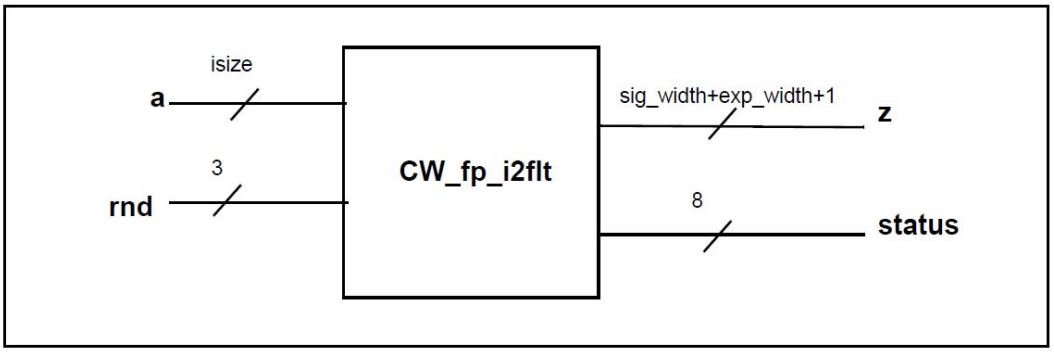
CSIE DIC Lab 2024

Final Exam. Pipeline floating-point divider

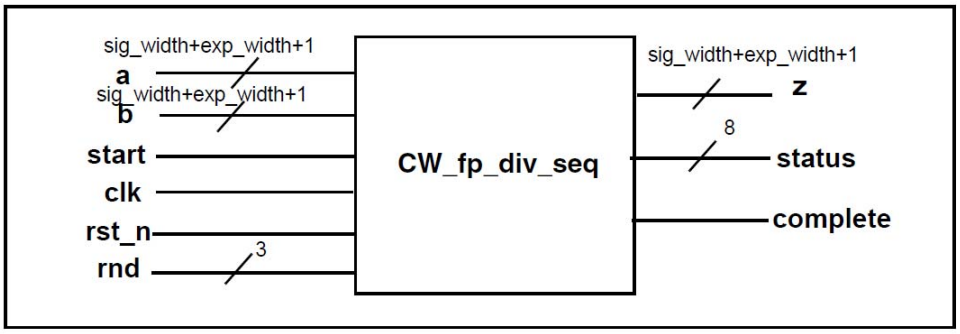
Description

In Final Exam, the testing module will randomly generate two 16-bit integers, ***a*** and ***b***, each time. You are required to first convert the inputs ***a*** and ***b*** into the IEEE 754 single-precision 32-bit floating-point format, then implement a 32-bit pipeline floating-point divider to compute the result of ***a÷b***. Finally, the result should be transmitted through an 8-bit output port over 4 cycles.



This component converts an integer `a` to a floating-point number, producing floating-point output `z` and status flags, according to the rounding mode `rnd`.

(a) Integer to floating-point converter



This component is a floating-point sequential divider component that divides two floating point numbers, `a` and `b`, producing a floating-point outputs `z` and `status` according to the rounding mode `rnd`.

(b) Floating-point sequential divider

Figure 1: Genus ChipWare IP Components.

Figure 1 illustrates the two IPs required for this Final Exam: **CW_fp_i2flt** and **CW_fp_div_seq**. **CW_fp_i2flt** is used to convert integers into the FP32 floating-point format, while **CW_fp_div_seq** performs FP32 floating-point division operations. This IP enables multi-cycle computation, distributing the calculation over several cycles, and signals completion when the division is done. The computed results need to be

transmitted through an 8-bit output port over 4 cycles.

For these two IPs, the rounding mode is set to **3'b000**, which corresponds to IEEE rounding to the nearest, even. This means rounding to the nearest representable value; if two values are equally near, the output result will have an even significand. Additionally, be aware that once the inputs for **CW_fp_div_seq** are set up, keeping the ***start*** signal at a high level for one cycle and then lowering it will initiate the computation of the division result in this IP. Furthermore, the ***complete*** signal will indicate whether the division calculation has been completed.

Specification

1. Top module name: CHIP
2. Top module filename: CHIP.v
3. Input/output definition

Signal Name	Direction	Bit Width	Description
clk	Input	1	Clock signal
rst_n	Input	1	Asynchronous active low reset signal
enable	Input	1	When the inputs **a** and **b** are updated, it will be pulled to a high level.
a	Input	16	Signed integer in two's complement representation (dividend).
b	Input	16	signed integer in two's complement representation (divisor).
ready	Output	1	When beginning to output the answer, set it to a high level, and return it to a low level once the output is complete.
out	Output	8	Divide the 32-bit calculation result into four cycles, starting from the lowest 8 bits and sequentially outputting to the higher bits.

4. All input signals are **synchronized at the clock rising edge**.
5. The reset scheme is an **active-low asynchronous reset**.
6. Each time when **enable=1**, a new set of inputs ***a*** and ***b*** are entered into the CHIP module. The timing diagram of the signal input is shown in Figure 2.
7. After completing the FP32 division calculation, set **ready=1** for 4 continuous cycles, and sequentially transmit the calculation result starting from the lowest 8 bits over four consecutive cycles.
8. In Figure 2, **your_out[7:0]** is the output signal from the CHIP module. After

outputting the FP32 division result over four cycles, **real_z** in the diagram represents the division result displayed as a real number.

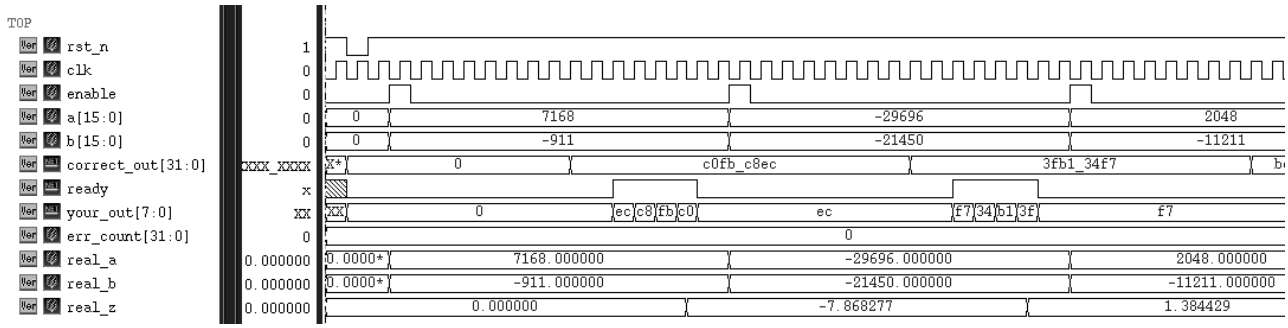


Figure 2: Input and output timing diagram

9. Clock period **MUST** be **0.54ns**. Thus your design should operate correctly at this clock speed without any timing violation.

Final Exam Instructions

1. Extract LAB data from TA's directory:
   % tar xzvf ~dicta/final_exam.tar.gz
2. The extracted LAB directory (**final_exam**) contains the following:
   a. **CHIP.v** :Empty design module
   b. **TEST_CHIP.v** :Design test bench for RTL simulation
   c. **TEST_CHIP_gate.v** :Design test bench for Gate-level simulation
   d. **SYN_RTL.sdc** :Design constraint file for logic synthesis
   e. **CHIP.rc** :nWave saved signal file
3. Write your RTL code in a synthesizable coding style:
   % gedit CHIP.v &
   or % gvim CHIP.v &
   or % joe CHIP.v
   or % vim CHIP.v
4. Run RTL simulation:
   % ncverilog –f run.f
5. Open waveform:
   % nWave &
   Choose **File → Open,** then select **"CHIP.fsdb"** and press **OK**
6. Restore signal recorded for debugging:
   Choose **File → Restore signal**, then select **"CHIP.rc"** and press **OK**
7. Meaning of signals in TOP group:
   real_a : ***a*** displayed as a real number
   real_b : ***b*** displayed as a real number
   real_z : division result displayed as a real number

error_count : accumulated error number

Demo:

1. **Demo1:** You must complete the gate-level simulation at **clock period = 0.54ns**, ensuring not only that the circuit functions correctly but also that there are no timing violations.
2. **Demo2:** You must complete the Auto Placement and Routing (APR) of the final exam design at **clock period = 0.54ns** and pass the Calibre DRC/LVS check.
3. **Demo3:** Besides, your design must operate at **clock period = 0.54ns** without any timing violation in post-layout gate-level simulation.

Grade:

Demo1: Correct Gate-level simulation (20%)
Demo2: Complete APR and Calibre DRC/LVS without error (50%)
Demo3: Correct post-layout gate-level simulation (30%)

Note: Only the first 50% of students who complete Demo1 to Demo3 can receive the full score of 100 points. Those who finish afterward will have their scores multiplied by 80%. At the end of the exam, students who have not completed Demo1 to Demo3 will have their scores calculated based on the points allocated for Demo1 to Demo3, and these scores will also be multiplied by 80%.

Ps. PDF password for other lab files: **ms112dicdic**