# GCD Processor Design

Assignment 3 - EE593

Shri Janani Senthil (B20232)
School of Computing and Electrical Engineering
IIT Mandi
Himachal Pradesh, India
Email: b20232@students.iitmandi.ac.in

Anisha Sharma (B20029)
School of Computing and Electrical Engineering
IIT Mandi
Himachal Pradesh, India
Email: b20232@students.iitmandi.ac.in

## I. Euclid's algorithm for finding GCD

Euclid's algorithm is a simple and efficient method for finding the greatest common divisor (GCD) of two positive integers. To apply Euclid's algorithm, we repeatedly divide the larger number by the smaller number, and take the remainder as the new smaller number. We continue this process until the remainder is zero. The last non-zero remainder that we obtained is the GCD of the original two numbers. Since division is repeated subtraction, we may use the algorithm using subtraction to find GCD.

The algorithm for finding the GCD of two positive integers by repeated subtraction is straightforward. Here are the steps:

1) Let $A$ and $B$ be the two positive integers whose GCD is to be found.
2) If $A$ and $B$ are equal, then their GCD is $A$ (or $B$, as they are equal). Otherwise, proceed to the next step.
3) Subtract the smaller number from the larger number, and let the result be the new larger number. In other words, if $A > B$, then let $A = A - B$; otherwise, let $B = B - A$.
4) Repeat step 2 and step 3 until $A$ and $B$ become equal.
5) The final value of $A$ (or $B$, as they are equal) is the GCD of the original two numbers.

## II. About the processor design

Given two inputs of 16-bits length, the GCD processor presented calculates the greatest common divisor using the Euclidean Algorithm. The inputs are loaded into FIFO and the output is obtained from a separate output FIFO. The schematic depicted in Fig. 1 shows the overall schematic of the GCD Processor.

The top module instantiates two FIFO modules, `in1_f1` and `in2_f2`, to store the two input numbers `in_1` and `in_2`, respectively. It also instantiates a GCD module g1 to compute the GCD of the two input numbers and then a FIFO module `out_f3` to store the GCD value computed.The top module has the following inputs and outputs:

- `clk`: a clock signal
- `reset`: a reset signal
- `wr_1, rd_1, wr_2, rd_2, wr_out, rd_out`: write and read enable signals for each FIFO and the output FIFO

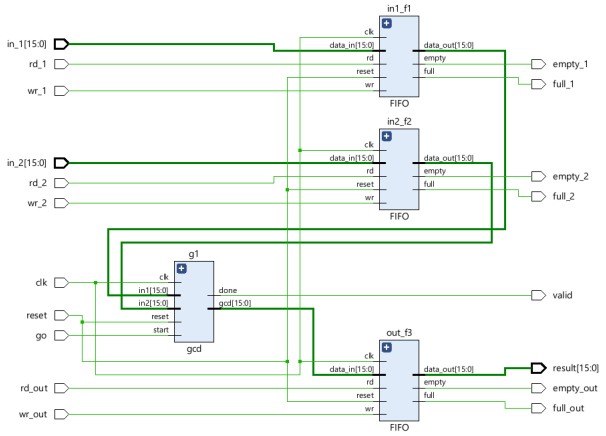


Fig. 1: GCD Processor - Overall Schematic

- `go`: a signal to start the GCD computation
- `in_1, in_2`: two 16-bit input numbers to be processed
- `valid`: a signal indicating that the GCD computation is complete
- `full_1, full_2, full_out`: signals indicating that the respective FIFOs are full
- `empty_1, empty_2, empty_out`: signals indicating that the respective FIFOs are empty
- `result`: a 16-bit signal containing the GCD result

The `in1_f1` FIFO has inputs `data_in` and `wr` and output `data_out` to write and read data to/from the FIFO, respectively. The `in2_f2` FIFO has similar inputs and outputs. The `g1` module has inputs `in1, in2, and start` to take in the two input numbers and start the GCD computation, and outputs `gcd` and `valid` to output the GCD result and signal when the computation is complete. Depicted in Fig. 2 is the schematic of the FIFO implemented. Further, Fig. 3 depicts the module that computes GCD.

The `empty` and `full` signals of the FIFOs are set in the FIFO module itself. The top module connects the `full_1, empty_1, full_2, empty_2, full_out` and `empty_out` signals to the `full` and `empty` outputs of the respective FIFOs instantiated within the top module.

The top module uses the FIFOs to communicate data between the input ports and the GCD circuit. The `wr_1` and

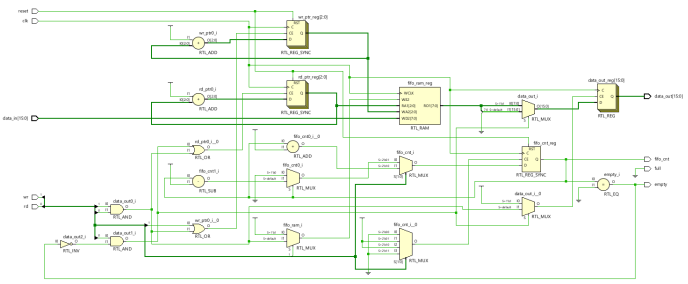

Fig. 2: FIFO Implementation

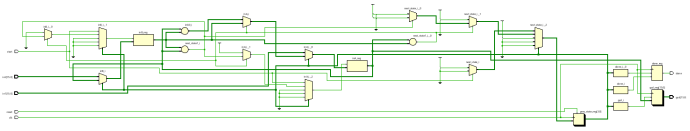

Fig. 3: GCD Processor Schematic

`rd_1` signals control the write and read operations of the `in1_f1` FIFO, while the `wr_2` and `rd_2` signals control the write and read operations of the `in2_f2` FIFO. The `wr_out` and `rd_out` signals control the write and read operations of the output FIFO.

The top module waits for the `go` signal to be asserted to start the GCD computation. When the computation is complete, the GCD result is stored in the `out1` variable, which is then written to the output FIFO. The result output of the top module is connected to the `output` of the output FIFO.

In summary, the top module combines two input FIFOs and a GCD circuit to compute the GCD of two input numbers. It uses a third output FIFO to store the GCD result and communicate it to the output. The module also provides status signals for the FIFOs and the GCD computation.

## III. Register values in GCD Calculation

If the input values are given as 27 and 15, the steps would be as follows:

- Let's begin by finding the larger and smaller number. Since, 27 is larger than 15, so we will start with 27.
- Now we subtract 15 from 27: $27 - 15 = 12$.
- Since, 12 is smaller than 15, so we will swap the numbers and subtract 12 from 15: $15 - 12 = 3$.
- Since, 3 is the smaller number, so we will subtract it from 12: $12 - 3 = 9$.
- Now we subtract 3 from 9: $9 - 3 = 6$.
- And finally, we subtract 3 from 6: $6 - 3 = 3$.
- At this point, we have reached a situation where both numbers are equal, and the common number is 3. Therefore, the GCD of 27 and 15 is 3.

These values would be reflected in each cycle as shown in Table 1.

| Cycle Number | In1 | In2 |
|---|---|---|
| 0 | 27 | 15 |
| 1 | 12 | 15 |
| 2 | 12 | 3 |
| 3 | 9 | 3 |
| 4 | 6 | 3 |
| 5 | 3 | 3 |

TABLE I: Cycle-wise register values

## IV. Results of Synthesis

### A. Power Report

Power consumption in an IC Chip is of two forms - static and dynamic. Dynamic power is the power dissipated during switching of the outputs. Static power is dissipated when the device is idle.

Static power consumption refers to the power consumed by an IC chip even when it is in a steady state and not switching between different logic states. This static power is a result of the leakage current that flows through the transistors and is present regardless of whether the circuit is actively processing data or not. To reduce static power consumption, techniques like transistor sizing, threshold voltage optimization, and circuit-level techniques like sleep mode can be employed.

Dynamic power consumption refers to the power consumed by an IC chip when it switches between different logic states, i.e., from high to low or vice versa. This switching requires the flow of current, which results in energy consumption. Dynamic power consumption can be reduced by optimizing circuit design, reducing clock frequency, or employing power gating techniques.

Summary of power report is depicted in Fig. 4.

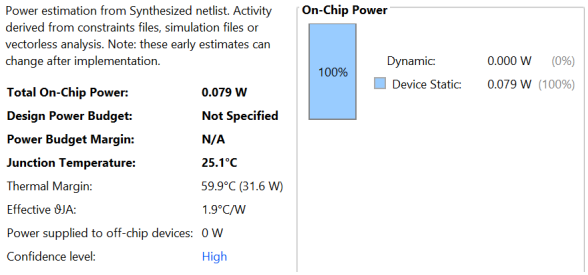


Fig. 4: Power Consumption according to Vivado Synthesis

### B. Timing Analysis

Determining the performance and suitability of a GCD module for a specific application is significantly influenced by its operating frequency. The frequency of a GCD module is measured in MHz or GHz and is indicative of the rate at which it can perform GCD calculations.

Factors that affect the frequency of a GCD module include the complexity of the GCD algorithm, the number of registers used, and the speed of the clock signal. Generally, a more complicated algorithm and a greater number of registers will result in lower frequencies, while a simpler algorithm and fewer registers will lead to higher frequencies.

To enhance the frequency of a GCD module, optimization of the GCD algorithm can be carried out to minimize the number of operations required and reduce data movement between registers. Another approach is to increase the clock signal speed, although this may result in higher power consumption.

The frequencies obtained pre-STA and post-STA are depicted in Table 2.

| S.No | Type | Maximum Frequency (in $MHz$) |
|---|---|---|
| 1 | Pre-STA | 565.289 |
| 2 | Post-STA | 529.168 |

TABLE II: Maximum Frequencies Computed

Pre-STA analysis evaluates design timing under ideal conditions, assuming instantaneous signal propagation and no interconnect delays. However, in reality, interconnects, buffers, and other design components' propagation delay can greatly affect timing. Pre-STA log is depicted in Fig. 5.

```
Path DFFPOSX1_102/CLK to DFFPOSX1_183/D delay 1752.82 ps
      0.0 ps      clk_bF_buf37:    CLKBUF1_5/Y -> DFFPOSX1_102/CLK
    223.3 ps  in2_f2_wr_ptr_0_: DFFPOSX1_102/Q ->   NAND2X1_86/A
    652.9 ps             _582_:   NAND2X1_86/Y ->     INVX2_32/A
    851.9 ps             _614_:     INVX2_32/Y ->   NAND2X1_93/B
   1185.8 ps             _699_:   NAND2X1_93/Y ->     OR2X2_13/A
   1446.2 ps             _700_:     OR2X2_13/Y ->  OAI21X1_184/A
   1537.2 ps             _576_:  OAI21X1_184/Y -> DFFPOSX1_183/D

   clock skew at destination = 20.0908
   setup at destination = 195.495

Computed maximum clock frequency (zero margin) = 565.289 MHz
```

Fig. 5: Pre-STA log

Post STA analysis considers all factors such as process variation, interconnect parasitics, and clock skew to provide an accurate assessment of design timing performance. Post-STA log is depicted in Fig. 6.

```
Path DFFPOSX1_102/CLK to DFFPOSX1_177/D delay 1838.19 ps
      0.6 ps      clk_bF_buf37:    CLKBUF1_5/Y -> DFFPOSX1_102/CLK
    223.6 ps  in2_f2_wr_ptr_0_: DFFPOSX1_102/Q ->   NAND2X1_86/A
    687.4 ps             _582_:   NAND2X1_86/Y ->     INVX2_32/A
    894.4 ps             _614_:     INVX2_32/Y ->   NAND2X1_93/B
   1250.1 ps             _699_:   NAND2X1_93/Y ->     OR2X2_13/A
   1524.8 ps             _700_:     OR2X2_13/Y ->  OAI21X1_172/A
   1619.5 ps             _570_:  OAI21X1_172/Y -> DFFPOSX1_177/D

   clock skew at destination = 21.0006
   setup at destination = 197.638

Computed maximum clock frequency (zero margin) = 529.168 MHz
```

Fig. 6: Post-STA log