GCD Processor Design
EE593

***Abstract*—This report presents the design and implementation of a 16-bit GCD calculator and processor in the context of Low Power VLSI Design. The GCD calculator module efficiently computes the greatest common divisor of two 16-bit numbers, utilizing FIFOs for input and output data handling. The design includes provisions for reset, start signal, and ready indication for result availability. Through proper synthesis and testing, the report evaluates the area, delay, and power characteristics of the implemented modules, aiming to optimize performance in VLSI applications.**

I. Introduction

GCD stands for Greatest Common Divisor, which is the largest positive integer that divides two or more numbers without leaving a remainder. In the context of VLSI design, a GCD calculator is a module that computes the GCD of two input numbers efficiently. The Euclidean Algorithm is a fundamental approach for calculating the greatest common divisor (GCD) of two integers. It is based on the principle that the GCD of two numbers also divides their difference. This document explains the algorithm and demonstrates its application with an example.

FIFO stands for First-In-First-Out, which is a data structure used for handling data in a sequential manner. In VLSI design, FIFOs are often used to store and retrieve data in a specific order, where the first data item to be stored is the first to be retrieved. FIFOs are commonly employed in digital systems for buffering data between different modules or processes.

II. Methodology

The algorithm reduces the problem of finding the GCD of two numbers a and b ($a \geq b$) by transforming it into a problem of finding the GCD of b and ab. This process is repeated until the remainder is zero. The last non-zero remainder is the GCD.

Consider the numbers 15 and 27. We apply the Euclidean Algorithm to find their GCD.

A. Iteration Details

To compute the greatest common divisor (GCD) of two numbers, I employed an algorithm that continuously evaluates whether one of the numbers, denoted as B, equals zero. If B equals zero, then the other number, denoted as A, holds the GCD. This condition is continuously monitored by a comparator.

Throughout the process, another comparator assesses whether A is less than B. If this condition holds true, indicating that A is indeed less than B, the algorithm subtracts B from A. Conversely, if A is not less than B, the algorithm swaps the values of A and B.

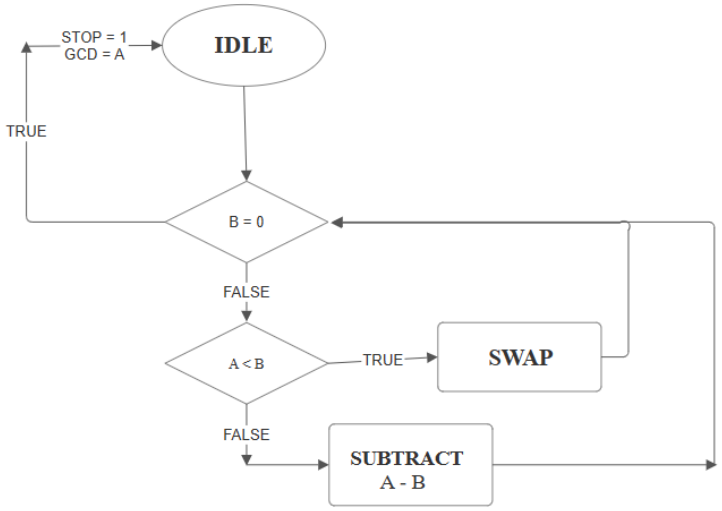

Fig. 1. Pseudo code to calculate GCD of 2 numbers

To illustrate the progression of the GCD calculation, a table is provided with the values of the internal registers, along with the corresponding cycle number, showcasing the step-by-step execution of the algorithm for the given example of computing the GCD of 27 and 15.

TABLE I
GCD CALCULATION INTERMEDIATE RESULT

no of clock cycle	Reg_X	Reg_Y
0	0	0
5	27	15
6	12	15
7	15	12
8	12	3
9	9	3
10	6	3
11	3	3
12	0	3
13	3	0

III. Design Overview:

The GCD calculator module is designed to efficiently compute the Greatest Common Divisor (GCD) of two 16-bit input numbers. Initially, the GCD algorithm was developed and validated in a high-level language such as Python to ensure correctness and efficiency. This approach provided a solid foundation for transitioning to hardware description languages like Verilog.

The processor design incorporates FIFOs for managing input data and storing GCD results, streamlining the data handling process. Inputs A and B are loaded into FIFO A and FIFO B, respectively, ensuring organized data storage and retrieval. Control signals are implemented to initiate GCD calculations and indicate readiness for subsequent computations, optimizing the computational workflow.

By integrating these components, the GCD processor is equipped to handle complex GCD computations with precision and efficiency, making it suitable for low power VLSI design applications

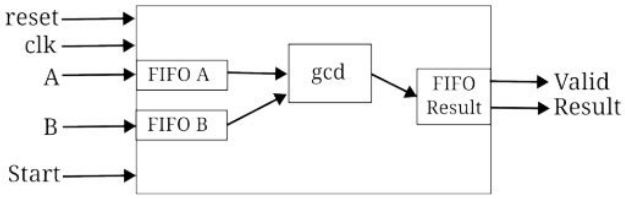

Fig. 2. GCD Processor's block Diagram

IV. Implementation and Testing:

The implementation of the GCD calculator and processor modules involves synthesizing the code to ensure proper functionality and adherence to design specifications. A test bench is created to verify the correct operation of the modules, including scenarios with sample input values to validate the GCD calculation process. Simulation results demonstrate the successful computation of the GCD and the efficient data flow within the processor design.

V. Simulation Result:

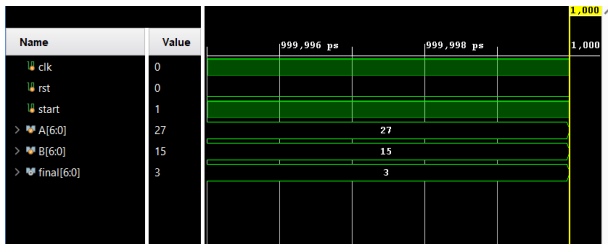

Fig. 3. Simulator results of GCD Processor.

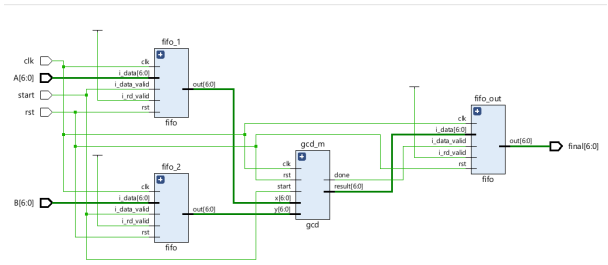

Fig. 4. RTL Schematic of GCD Processor

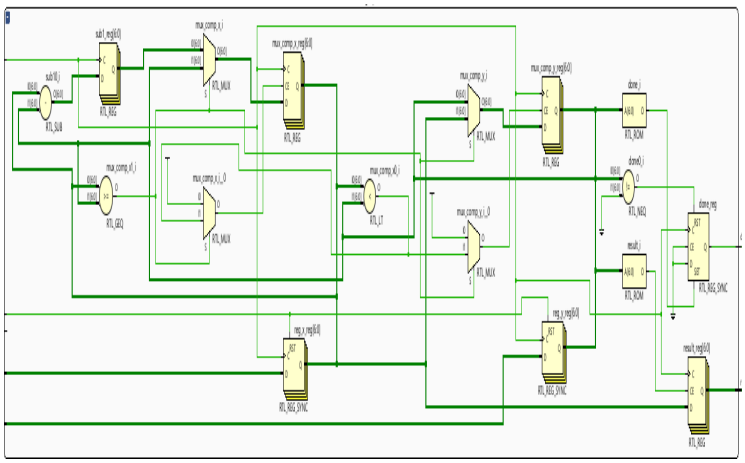

Fig. 5. RTL Schematic of GCD Calculator block.

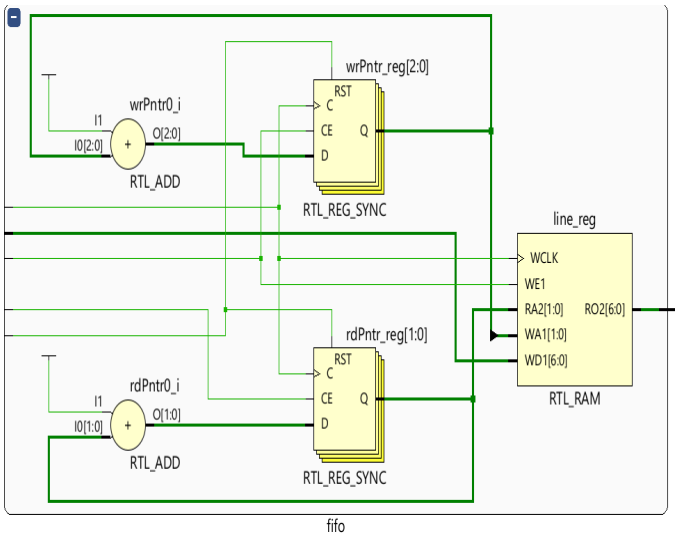

Fig. 6. RTL Schematic of FIFO block.

VI. Synthesis Reports

Synthesizing the Verilog code for the GCD calculator and processor modules on the Nexys 4 DDR FPGA board enables the estimation of key metrics such as area, delay, and power consumption. These metrics provide insights into the efficiency and performance characteristics of the design, guiding optimizations for enhanced functionality on the specific FPGA platform.

A. Resource Utilization

TABLE II
Resource Utilization

Site Type	Used	Fixed	Available	Utilization (%)
Slice LUTs	46	0	63400	0.07
LUT as Logic	28	0	63400	0.04
LUT as Memory	18	0	19000	0.09
Slice Registers	55	0	126800	0.04
Register as Flip-Flop	55	0	126800	0.09

B. Power Utilization

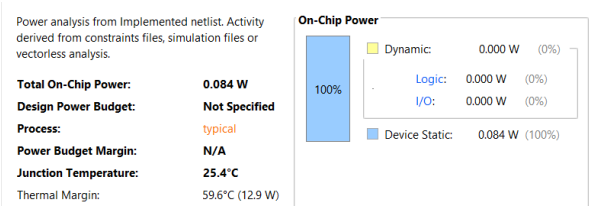

Fig. 7. RTL Schematic of GCD Calculator block.

TABLE III
On-Chip Power Utilization

Component	**Power (W)**	**Used**	**Available**	**Utilization (%)**
Slice Logic	0.000	1	—	—
Others	0.000	1	—	—
I/O	0.000	15	200	7.50
Static Power	0.103	—	—	—
Total	**0.103**	—	—	—

C. Delay

The total data path delay is:
Data Path Delay: 6.83ns (logic 4.86ns route 1.97ns)

VII. Conclusion

In conclusion, the design and implementation of a GCD calculator and processor in the context of Low Power VLSI Design showcase the importance of efficient data processing and handling in digital systems. By synthesizing the code, testing functionality, and analyzing key metrics, this lab experiment offers valuable insights into optimizing performance and minimizing power consumption in VLSI applications. The Euclidean Algorithm provides a systematic approach to compute the GCD of two numbers. The example of 15 and 27 illustrates the simplicity and effectiveness of this method, achieving the result in a concise number of steps.