Digital design with FPGAs

4x16 Bit High-Speed Multiplier

Abdullah Emir Atmaca 230208060

1. Problem Definition and Architecture Overview

1.1 Problem Summary

This project focuses on designing a **16×4-bit high-speed multiplier**.
Unlike the classic sequential multiplier, which multiplies **one bit per clock cycle**, the proposed design processes **four bits per cycle**.
As a result, the total number of states required to complete the multiplication operation is significantly reduced, leading to a **faster execution time** and **improved performance**.

1.2 System Architecture

The system consists of two main components: the **Control Unit** and the **Datapath**.
The Control Unit generates control signals that coordinate the Datapath operations, determining when and how data should be processed.
The Datapath contains a **4×4 Array Multiplier**, an **8-bit Adder**, and an **Accumulator Register**.
These components work together to perform the required **arithmetic and logic operations** under the supervision of the Control Unit.

2. State Graph Design

2.1 State Graph Diagram

Fig2.1:

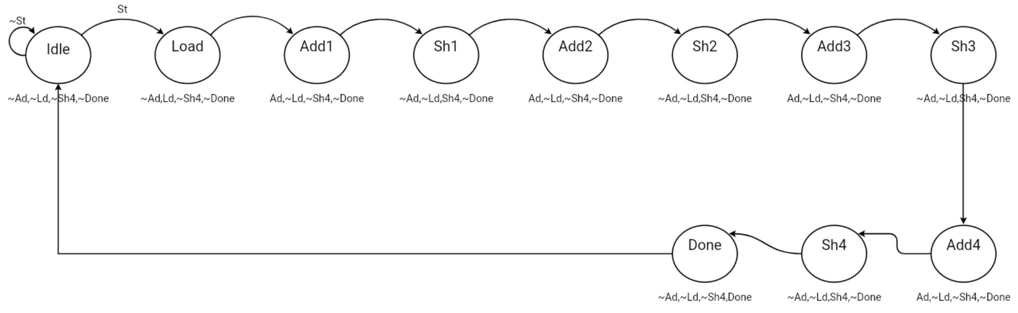


Figure 2.1 shows the complete FSM diagram, including all conditions for state transitions.
The diagram also indicates which control signals are active for the Datapath in each state.
Note that Ad represents an active-high signal (1), while ~Ad indicates that the signal is inactive (0).

2.2 Description of Each State

The FSM starts in the **IDLE** state. In this state, no operations are performed, and the system waits for the Start (St) signal.
When St is asserted, the FSM transitions to the **LOAD** state. In the LOAD state, the A register is loaded: the 8 most significant bits are cleared, and the 16-bit multiplier value is loaded into the least significant bits.
After LOAD, the FSM moves to **AD1**, where the first 8-bit partial product (Mcand × A[3:0]) is added to A[23:16].
The next state, **SH1**, performs a 4-bit right shift on the A register

to prepare for the next partial multiplication.
This sequence continues with **AD2**, **AD3**, **AD4** and **SH2**, **SH3**, **SH4**, where the same arithmetic and logic operations are performed on the remaining groups of bits.
After the final shift (**SH4**), the FSM transitions to the **DONE** state. In DONE, the Done pulse is asserted to indicate completion of the multiplication operation.
Finally, the FSM returns to the **IDLE** state, ready to start the next operation cycle.

3. FSM and Datapath Design Details

3.1 FSM Functionality

The FSM controls the operation sequence based on the current state (Moore FSM).
Each state generates specific control signals for the Datapath.
LOAD initializes the A register, AD1–AD4 add partial products, and SH1–SH4 perform right shifts.
The Done signal is asserted only in the DONE state, indicating completion of the multiplication.

3.2 Datapath Components

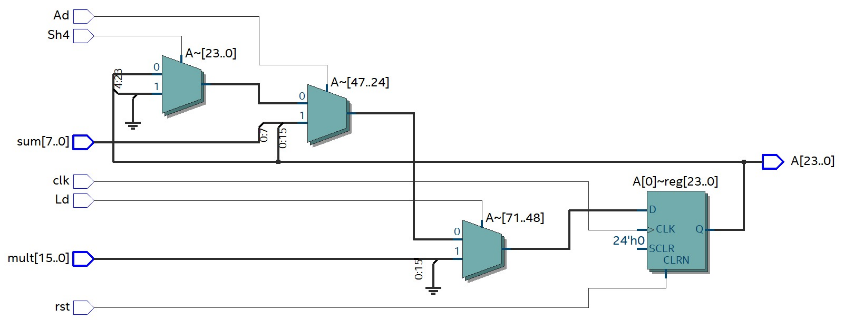


Fig3.2

The Datapath consists of registers and multiplexers that perform the arithmetic and logic operations required for the 16×4-bit multiplication. As shown in Figure 3.2, the 24-bit A register stores the partial product, while multiple 8-bit segments are processed through the multiplexer network to select appropriate inputs for addition and shifting. The sum[7:0] input represents the partial product generated by the 4×4 multiplier, and the control signals Ad and Sh determine whether addition or shifting operations are performed. The Datapath is clocked by clk, and the Ld signal is used to load new values into the registers. This modular structure ensures that arithmetic operations and shifting are performed efficiently under the supervision of the FSM.

3.3 Control Signal Effects on A Register

The 24-bit A register stores the partial product and is controlled by the following internal signals from the Control Unit:

- **Ld:** Loads the register with {8'b0, Mult[15:0]}, initializing the partial product.
- **Ad:** Adds the 8-bit product (Mcand * A[3:0]) to the upper byte A[23:16].
- **Sh4:** Performs a **right shift by 4 bits** on the entire A register (A <= A >> 4), aligning the partial product for the next addition.

These signals allow the A register to correctly accumulate partial products under FSM supervision, ensuring the final product is accurate.

4. Simulation and Verification

4.1 Simulation Setup

The multiplier was tested using a Verilog testbench. The clock (clk) toggles every 5 ns, and the system is initialized with rst asserted for 20 ns. The start signal st triggers the multiplication operation. Inputs mcand (4-bit) and mult (16-bit) are applied sequentially for each test case, and outputs product and done are monitored to verify correct operation.

4.2 Test Cases and Waveforms

Three test cases were applied to check the functionality of the FSM-controlled multiplier:

Test 1: mcand = 3, mult = 1000 → Expected product: 3000
Test 2: mcand = 15, mult = 0 → Expected product: 0
Test 3: mcand = 15, mult = 65535 → Expected product: 983025

For each test, st is asserted to start the FSM sequence: IDLE → LOAD → ADD → SH → … → DONE. The A register is loaded (Ld) with {0, mult}, partial products are accumulated (Ad), and shifted right by 4 bits (Sh4) at each stage. The done signal asserts only when the FSM reaches the DONE state.

4.3 Analysis of Results

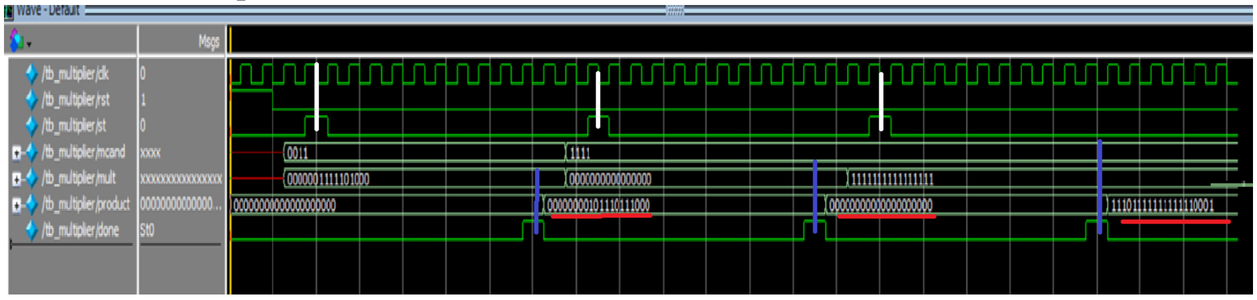
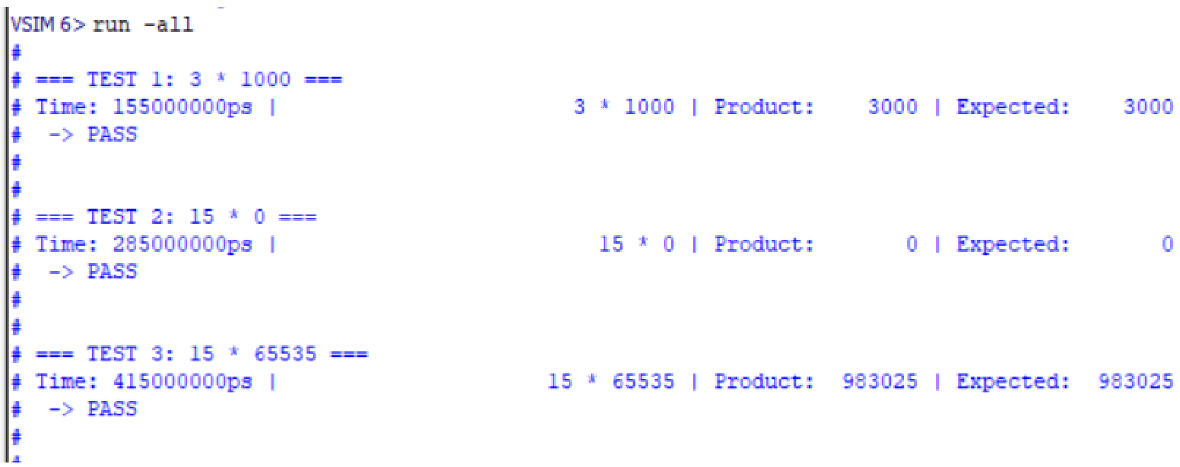


Fig4.3

As shown in **Figure 4.3**, the **white lines** represent the St signal, the **blue stripes** indicate the Done signal, and the **red line** shows the multiplication result (product).
From the simulation, it can be seen that the correct product value appears approximately **one clock cycle after** the Done pulse. This one-cycle delay occurs because of the sequential nature of the always block.
Initially, the design used the statement assign product = A[19:0];, which continuously updated the output during operation. To ensure that the product value is only written when the multiplication is complete, the code was modified so that the value updates only after the Done signal is asserted. This change introduces a one-clock delay but ensures that only the final, correct product is displayed.



```
VSIM 6> run -all
#
# === TEST 1: 3 * 1000 ===
# Time: 155000000ps |                 3 * 1000 | Product:    3000 | Expected:    3000
#   -> PASS
#
#
# === TEST 2: 15 * 0 ===
# Time: 285000000ps |                  15 * 0 | Product:       0 | Expected:       0
#   -> PASS
#
#
# === TEST 3: 15 * 65535 ===
# Time: 415000000ps |               15 * 65535 | Product:  983025 | Expected:  983025
#   -> PASS
#
```

Fig4.3.2

As shown in **Figure 4.3.2**, all test cases defined in the testbench passed successfully, confirming that the FSM and Datapath function as expected.

5. Conclusion

The 4×16-bit high-speed multiplier design operates correctly, as verified by the simulation results. All test cases passed successfully, demonstrating that the FSM and datapath were implemented properly.
The only minor drawback observed is that the correct product value appears approximately **one clock cycle after** the Done pulse. This delay occurs due to the sequential behavior of the always block, which updates the output one cycle later.
Despite this small timing offset, the design performs accurately and efficiently for the intended purpose.