Design Vector Multiplier using System Verilog

1. Introduction

The objective of this project is to design, simulate, and analyze a two-input vector multiplier using **SystemVerilog** on Modelsim and Quartus for Synthsis. The multiplier was implemented in two styles: one is purely combinational Array Multiplier, and the other one has registered inputs and outputs therefore it is Clock-Driven or Sequential (Registerd) Multiplier. Figure 1 and 2 shows the structure of an array multiplier.

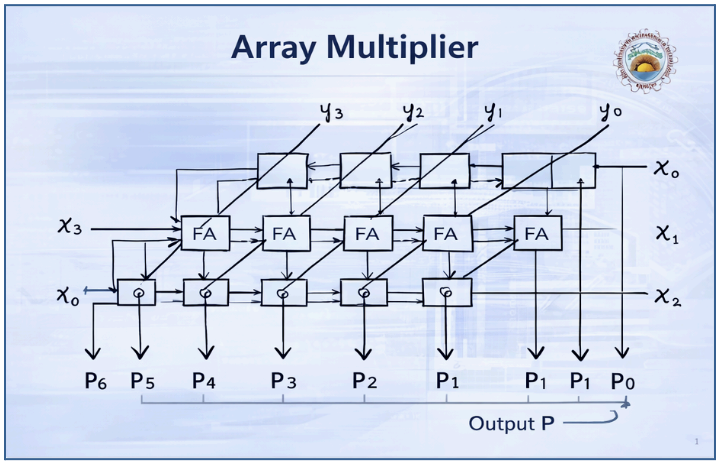


Figure 1: Combinational Vector Multiplier

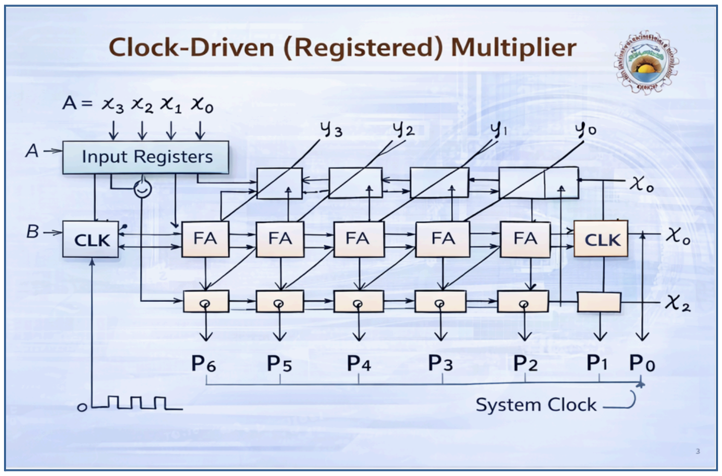

Clock-Driven (Registered) Multiplier
A = x_3 x_2 x_1 x_0
A
Input Registers
y_3 y_2 y_1 y_0
x_0
B
CLK
FA FA FA FA FA
CLK
x_0
x_2
P_6 P_5 P_4 P_3 P_2 P_1 P_0
System Clock

Figure 2: Clock-Driven Vector Multiplier

The array multiplier shown in figure 1 has blocks named FA, named upon Full-Adder. But they are actually o exact full adders though they are modified to receive one input directly, and other input is fed from an AND gate. This is done to implement multiplication.

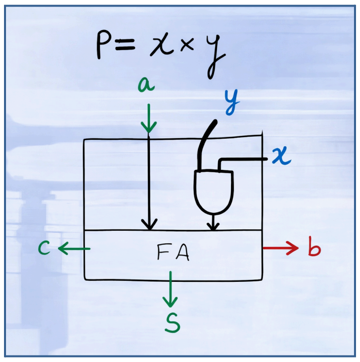


Figure 3: Modified Full-Adder Block for Multiplication

These designs were developed and verified using ModelSim for functional simulation and Intel Quartus Prime for synthesis and post-fitting analysis. The performance and resource utilization of each design were also compared.

2. Design Overview

2.1 Array (Combinational) Multiplier

The first implementation is a purely combinational *array multiplier*. It accepts two operand vectors (A and B) and produces the product (P) without clock dependency.

2.1.2 RTL Diagram

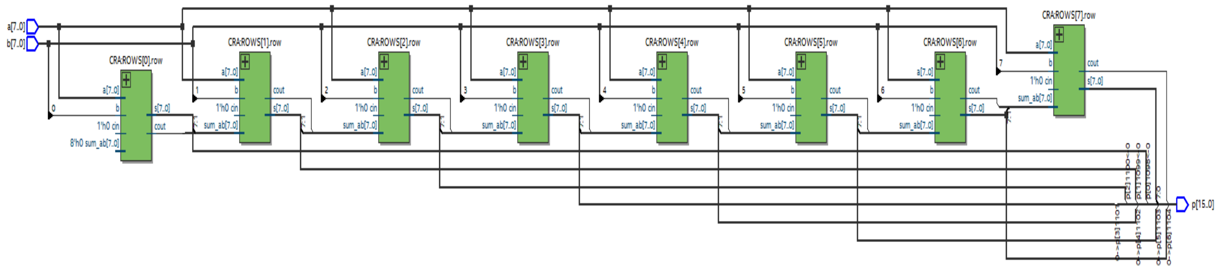


Figure 4: Vector Multiplier Array RTL Schematic

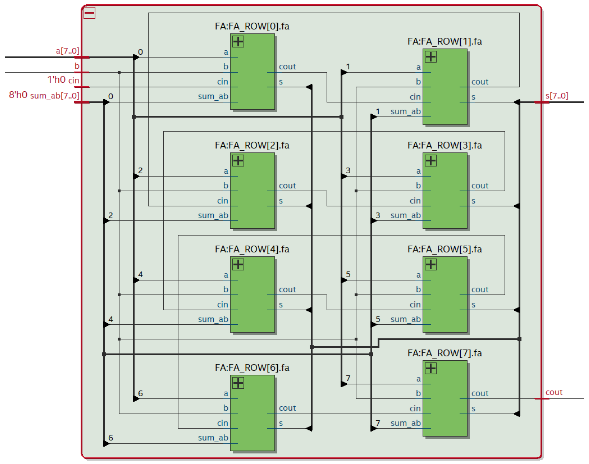


Figure 5: Elaborated Single Multiplier

The RTL schematic illustrates direct data flow from the input vectors to the product output using combinational logic only.

2.1.3 Post-Fitting Diagram

A post-fitting diagram or Post-Fit Netlist in Intel Quartus Prime software is a visual representation of the FPGA design after the Fitter has mapped logic to specific, physical device resources (Logic Elements, pins, RAM blocks) and performed routing. It shows the final, routed layout, allowing for debugging and timing verification.

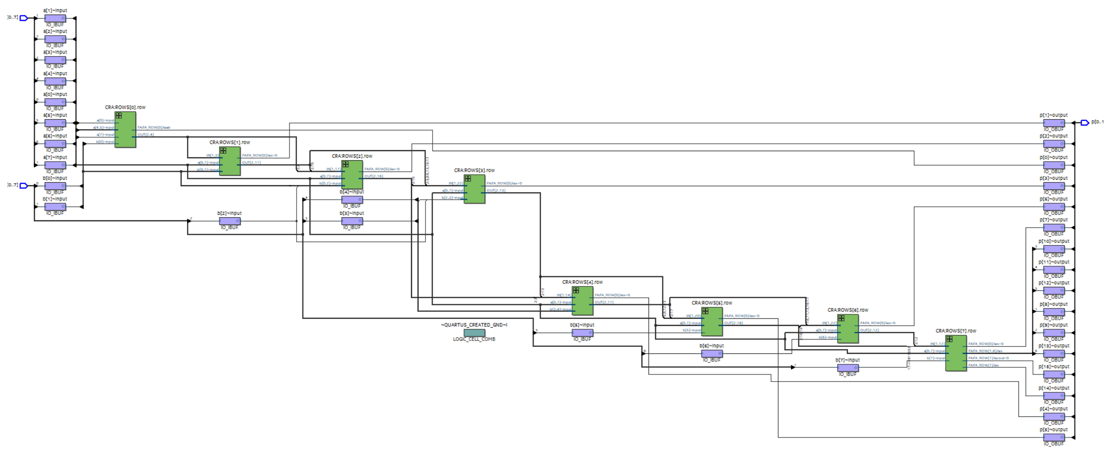


Figure 6: Post-Fitting Diagram

2.1 Clock-Driven Multiplier

In the second implementation, all inputs and outputs are **registered**. This means Input operands A and B are latched on the rising edge of the clock. The output P is also stored in output registers.This design improves timing performance and data stability, particularly in hardware implementations.

2.2.1 RTL Diagram

The RTL view shows input and output registers clocked by a single system clock signal.

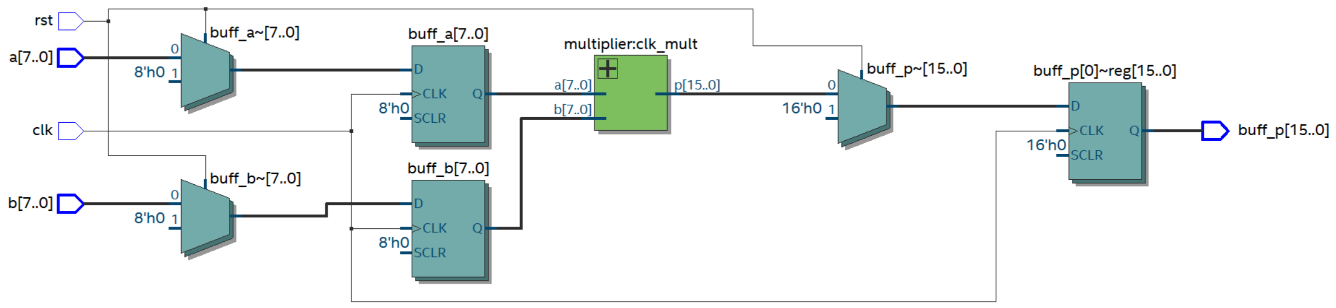


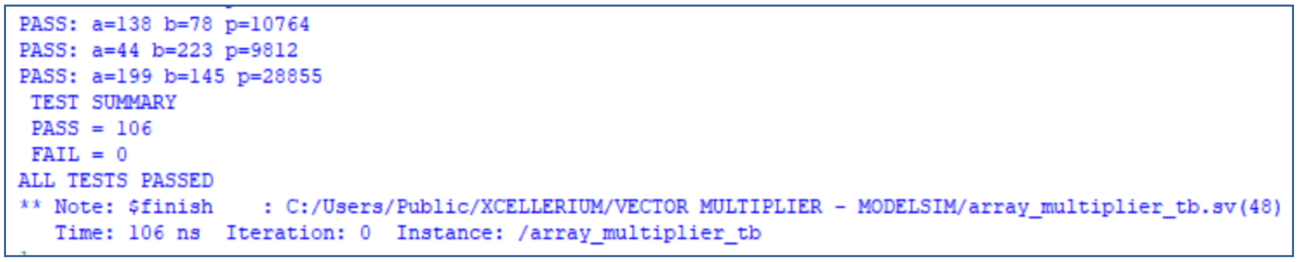
Figure 7: Clock-Driven Multiplier RTL

3.Testbench and Simulation

In order to functionally verify the design, we created self-checking testbenches to test maximum inputs and outputs.

3.1 Self-Checking Testbench for Array Multiplier

A self-checking testbench was written in SystemVerilog to automate verification wher each simulation automatically compares the DUT's output with the expected results. The expected output appears delayed in the waveform. This is because a deliberate delay was added inside the self-checking task to synchronize signal evaluation and ensure correct timing. Without this synchronization, test cases would fail due to immediate signal sampling.



```
PASS: a=138 b=78 p=10764
PASS: a=44 b=223 p=9812
PASS: a=199 b=145 p=28855
 TEST SUMMARY
 PASS = 106
 FAIL = 0
ALL TESTS PASSED
** Note: $finish    : C:/Users/Public/XCELLERIUM/VECTOR MULTIPLIER - MODELSIM/array_multiplier_tb.sv(48)
   Time: 106 ns  Iteration: 0  Instance: /array_multiplier_tb
```

Figure 8: Self-Checking Array Multiplier Testbench Results

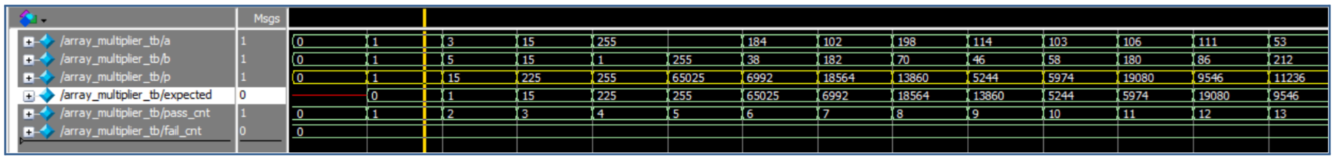
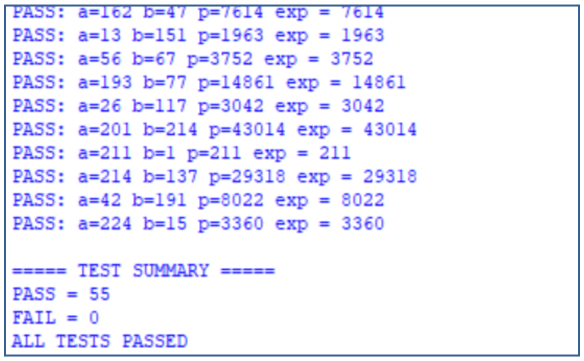

Msgs
/array_multiplier_tb/a 1 | 0 1 3 15 255 184 102 198 114 103 106 111 53
/array_multiplier_tb/b 1 | 0 1 5 15 1 255 38 182 70 46 58 180 86 212
/array_multiplier_tb/p 1 | 0 1 15 225 255 65025 6992 18564 13860 5244 5974 19080 9546 11236
/array_multiplier_tb/expected 0 | 0 1 15 225 255 65025 6992 18564 13860 5244 5974 19080 9546
/array_multiplier_tb/pass_cnt 1 | 0 1 2 3 4 5 6 7 8 9 10 11 12 13
/array_multiplier_tb/fail_cnt 0 | 0

Figure 9: Modelsim Waveform

3.2 Self-Checking Testbench for Clock-Driven Array Multiplier

The sequential version was tested similarly using a self-checking testbench. The expected output is shown as *Hi-Z (red)* in certain waveform segments. However, all test cases passed successfully. This indicates that, the testbench likely drives the expected signal tri-stated ('z') between transactions. The DUT output remains latched correctly, verifying proper register operation. Thus, the Hi-Z state is only a transient visualization artifact — not a functional error.



```
PASS: a=162 b=47 p=7614 exp = 7614
PASS: a=13 b=151 p=1963 exp = 1963
PASS: a=56 b=67 p=3752 exp = 3752
PASS: a=193 b=77 p=14861 exp = 14861
PASS: a=26 b=117 p=3042 exp = 3042
PASS: a=201 b=214 p=43014 exp = 43014
PASS: a=211 b=1 p=211 exp = 211
PASS: a=214 b=137 p=29318 exp = 29318
PASS: a=42 b=191 p=8022 exp = 8022
PASS: a=224 b=15 p=3360 exp = 3360

===== TEST SUMMARY =====
PASS = 55
FAIL = 0
ALL TESTS PASSED
```

Figure 10: Self-Checking Clock-Driven multiplier Testbench Results

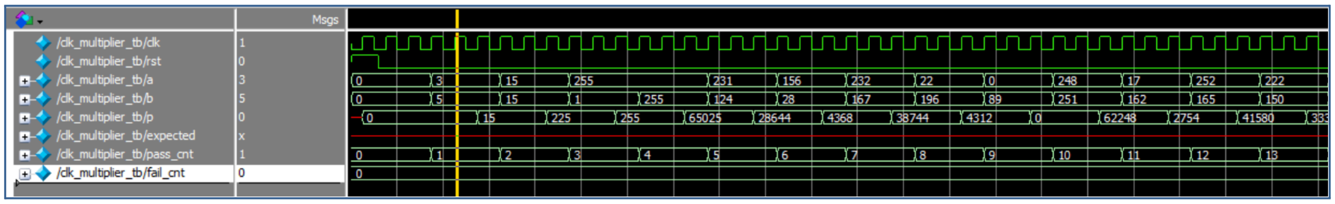


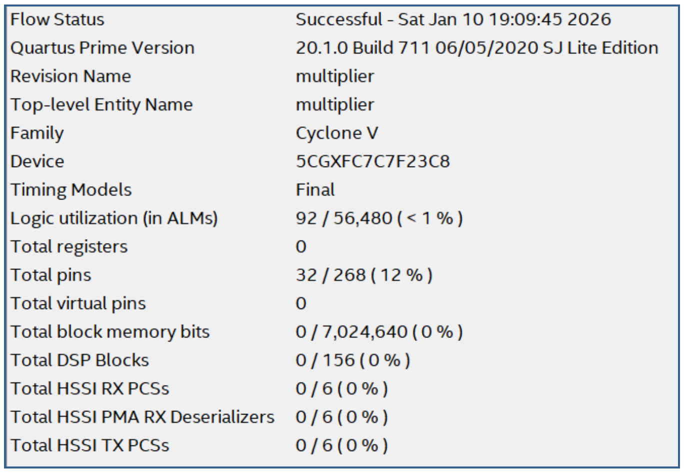
Figure 11: Self-Checking Testbench Results

4. Resource Utilization Summary

The Resource Utilization Summary in Intel Quartus Prime is a compilation report generated after the Fitter stage that details how much of the FPGA's hardware resources (ALMs, registers, memory blocks, DSPs, I/O pins) your design uses. It helps determine if the design fits, identify bottlenecks, and verify if the device has enough resources, accessed via Processing > Compilation Report > Fitter > Resource Section.

4.1 Combinational Array Multiplier

The FPGA resource usage includes all the logic elements for array multiplication and no flip-flops are used which shows that it is purely combinational.

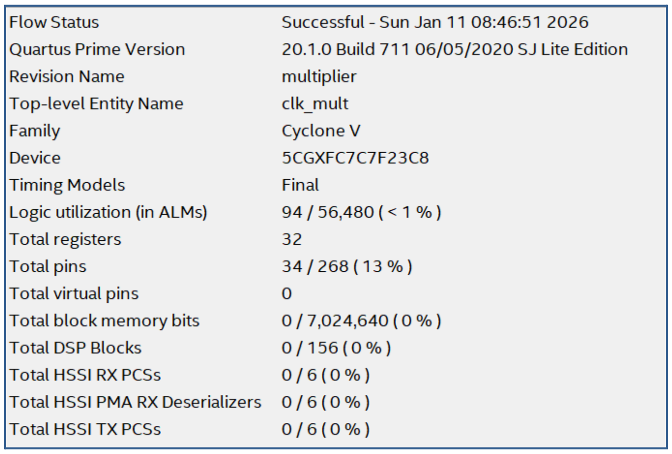


Flow Status	Successful - Sat Jan 10 19:09:45 2026
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	multiplier
Top-level Entity Name	multiplier
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	92 / 56,480 (< 1 %)
Total registers	0
Total pins	32 / 268 (12 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)

Figure 12: Utilization Report

4.2 Clock-Driven Multiplier

Utilization report tells that there are 8 input registers for A, likewise 8 input registers for B, and 16 Output registers for P. Therefore total registers used are 32. This architecture ensures better timing closure and pipelined behavior, suitable for high-speed applications.



Flow Status	Successful - Sun Jan 11 08:46:51 2026
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	multiplier
Top-level Entity Name	clk_mult
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	94 / 56,480 (< 1 %)
Total registers	32
Total pins	34 / 268 (13 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)

Figure 13: Utilization Report

5. Results and Discussion

Design Type	Nature	Testbench Type	Resource Usage	Comments
Array Multiplier	Combinational	Self-Checking	Low	Suitable for simple, fast logic
Clock-Driven Multiplier	Sequential	Self-Checking	Higher (due to registers)	Stable and FPGA-friendly design

The clock-driven design, though consuming more resources, provides more reliable performance for synthesized hardware, especially in timing-critical FPGA environments.

6. Conclusion

Both designs of the two-input vector multiplier were successfully designed, simulated, and verified using SystemVerilog. The **array multiplier** offers a fast, simple structure suited for combinational logic applications. Whereas, the **clock-driven multiplier** enhances timing performance using registered I/O, making it more robust for FPGA deployment. Future improvements could include pipelined or parallelized multiplier structures for higher throughput.