ECE 310
Project 3 Report

Brody Enget

April 14, 2025

Abstract

This lab implements a sequential 8×8 unsigned multiplier using a behavioral implementation in Verilog.

1 Introduction

This lab implements a sequential 8×8 unsigned multiplier using behavioral modeling in Verilog. The design takes two 8-bit unsigned inputs and produces a 16-bit product by performing shift-and-add multiplication over multiple clock cycles. On reset, the inputs are loaded into internal registers, and the multiplication begins by conditionally adding the shifted multiplier to the product based on the least significant bit of the multiplicand. Both the multiplier and multiplicand are shifted each cycle until the operation is complete. The implementation avoids combinational logic and counters, relying instead on the shifting of the multiplicand to determine completion.

Signal	**Direction**	**Description**
clock	input	Free-running system clock
reset	input	Active-high synchronous reset
multiplier	input	8-bit unsigned multiplier operand
multiplicand	input	8-bit unsigned multiplicand operand
product	output	16-bit unsigned multiplication result

Table 1: Top-Level Ports for `mult_seq_8x8` Module

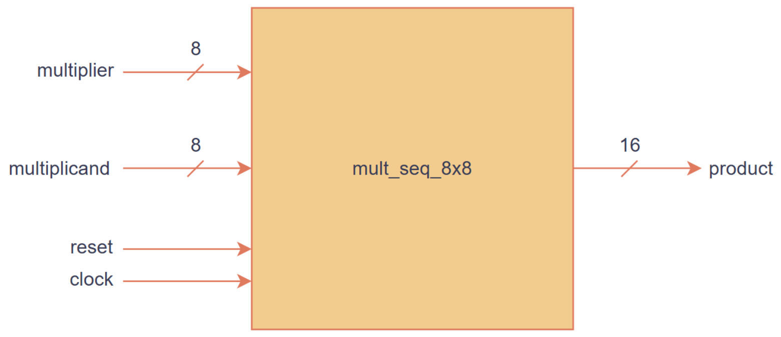

Figure 1: Block Diagram

2 Design & Theory

The design for this lab centers around the use of shift registers to perform sequential addition-based multiplication. The multiplier is loaded into a 16-bit register, while the multiplicand is stored in an 8-bit register. The product is maintained in a separate 16-bit register initialized to zero.

The first step in the process is to check the state of the `reset` signal. When `reset` is high, it indicates that the system is ready to load new inputs. At this point, the multiplier and multiplicand are loaded into their respective registers, and the product register is cleared to all zeros.

When `reset` goes low, the system begins computation. The computation continues as long as the multiplicand register is not equal to zero, meaning there are still bits to process. During each clock cycle, the least significant bit of the multiplicand register (`multiplicand_reg[0]`) is evaluated. If it is 1, the current value of the multiplier register is added to the product register. Regardless of whether the bit is 0 or 1, the multiplicand register undergoes a logical right shift (dropping its LSB), and the multiplier register undergoes a logical left shift (increasing its magnitude by one bit).

This shifting mechanism enables the sequential accumulation of partial products. The process repeats on each rising clock edge until all bits of the multiplicand have been processed, at which point the multiplication is complete and the final product resides in the product register.

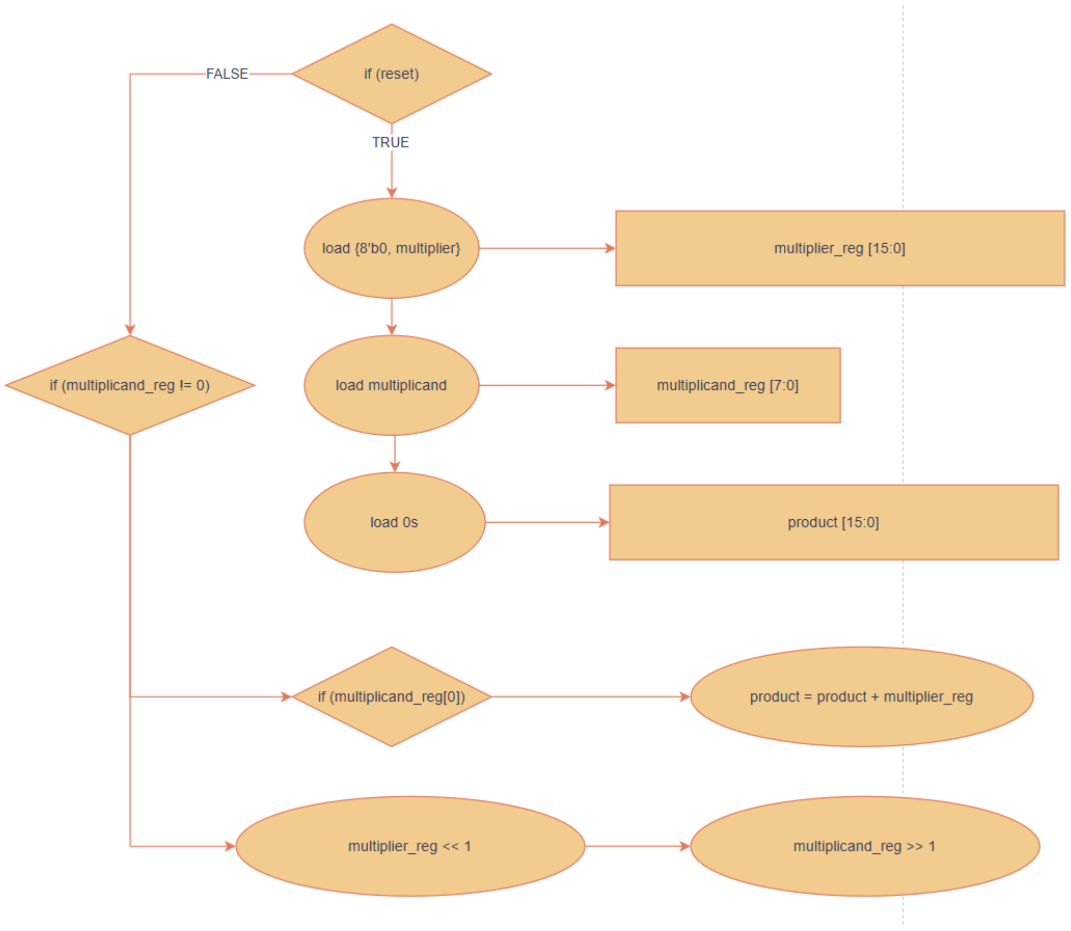

Figure 2: Flow Diagram

3 Testing & Results

This section outlines the testing procedure used to validate the complete functionality of the system.

3.1 Intensive Testing

The system was tested using a single continuous stream of inputs, incorporating four primary test cases. To ensure comprehensive verification, the test stream was designed to evaluate the following critical functionalities:

- **Reset:** Asserting the `reset` signal high should correctly initialize the system to a known state by clearing all D flip-flops and setting all internal registers to zero.
- **Multiplication Computation:** The system must accurately perform unsigned multiplication, producing correct results over multiple clock cycles. To thoroughly evaluate this, a combination of normal and edge cases must be used.

Each test case begins with a high `reset` signal to ensure proper initialization. The selected inputs are as follows:

- **Input 1 - 255 × 255:** This edge case tests the upper bound of the system, using the maximum values for both operands. It validates the system's ability to handle full-width multiplication and verify that the product does not overflow or truncate. Expected output = 65025
- **Input 2 - 179 × 131:** A standard case involving two non-trivial prime numbers. This verifies that the system handles typical computations correctly and is not limited to special-case optimizations. Expected output = 23449
- **Input 3 - 255 × 0:** An edge case where the multiplicand is zero. This tests whether the system correctly short-circuits the computation and immediately produces a zero result. Expected output = 0
- **Input 4 - 0 × 255:** An edge case where the multiplier is zero. This ensures that the system properly handles multiplication when the initial multiplier is zero, again resulting in a zero product without unnecessary computation. Expected output = 0

3.2 Results

The following waveforms depict the outputs corresponding to the previously described input cases, with each test case is discussed in detail below:

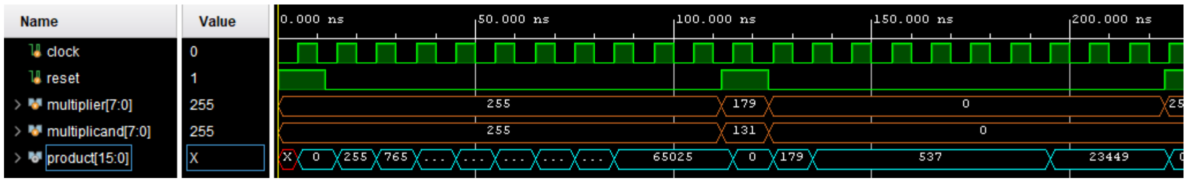

Figure 3: Waveform 1 — Inputs 1 and 2

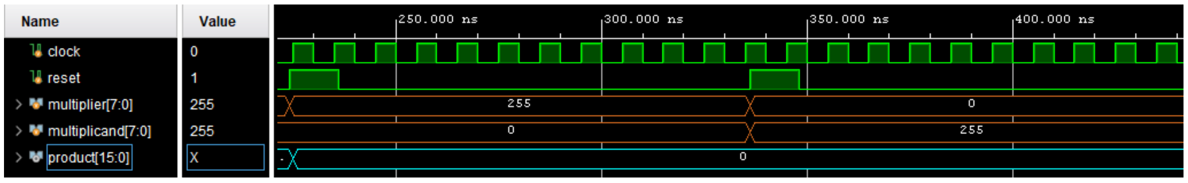

Figure 4: Waveform 2 — Inputs 3 and 4

- **Input 1 — `255 × 255`:** The system correctly produces the expected result after 8 clock cycles, as required by the shift-and-add multiplication algorithm. The `reset` signal successfully initializes the registers, loading the operands into their respective registers and clearing the product register to zero.
- **Input 2 — `179 × 131`:** This input was applied simultaneously with the reset signal to test the system's ability to handle simultaneous initialization and data loading. The system correctly loads the inputs and computes the result across 8 clock cycles. As expected, computation is only performed on cycles where the least significant bit of the multiplicand is 1, validating the selective addition logic.
- **Input 3 — `255 × 0`:** The system appropriately handles a multiplicand of zero. As there are no bits set in the multiplicand, no additions are performed, and the product register remains at zero throughout the operation.
- **Input 4 — `0 × 255`:** Similarly, the system correctly handles a multiplier of zero. The product register remains zero throughout, and no computation is performed, validating the edge case handling for zero operands.

4 Synthesis Analysis

This section discusses the Vivado synthesis results and how they compare to the intended design. It is important to note that the implementation primarily uses behavioral modeling, which gives the synthesis tool considerable flexibility in how it generates the corresponding hardware.

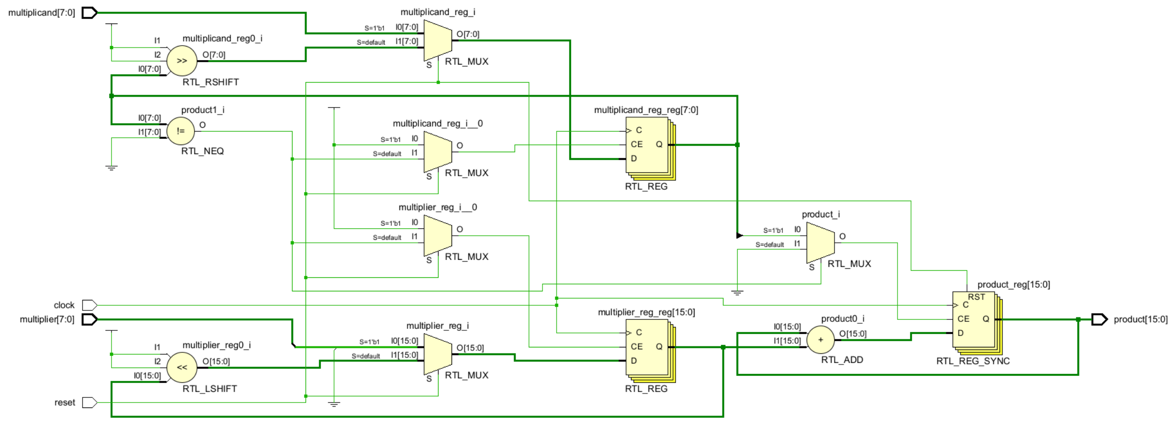

Figure 5: Synthesized schematic of the sequential multiplier

Figure 4 shows the synthesized schematic corresponding to the behavioral design described earlier. As expected, the tool translates the behavioral description into an equivalent hardware representation. Key components such as shifters, adders, and comparison logic (`!=`) are clearly visible, demonstrating that the design uses shift-and-add multiplication as intended. The resulting hardware faithfully preserves the functionality of the original Verilog implementation while optimizing structure and layout for synthesis.

5 Conclusion

The sequential 8×8 unsigned multiplier was successfully implemented using behavioral Verilog and validated through both simulation and synthesis. By leveraging shift registers and conditional addition based on the multiplicand's least significant bit, the design achieved an efficient and resource-conscious approach to multiplication over multiple clock cycles. The use of behavioral modeling allowed for a clean, high-level implementation while still enabling the synthesis tool to generate an optimized and functional hardware representation. The synthesized schematic confirmed that the expected computational elements—such as shifters and adders—were inferred correctly and matched the intended design behavior.

The system was thoroughly tested using an intensive input stream that covered both normal and edge case scenarios. These included full-range multiplication, multiplication with prime numbers, and cases involving zero operands. In each case, the system produced the correct result, confirming that the internal register logic, reset behavior, and shift-based computation were functioning as expected. Overall, the project demonstrated a reliable and accurate implementation of a sequential multiplier, showcasing the effectiveness of behavioral modeling in designing synchronous arithmetic circuits.

The report was compiled using LaTeX .