Design and Implementation of 16-bit Serial Multiplier Using Verilog HDL and ASIC Flow

Shaik Umran

B.Tech

(ECE), IIIT Kurnool November 2025

Abstract:

This project presents the design and implementation of a 16×16-bit serial multiplier using the shift-and-add method. The design has been developed at the Register Transfer Level (RTL) using Verilog HDL and verified in Xilinx Vivado through functional simulation. The synthesized netlist was then implemented in the ASIC design environment using Cadence Genus and Innovus tools. The final layout generation produced the GDSII file representing the chip-ready design. The implementation focuses on optimizing hardware resources while maintaining correctness and performance.

1. Introduction:

Multiplication is one of the most fundamental arithmetic operations in digital signal processing and computing systems. A serial multiplier provides an area-efficient approach by processing one bit per clock cycle, reducing the overall hardware complexity compared to parallel architectures. This work implements a 16-bit serial multiplier using Verilog HDL, verified in Xilinx Vivado, and synthesized for ASIC realization in Cadence Genus.

2. Design Methodology:

The proposed design follows the complete digital VLSI design flow:

1. RTL design and simulation using Verilog HDL in Xilinx Vivado
2. Functional verification through waveform and output analysis
3. Synthesis in Cadence Genus for gate-level netlist generation

4. Layout generation in Cadence Innovus
5. GDSII export for fabrication readiness

3. Working Principle of 16-bit Serial Multiplier:

The design performs 16×16-bit serial multiplication using the shift-and-add technique.

Inputs:

- **Serial_multin1** → Multiplicand (A)
- **Serial_multin2** → Multiplier (B)
- **clk, rst** → Clock and Reset signals

Output:

- **Serial_multout** → 32-bit final product

Step-by-Step Operation:

1. A 4-bit counter increments with each clock pulse to track the current bit of the multiplication.
2. The multiplier (Serial_multin2) is right-shifted by the counter value to extract the active bit.
3. If the current bit (LSB) is 1, the multiplicand (Serial_multin1) is added to the partial sum.
4. A partial product is generated and left-shifted by the counter value for proper alignment.
5. Each partial product is added to the accumulated sum (Parproduct sum) every clock cycle.
6. After 16 clock cycles, the complete 32-bit product is available at Serial_multout.

Key Concept: Every clock cycle processes one bit. After 16 cycles, the complete multiplication is performed.

4. Implementation and Simulation Results:

4.1 RTL Implementation Diagram:

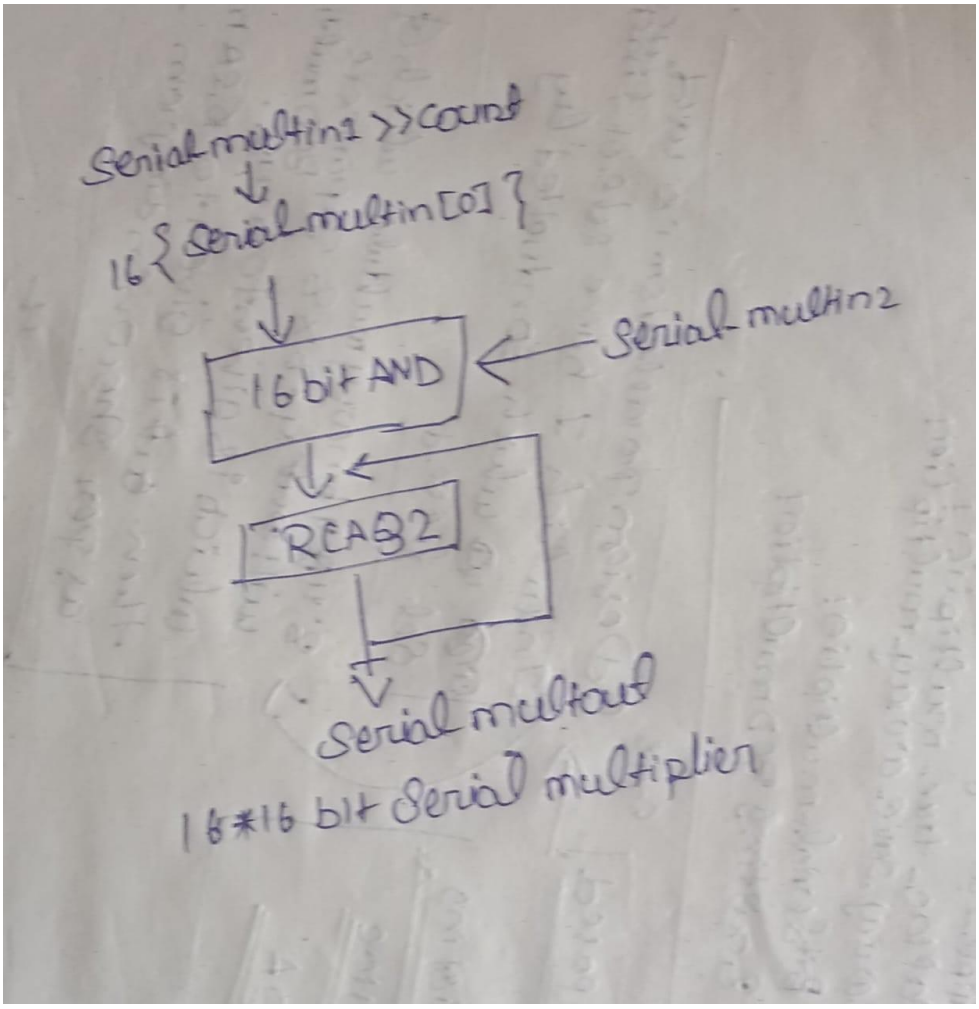

Figure 1: Implementation Diagram of 16-bit Serial Multiplier

4.2 Simulation Waveform:

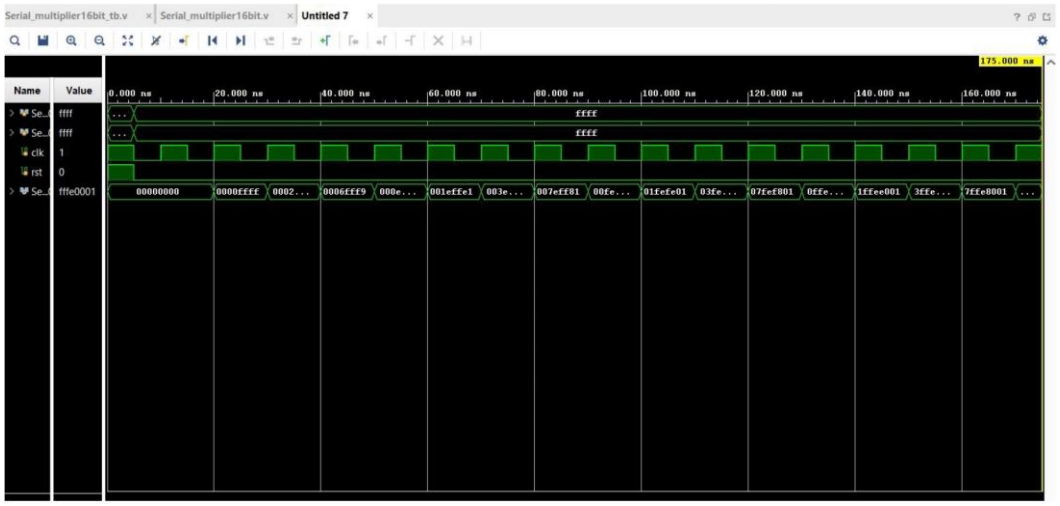

Figure 2: Simulation Waveform of 16-bit Serial Multiplier

4.3 Output Window in Vivado:

Figure 3: Vivado Output Window Showing Final Product

5. Netlist Generation and Analysis:

After verification, synthesis was carried out using Cadence Genus tool. The generated netlist and report are summarized below.

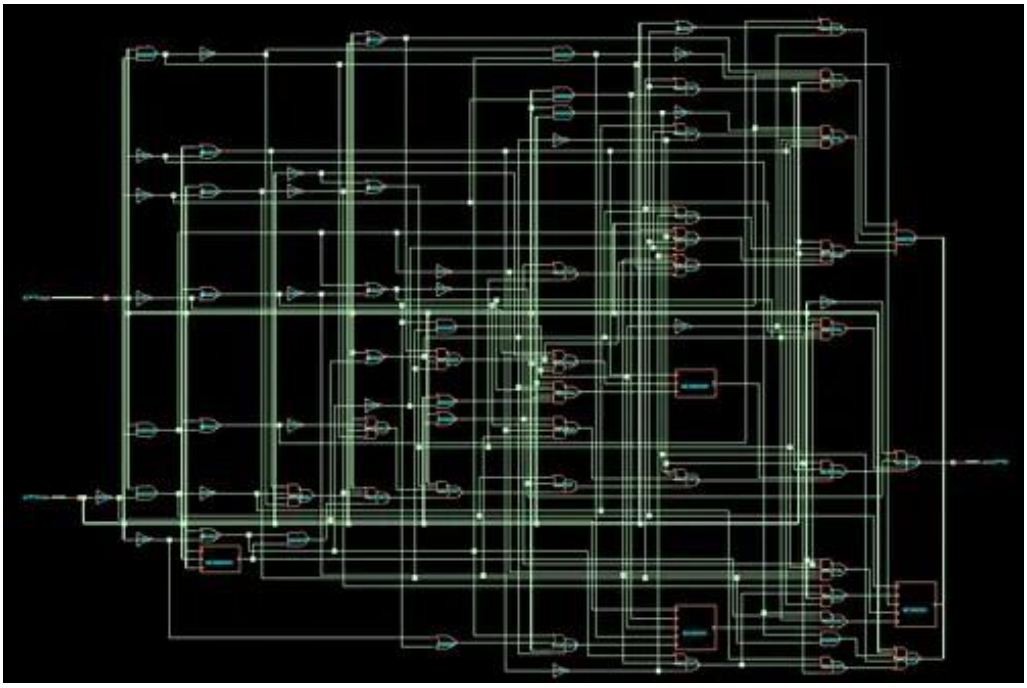

Figure 4: Generated Schematic from Netlist in Cadence Genus

5.1 Synthesis Report Summary:

```
============================================================
   Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
   Generated on:           Oct 19 2025  05:36:18 pm
   Module:                 Serial_multiplier16bit
   Operating conditions:   slow (balanced_tree)
   Wireload mode:          enclosed
   Area mode:              timing library
============================================================
```

Instance	Module	Cell Count	Cell Area	Net Area	Total Area
Serial_multiplier16bit		351	2801.287	0.000	2801.287

5.2 Gate-Level Area Report

Gate	Instances	Area	Library
ADDFHX1	2	42.386	slow
ADDFX1	18	354.229	slow
ADDHX1	1	12.110	slow

AND2X1	2	9.083	slow
AOI21X1	4	18.166	slow
...			
total	351	2801.287	

5.3 Power Report:

Instance: /Serial_multiplier16bit

Power Unit: W

Category	Leakage	Internal	Switching	Total
register	6.54e-06	1.08e-04	4.35e-06	1.19e-04
logic	8.01e-06	4.25e-05	1.68e-05	6.73e-05
clock	0.00e+00	0.00e+00	9.23e-06	9.23e-06
Subtotal	1.46e-05	1.51e-04	3.04e-05	1.96e-04

5.4 Timing Report:

Path 1: MET (4 ps) Setup Check with Pin Parproduct_sum_reg[31]/CK->D

Group: clk

Startpoint: Serial_multin2[6]

Endpoint: Parproduct_sum_reg[31]/D

Slack: 4 ps (Setup met)

6. Layout Implementation:

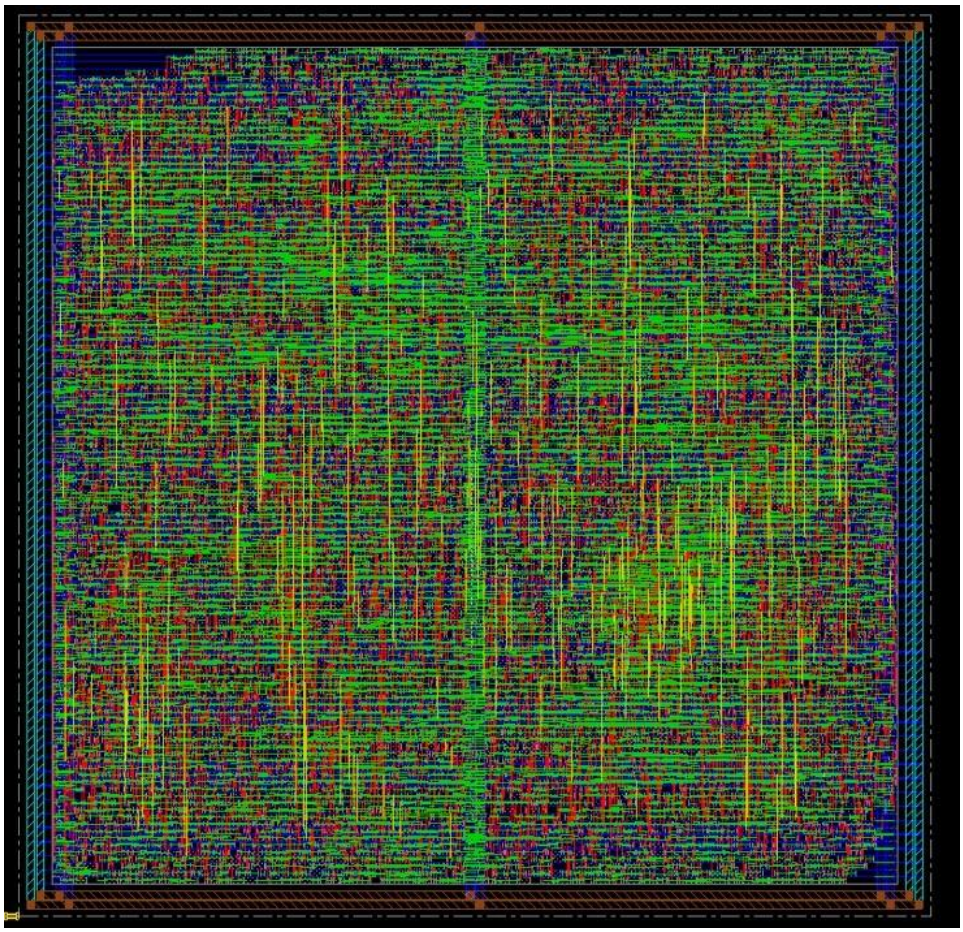

Figure 5: Final Layout Generated in Cadence Innovus

7. Conclusion:

The 16-bit serial multiplier was successfully designed and verified using Verilog HDL. The design was synthesized and implemented in the ASIC flow, generating a clean layout with no DRC or LVS violations. The results confirm the area-efficient operation of the serial multiplier with correct functionality and acceptable timing performance. This work demonstrates the complete digital design flow from RTL to GDSII.

8. References:

1. Mohamed Asan Basiri M and Noor Mahammad, “Configurable folded IIR filter design”, IEEE Transactions on Circuits and Systems II, vol. 62, no. 12, pp. 1144–1148, 2015.

2. A. Ahlander and Bertil Svensson, “Floating point calculations in bit-serial SIMD computers”, Fourth Swedish Workshop on Computer Systems Architecture, pp. 1–11, 1992.

3. Sadeghi, Mohsen, Mahya Zahedi, and Maaruf Ali, “The Cascade Carry Array Multiplier – A Novel Structure of Digital Unsigned Multipliers for Low-Power Consumption and Ultra-Fast Applications”, AETiC, vol. 3, no. 3, pp. 19–27, 2019.

4. C. S. Wallace, “A suggestion for a fast multiplier”, IEEE Transactions on Electronic Computers, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.

5. Carl Hamacher, Zvonko Vranesic, Safwat Zaky, and Naraig Manjikian, “Computer Organization and Embedded Systems”, McGraw Hill Publications, 6th Edition, 2002.

6. Mohamed Asan Basiri M and SK Noor Mahammad, “Memory based multiplier design in custom and FPGA implementation”, Advances in Intelligent Informatics, Springer, pp. 253–265, 2015.

7. Shuli Gao, Dhamin Al-Khalili, J. M. Pierre Langlois, and Noureddine Chabini, “Efficient Realization of BCD Multipliers Using FPGAs”, IJRC, vol. 2017, no. 2410408, pp. 1–12, 2017.

8. Parthibaraj Anguraj and Thiruvenkadam Krishnan, “Design and implementation of modified BCD digit multiplier for digit-by-digit decimal multiplier”, Analog Integrated Circuits and Signal Processing, Springer, vol. 107, pp. 683–694, 2021.

9. Guardia and Carlos Eduardo Minchola, “Implementation of a fully pipelined BCD multiplier in FPGA”, VIII Southern Conference on Programmable Logic, pp. 1–6, 2012.

10. Z. T. Sworna, M. U. I. Haque, and D. M. Anisuzzaman, “High-Speed and Area Efficient LUT-based BCD Multiplier Design”, IEEE WIECON-ECE, pp. 1–4, 2018.

11. https://www.realdigital.org/doc/6dae6583570fd816d1d675b93578203d

12. Mohamed Asan Basiri M, Samaresh Chandra Nayak, and Noor Mahammad, “Multiplication acceleration through quarter precision Wallace tree multiplier”, IEEE SPIN Conference, pp. 502–505, Feb. 2014.

13. Sjalander M and Larsson-Edefors P, “High-Speed and Low-Power Multipliers Using the Baugh-Wooley Algorithm and HPM Reduction Tree”, IEEE ICECS, pp. 33–36, 2008.

14. Mohamed Asan Basiri M, “High Throughput Instruction-Data Level Parallelism Based Arithmetic Hardware Accelerator”, Int. J. of Parallel Programming, Springer, vol. 53, no. 6, pp. 1–30, Feb. 2025.