🧮 GCD Calculator Using Euclidean Algorithm in Verilog

📌 1. Introduction

The **Greatest Common Divisor (GCD)** of two numbers is the largest integer that divides both numbers exactly. It is widely used in:

🔹 **Number theory** 🔹 **Cryptography** 🔹 **Digital hardware design**

This document explores the theoretical aspects of designing a **GCD Calculator** using the **Euclidean Algorithm** in **Verilog HDL**.

📖 2. Understanding GCD

Given two positive integers **A** and **B**, the **GCD(A, B)** is the largest integer **D** such that: ✅ **D divides A** ✅ **D divides B**

📌 Examples:

- 🟢 GCD(12, 8) = 4
- 🟢 GCD(9, 6) = 3
- 🟢 GCD(15, 5) = 5

The most efficient way to compute GCD is through the **Euclidean Algorithm**.

🔄 3. Euclidean Algorithm for GCD

The **Euclidean Algorithm** works by repeatedly subtracting the smaller number from the larger number until both numbers become equal.

⚙️ Algorithm Steps:

1️⃣ Start with two numbers **A** and **B**. 2️⃣ If **A = B**, then **GCD = A (or B)**. 3️⃣ If **A > B**, replace **A** with **A - B**. 4️⃣ If **B > A**, replace **B** with **B - A**. 5️⃣ Repeat **steps 2-4** until **A = B**. 6️⃣ The final value of **A (or B)** is the **GCD**.

📊 Example Calculation:

- **Find GCD(12, 8)**

Step	🅰️ A	🅱️ B	🔄 Action
1	12	8	A > B → A = 12 - 8 = 4
2	4	8	B > A → B = 8 - 4 = 4
3	4	4	A = B → ✅ GCD = 4

- **Find GCD(9, 6)**

Step	🅰️ A	🅱️ B	🔄 Action
1	9	6	A > B → A = 9 - 6 = 3
2	3	6	B > A → B = 6 - 3 = 3
3	3	3	A = B → ✅ GCD = 3

🏗️ 4. Hardware Implementation Strategy

To implement the **GCD Calculator** in Verilog, we use a **Finite State Machine (FSM)** with three states:

- **State 1: IDLE** 🕒
  - Waits for the `start` signal.
  - Loads the inputs **A** and **B** into internal registers.
- **State 2: COMPUTE** 🔄
  - If **A > B**, subtract `B` from `A`.
  - If **B > A**, subtract `A` from `B`.
  - Repeat the process until **A = B**.
- **State 3: DONE** ✅
  - The computed GCD is stored in the output register.
  - The `done` signal is set high to indicate completion.

⚡ 5. Hardware Design Considerations

🔹 **Bit-Width Selection**: Supports **4-bit numbers**, extendable to **8-bit or 16-bit**. 🔹 **Clock & Reset Handling**: The FSM operates on a **clock signal**, and a **reset signal** reinitializes the system. 🔹 **Optimization**: A **Modulo-based Euclidean Algorithm** can be used for better efficiency. 🔹 **Synthesis**: The design can be implemented on **FPGAs** for real-time applications.

🎯 6. Applications of GCD Calculation in Digital Systems

✅ **Cryptography**: RSA encryption uses GCD for key generation. ✅ **Signal Processing**: Used in **sampling rate conversion**. ✅ **Computer Networks**: Used in **checksum calculations** and **error detection**. ✅ **Mathematical Computation**: Helps in **simplifying fractions** and **number theory problems**.

📌 7. Conclusion

The **GCD Calculator** is a fundamental module in digital hardware. The **Euclidean Algorithm** provides an efficient way to compute GCD. By using a **state-machine approach in Verilog**, we can design a reliable GCD calculator for FPGA or ASIC implementations. Future enhancements can include **modulo-based computation** and **pipeline-based architectures** for higher performance.

🚀 8. Future Improvements

🔹 Implementing **Modulo-based Euclidean Algorithm**. 🔹 Expanding bit-width support for **16-bit and 32-bit GCD calculations**. 🔹 Using **pipelining** to compute GCD for multiple number pairs in parallel. 🔹 Implementing **GCD in hardware accelerators** for cryptographic applications.