ECE 310
Project 1 Report: 8x8 Dadda Multiplier

Mason Deal

February 17, 2026

Abstract

Efficient binary multiplication is essential in digital systems where performance and delays directly impacts speed. This project showcases the design and gate-level implementation of an 8×8 Dadda multiplier using Verilog. The architecture reduces partial products using half adders and full adders before performing a final carry-propagate addition with a ripple carry adder. This staged reduction minimizes the number of sequential carry operations and is constructed from basic logic gates and verified using a simulation testbench that applied boundary, worst-case, and random input combinations. All test cases produced correct products, confirming accuracy and proper carry propagation through the system. The results demonstrate that the Dadda reduction technique provides a fast solution for 8x8 binary multiplication while maintaining relatively straightforward hardware implementation.

1 Introduction, Background, and Theory

Binary multiplication is a crucial arithmetic operation used in digital signal processing, embedded processors, parallel computing, and computer architecture. Because multiplication requires the generation and summation of many partial products, simple implementations can suffer from large propagation delays and excessive hardware usage if computed linearly. Efficient multiplier architectures are therefore critical for improving overall system performance.

One such architecture is the Dadda multiplier, which minimizes the number of reduction stages while preserving fast carry-save addition. In this project, an 8×8 Dadda multiplier was designed and implemented at the gate level using half adders, full adders, and a 16-bit ripple carry adder. The design demonstrates hierarchical construction, starting from basic logic gates to an efficient and optimized multiplier. Functional verification was performed through simulation to validate correctness and evaluate behavior across representative test cases.

1.1 Binary Multiplication

Binary multiplication follows the same process as decimal multiplication. Each bit of one operand multiplies every bit of the other operand to form partial products. For two n-bit numbers, this produces n shifted partial-product rows, which are all added together to compute a final product. The primary design challenge is efficiently summing these rows. However, using simple binary addition tools such as half adders, full adders, and ripple-carry adders can maintain simplicity while optimizing for efficiency if calculations are done in parallel.

1.2 Half Adder Gate Composition

The half adder is the simplest combinational circuit used to add two single-bit inputs. It produces a sum and carry output.

$$S = A \oplus B \tag{1}$$

$$C = A \cdot B \tag{2}$$

Half adders are commonly used in partial-product reduction where no carry-in is present.

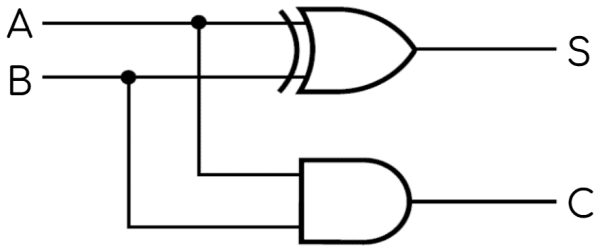

Figure 1: Logic diagram of Half Adder. [1]

1.3 Full Adder Gate Composition

A full adder extends the half adder by including a carry-in input, allowing addition of three single-bit values.

In particular, the sum output follows the relation

$$S = A \oplus B \oplus Cin \tag{3}$$

and the carry output follows

$$Cout = AB + Cin(A \oplus B) \tag{4}$$

Full adders are the primary building blocks for multi-bit addition and are heavily used in multiplier reduction stages to compress three bits into two outputs (a sum and carry).

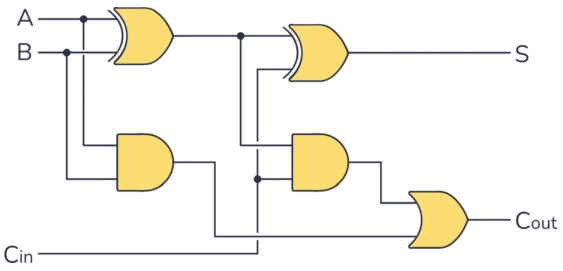

Figure 2: Logic diagram of Full Adder. [2]

1.4 Ripple Carry Adder

Multi-bit binary addition is achieved by cascading multiple full adders, where the carry-out of each stage feeds the carry-in of the next higher-order stage. This process repeats until all of the individual input bits are all added. Finally, the final carry out becomes the most significant bit. The ripple carry adder is used at the end of the Dadda reduction process to combine the final products and sums into the final answer.

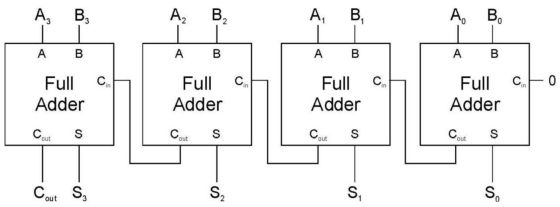

Figure 3: Diagram of 4x4 Ripple Carry Adder. [3]

1.5 Dadda Multiplier

The Dadda multiplier is a tree-based multiplier that reduces partial products using a minimal number of reduction stages. Depending on the height of the partial product columns, full adders and half adders are used for reduction.

1. **Partial Product Generation:** Each bit of the multiplied numbers compute a binary AND operation with each other, creating a matrix of partial products.
2. **Height Target:** Each stage defines a maximum height that the columns of partial products can reach. This limit is defined by:
$$d_{k+1} = [1.5 \times d_k], d_1 = 2 \tag{5}$$
3. **Stage Reduction:** Within each stage:
   - Groups of three bits are reduced using full adders
   - Groups of two bits are reduced using half adders
   - Carries are moved to the next column towards the LSB
4. **Final Addition:** After the reductions to a height of 2, the final rows are combined using any binary adder, such as a ripple-carry-adder.

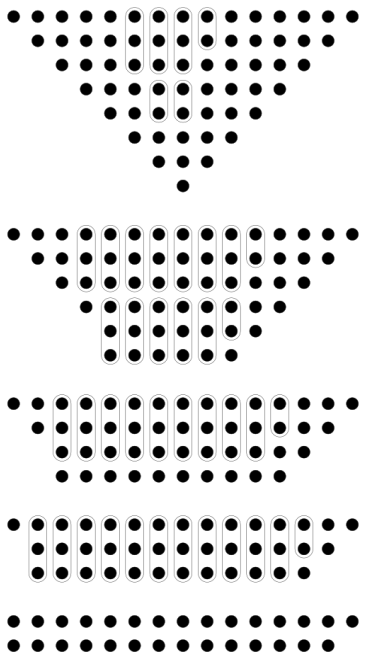

Figure 4: Dot reduction of a 8x8 Dadda multiplier. [4]

2 Results & Analysis

2.1 Code Structure

The 8×8 Dadda multiplier was implemented and verified through testbench simulation. The design followed the Dadda reduction strategy, in which partial products are compressed through multiple carry stages before a final ripple-carry addition. The reduction procedure used in this work follows the Dadda multiplier method described in lecture and on Wikipedia. Simulation was performed using a testbench that compared the hardware output to the expected arithmetic product computed with Verilog's built-in multiplication operator. To view the output, the waveform generated from the testbench was examined and the hexadecimal outputs were checked with the known product. No mismatches or

unwired (X) values were observed during testing.

The multiplier was constructed from the following basic gate-level modules:

- Half Adder (HA) using XOR and AND gates
- Full Adder (FA) using XOR/AND/OR gates
- 16-bit Ripple Carry Adder (RCA) for final addition
- 4-stage 8x8 Dadda reduction algorithm

This modular structure simplified debugging and allowed each module to be validated independently before system integration.

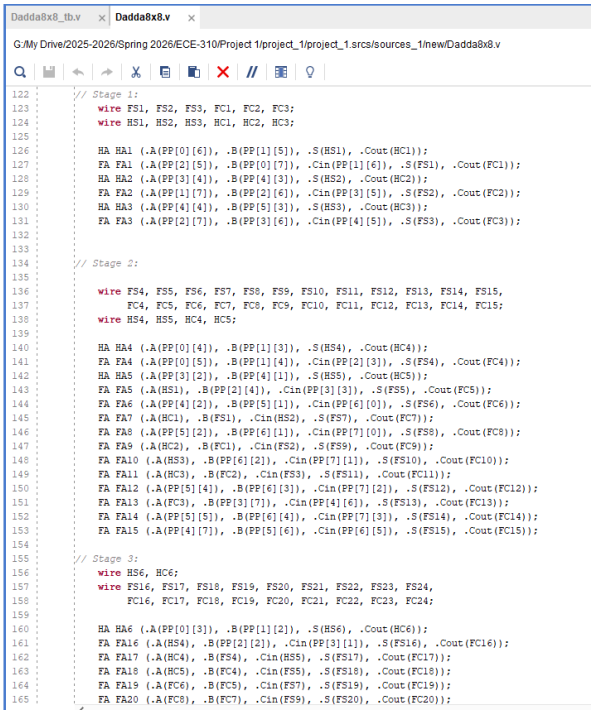

Figure 5: Code for stage 1-3 reductions.

The partial products were generated using a two-dimensional AND gate array. These bits were then grouped by weight and reduced using half and full adders according to Dadda's height limits. Carries were forwarded diagonally to the next column, encouraging carry behavior and preventing long propagation chains and delays during intermediate stages.

During the simulation, intermediate signals confirmed that each stage successfully reduced column heights:

- Stage 1–4 progressively compressed the partial-product matrix
- Each full adder reduced three bits to two
- Each half adder reduced two bits to two
- Carries propagated only between adjacent columns

Because carry propagation was deferred until the final stage, addition occurred in parallel. This significantly reduced delay compared to a standard adder approach, which scales linearly with ever increasing bits. After Stage 4, only two rows remained. These were connected to the 16-bit ripple carry adder for the final sum.

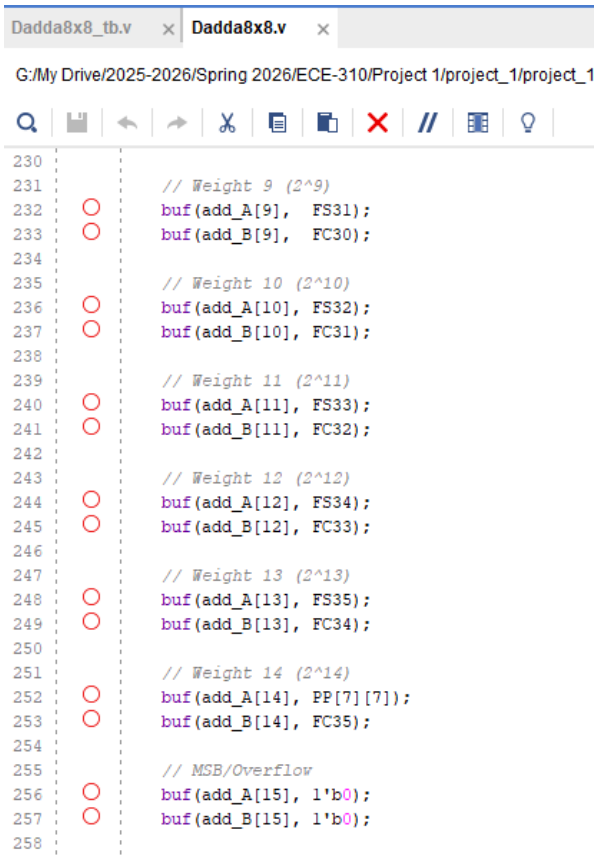

Figure 6: Code for ripple-carry addition of final products.

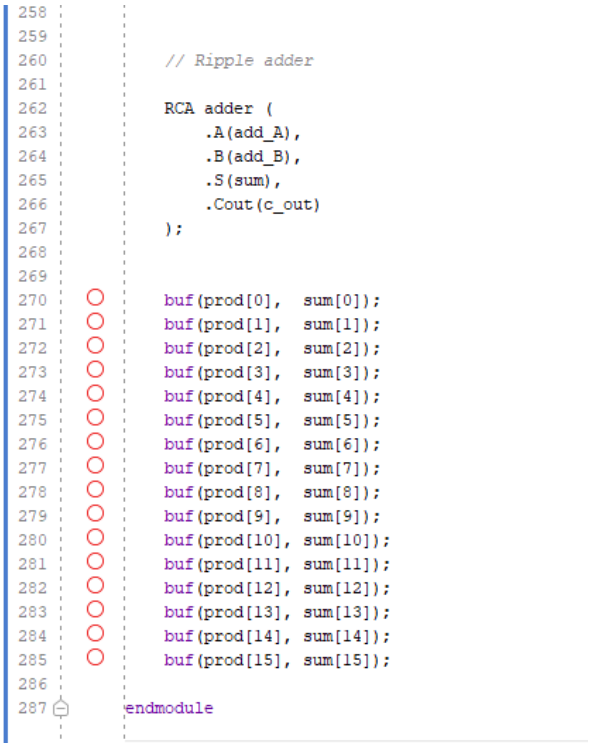

Figure 7: Code for final buffer of final RCA sum to Dadda output.

2.2 Testing & Results

All tested cases produced correct results, confirming both functional correctness and proper carry propagation across all reduction stages and the final ripple carry adder.

Input A	Input B	Product	
0	55	0	
123	0	0	
1	255	0	255
255	1	255	
2	4	8	
8	16	128	
255	255	65025	
127	127	16129	
170	85	14450	
37	91	3367	

Table 1: Vivado testbench results.

A variety of different test cases were used to ensure proper functionality across the following conditions:

1. Zero Property
2. Identity Property
3. Powers of two
4. Alternating binary bit patterns
5. Overflow/carry propagation case
6. Random test cases

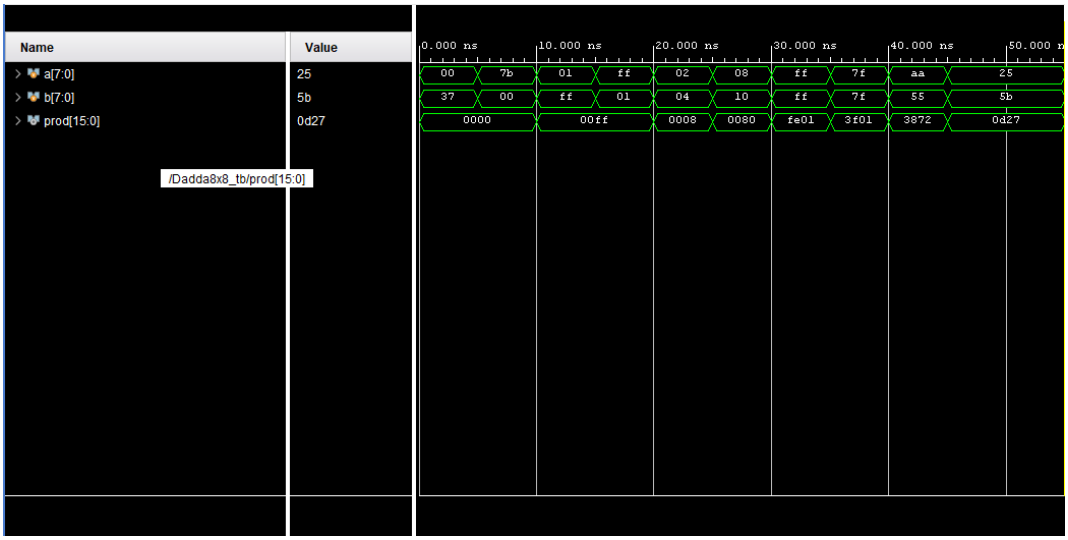

Figure 8: Waveform testbench output.

Waveform inspection further confirmed output behavior. Partial products appeared immediately after input changes, reduction signals were accurate, and the final ripple carry adder resolved the last carry propagation successfully when applicable. The stage reduction gate-level implementation also provided educational insight into how complex digital systems can be are constructed from simple logic components. Overall, the simulation results confirm that the multiplier is both functionally correct and structurally consistent with Dadda reduction theory.

2.3 Initial Design Error and Debugging Process

Although the final implementation functioned correctly, the initial design produced consistently incorrect products. Early results exhibited systematic errors, including outputs that were scaled by factors of two or four, reversed bit orders, or shifted values. At first, these discrepancies suggested minor wiring or indexing mistakes. However, repeated debugging of individual signals did not resolve the issue, indicating that the problem was architectural and flawed in design rather than minor.

Upon revisiting the theory of Dadda reduction, it became clear that the partial-product placement was fundamentally incorrect. In the Verilog implementation, `prod[0]` represents the least significant bit (LSB), and therefore lower-weight partial products must align toward the lower-index columns of the reduction tree. In the original design, the partial products were arranged in the opposite orientation, effectively mirroring the entire partial-product matrix about the center column. This caused carries and sums to propagate in incorrect directions, leading to widespread and non-intuitive output errors.

Figure 9 shows a sketch of the original Stage 1 layout. The mirrored arrangement misaligned bit weights, so the reduction network combined signals corresponding to different significance levels. As a result, even though individual adders operated correctly, the overall arithmetic result was incorrect.

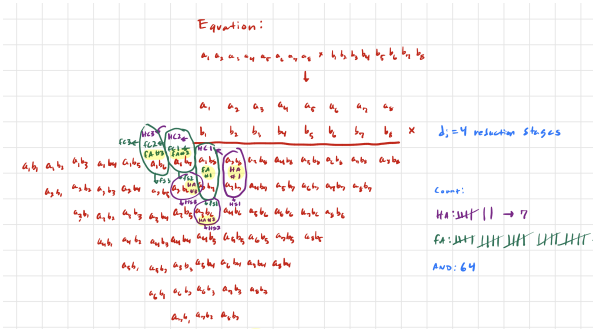

Figure 9: Initial Stage 1 design with incorrectly mirrored partial-product placement.

After correcting the orientation of the partial products and realigning each column with its proper binary weight, the reduction stages behaved as expected. Carries propagated diagonally toward higher significance bits, and the final ripple carry adder produced the correct product for all test cases. The corrected design is shown in Figure 10.

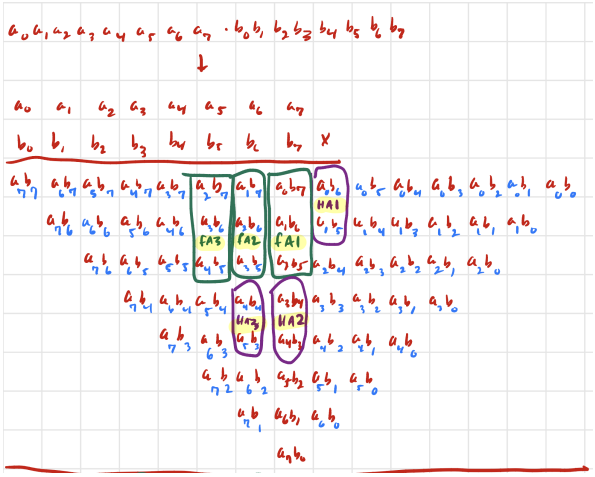

Figure 10: Corrected Stage 1 design with properly aligned partial products.

This debugging process emphasized how in tree-based multipliers, correct bit-weight alignment is just as critical as the correctness of individual adders. Even a fully functional reduction network will fail if partial products are not positioned correctly.

3 Conclusions

In this project, an 8×8 Dadda multiplier was successfully designed, implemented, and verified at the gate level. The multiplier was built using half adders, full adders, and a ripple carry adder at the end, reinforcing the concept that complex arithmetic circuits can be constructed from simple combinational logic gates. Partial products were generated using AND gates and compressed through multiple reduction stages utilizing the Dadda reduction system, resulting in only two final rows that were summed using a carry-propagate adder. Simulation results confirmed correct functionality for a few tested input cases, including zero values, powers of two, carry propagation overflow, alternating bit patterns, and random cases. No incorrect outputs or unknown states were observed.

The project highlights the effectiveness of tree-based reduction techniques for high-performance multiplication and provides practical experience in structural hardware design, modular development, and verification. Future improvements could include replacing the ripple carry adder with a faster carry-lookahead or Kogge-Stone adder, optimizing the design for larger bit widths, or using dataflow or behavioral-level modeling for better code efficiency.

References

[1] Build Electronic Circuits, "Half adder circuit diagram and truth table," n.d. Accessed: 2026-02-17.

[2] Build Electronic Circuits, "Full adder circuit diagram and truth table," n.d. Accessed: 2026-02-17.

[3] All Things VLSI, "4-bit ripple carry adder," 2013. Accessed: 2026-02-17.

[4] Wikipedia contributors, "Dadda multiplier," 2025. Accessed: 2026-02-17.

This report was compiled using LaTeX .