NYCU-ECE DCS-2025

HW02

Design: GCD Compression

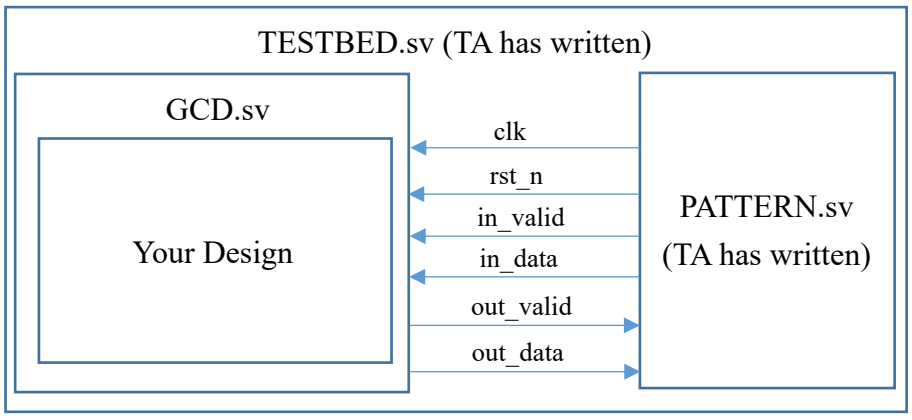
Learning Objectives

1. Separate sequential and combinational block.
2. How to design a Finite State Machine (FSM) to control the data flow.
3. Handling and organizing data efficiently.
4. Getting used to shifter and counter.

Data Preparation

1. Extract the files needed from TA's directory:
   % tar -xvf ~dcsTA01/HW02.tar
2. The extracted files contain:
   a. 00_TESTBED/
   b. 01_RTL/
   c. 02_SYN/
   d. 03_GATE/
   e. 09_UPLOAD/

Block Diagram



Design Description

GCD compression involves compressing an array A containing 2n numbers into (n-1) numbers as follows:

First, arbitrarily choose two numbers from array A and remove them, then select two numbers from A and add their sum to array B. Repeat this process until all numbers in array A have been removed. And the greatest common divisor (GCD) of all elements in the final array B should be greater than 1.

Example: n=4

Array A: [1, 2, 3, 4, 5, 6, 7, 8] => discard 1,3

Array B: [2+4, 5+7, 6+8] = [6, 12, 14], GCD = 2

For this assignment, you are required to design and implement a circuit for GCD compression. Details are as follows:

INPUT:

The value of n is fixed at 4, which means the input length is fixed at 8 numbers. Each number is a 4-bit unsigned integer. When in_valid is raised, one number will be input through in_data per cycle, with 8 consecutive cycles of input.

OUTPUT:

When you're ready to output the results, raise out_valid and output the results through out_data, with one number per cycle. The first three cycles output the three processed numbers, and the fourth cycle outputs the GCD of these three numbers. After output is complete, both out_valid and out_data must return to zero. The four output cycles must be consecutive.

Additional information

To find the Greatest Common Divisor (GCD), you can use the Euclidean algorithm, follow these steps:

1. Take the two numbers a and b (assuming $a \geq b$).
2. Divide a by b to get quotient q and remainder r, such that $a = b \times q + r$.
3. If the remainder $r = 0$, then b is the greatest common divisor.
4. If the remainder $r \neq 0$, then set $a = b$, and $b = r$, then repeat steps 2-4.
5. Repeat the above steps until the remainder becomes 0.

Let's look at an example: suppose we want to find the greatest common divisor of 48 and 18.

1. a = 48, b = 18
2. 48 ÷ 18 = 2 remainder 12, i.e., 48 = 18 × 2 + 12
3. remainder ≠ 0, new a = 18, new b = 12

Repeat:

- 18 ÷ 12 = 1 remainder 6, i.e., 18 = 12 × 1 + 6
- remainder ≠ 0, new a = 12, new b = 6

Repeat:

- 12 ÷ 6 = 2 remainder 0, i.e., 12 = 6 × 2 + 0
- remainder = 0, stop

Since the remainder is 0, b = 6 is the greatest common divisor of 48 and 18.

Inputs

Signal name	Number of bit	Description
clk	1	10ns clock signal
rst_n	1	Asynchronous negedge reset signal
in_valid	1	Pulled high during input
in_data	4	An array of 8 numbers are provided sequentially over 8 clock cycles when in_valid is high. (Data range: 0 ~ 15)

Outputs

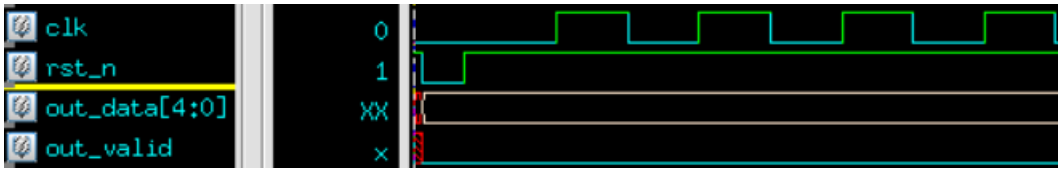
Signal name	Number of bit	Description
out_valid	1	Pulled high during out_data output (reset required)
out_data	5	When out_valid is high, output the 3 compressed numbers in 3 cycles, and then output the GCD of them in 1 cycle. (reset required)

Specifications

1. Top module name: **GCD**(File name : **GCD.sv**)
2. After asynchronous active-low reset, all output signals must be reset to zero.
3. Output must be produced within 100 cycles after the Input is complete.
4. Output must be exactly 4 cycles, no more, no less.
5. The next input will be sent 1-4 cycles after out_valid is pulled down.
6. The 02_SYN result must not have any errors and must not contain any latches.
7. The slack in the timing report must be non-negative and the result must be "Met".
8. Gate level simulation must not have any timing violation.
9. Clock period is 7 ns.
10. Input delay = 0.5 * clock period; Output delay = 0.5 * clock period
11. The design must actually implement the required functionality. Do not design specifically for the test patterns, such as determining it's the nth pattern and directly setting the output. Such designs will be judged as fail during the demo.
12. Do not use *error*, *latch*, *congratulation* or *fail* as names for logic / wire / reg / submodule / parameter , otherwise the demo result will be fail.
    Note: * represents any symbol before or after the word. For example: error_test is prohibited.
13. Combinational blocks and sequential blocks should be separated.
14. Do not use for loops in your design. ◦

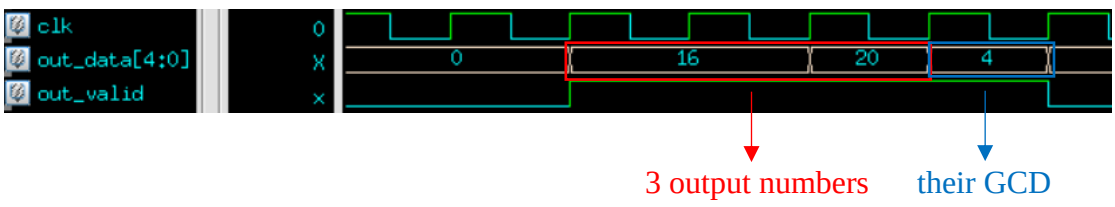
Example waveform

Reset all output after asynchronous and active-low reset signal



8 cycles for input 8 numbers in each pattern

Output result in each pattern



Upload file

1. Files to be submitted: SystemVerilog Code and Report, a total of two files. Please upload them to new E3.
2. Code filename: GCD_dcsxxx.sv, where xxx is your server account.
3. Report filename: report_dcsxxx.pdf, where xxx is your server account.
4. 1de deadline: 3/26 23:59 / 2de deadline: 4/2 23:59.
5. Incorrect filenames will result in a 5-point deduction.

Grading policy

1. Pass the RTL& Synthesis simulation: 70%
2. performance: 25%
   Ranking formula: total latency * area
4. Report: 5%

Note

Template folders and reference commands:

1. 01_RTL/ (RTL simulation) → **./01_run**
2. 02_SYN/ (synthesis) → **./01_run_dc**
3. 03_GATE/ (gate-level simulation) → **./01_run**

Please write your report concisely and to the point, not exceeding two A4 pages, and include the following content:

1. Describe your design method, including but not limited to how to accelerate (reduce critical path) or reduce area.
2. Based on the above, draw your architecture diagram (Block diagram).
3. Your thoughts, you can write about either the assignments or course content.
4. Difficulties encountered and how you solved them.