication to two complex numbers a+bi and c+di. The arithmetic result e+fi will be: addition: (a+c) + (b+d)i, subtraction: (a-c) + (b-d)i, and multiplication:
(ac-bd) +(ad+bc)i. Try to implement the calculator as small as possible.

Inputs

1. You will receive a sequence of 8-bit **2's complement *signed numbers* IN**[7:0] with length 4, representing **a**, **b**, **c** and **d** in order sequentially. The input values' range is -128~127.
2. The sequence with length 4 is valid when **IN_VALID** is high.
3. **MODE** is valid when the ***first cycle*** of **IN_VALID** is high.

   MODE 0: **Addition**

   MODE 1: **Subtraction**

   MODE 2: **Multiplication**
4. There is ***only 1 reset*** before the first pattern, thus, your design must be able to reset automatically.
5. All inputs will be changed at clock ***negative*** edge.
6. The next input pattern will come in 1~3 cycles after **OUT_VALID** falls.

Outputs

1. You have to output a sequence of 17-bit ***2's complement signed numbers*** **OUT[16:0]** in consecutive 2 cycles, representing **e** and **f** respectively.
2. All outputs are synchronized at clock ***positive*** edge.
3. **OUT_VALID** should be low after initial reset.
4. **OUT_VALID** should not be raised when **IN_VALID** is high.
5. **OUT_VALID** is set to high when the output value is valid.
6. The test pattern will check whether your output sequence is correct or not at clock ***negative*** edge.

Specifications

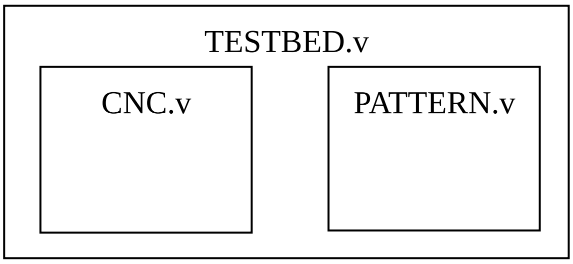
1. Top module name : **CNC** (File name : **CNC.v**)
2. Input pins: **clk, rst_n, IN_VALID, IN[7:0], MODE**
3. Output pins: **OUT_VALID, OUT[16:0]**
4. It is **synchronous** reset and **active-low** architecture.
5. All numbers are represented in **2's complement signed number** format, so you have to do sign extension whenever necessary.
6. The latency of your design in each pattern should not be larger than **100** cycles. The latency is defined in Fig.1.
7. The clock period is set to **6ns**, and both input and output delay are set to **3ns**.
8. The output loading is set to 0.05.
9. The synthesis result of data type cannot include any **LATCH** (in syn.log).
10. After synthesis, you can check CNC.area, CNC.timing and CNC.resource.
11. The slack in the end of CNC.timing should be **non-negative** and the result should be **MET**.

Note: If you want to use signed multiplication, both two inputs of the multiplier should be declared with **signed**.

Ex:

reg **signed** [3:0] A;
reg **signed** [3:0] B;
reg [3:0] C;
reg [7:0] OUT;
	always @* OUT=A*B; ← **signed multiplication**
	always @* OUT=A*C;
	always @* OUT={A[3], A[3:1]}*B;
	always @* OUT=A*{1'b0, A[2:0]}; ← **unsigned multiplication**

Block Diagram



Note

1. This practice is for practicing the finite state machine (FSM), thus some codes are given; try to complete the codes to achieve the correct result. However, you can implement the design with your own code.
2. Try to use as less hardware as possible.
   Hint: Accumulator can be reused.
3. Template folders and reference commands:
   01_RTL/ (RTL simulation) **./01_run**
   02_SYN/ (Synthesis) **./01_run_dc**
   (Check the design which contains **Latch** or not in **syn.log**)
   (Check the design's timing in /Report/CNC.timing)
   03_GATE_SIM/ (GL simulation) **./01_run**
   You can key in **./09_clean_up** to clear all log files and dump files in each folder

4. Sample waveform:

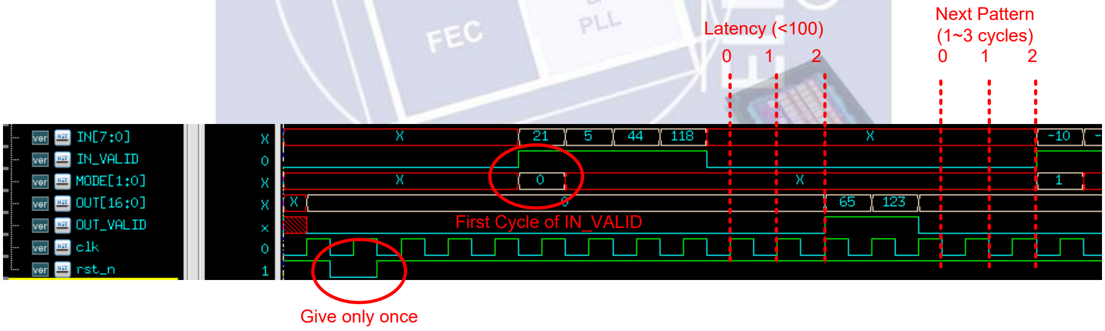


Fig.1