8×8 Dadda Multiplier (transistor-level)

Carlet Pierrelouis
dept. Electrical and Computer Engineering
Purdue University
Indianapolis, US
cpierrel@purdue.edu

***Abstract*— In designing an 8×8-bit digital multiplier for a high-performance RISC microprocessor ALU, a key engineering concern was optimizing for limited chip area while maintaining competitive speed. This project utilizes Dadda's multiplier architecture due to its space-efficient partial product reduction structure, making it ideal for integration into a chip under tight area constraints. The design leverages 64 AND gates for product term generation and incorporates a custom Manchester adder for final summation. While Booth encoding could have been employed to generate the product terms for improved speed and efficiency, this implementation focuses on feasibility under project constraints. The resulting design achieves a propagation delay of 51.6235ps and an average power dissipation of 1.209µW, balancing performance and simplicity for next-generation integration.**

I. Introduction

The design and optimization of multipliers play a pivotal role in digital systems, as multiplication remains one of the most computationally expensive operations compared to addition and subtraction. High-performance multipliers are integral to applications such as image and video processing, cryptography, and machine learning, where speed, power efficiency, and area optimization are critical. Among various architectures, the Dadda multiplier is recognized for its efficient partial product reduction process, requiring fewer levels compared to the Wallace tree multiplier, which makes it an attractive choice for high-speed applications [1], [2].

In this project, we propose the design of an 8×8-bit Dadda multiplier using a custom Manchester 8-bit adder in the final addition stage. The Manchester adder, though simple, allows for a functional implementation within the time constraints of this assignment. While this design serves as a feasibility study, it is not optimized for efficiency due to the time constraints. For instance, our transistor counts before incorporating the Manchester adder is approximately 708 transistors, which could have been significantly reduced with a more systematic approach.

Our full adders are implemented with 42 transistors each. By utilizing two half adders, this count could have been reduced to 24 transistors per full adder, improving the area efficiency of the reduction tree [1].
Partial Product Generation: While we use straightforward AND gates to generate the partial products, employing Booth encoding could have reduced the number of partial products and improved both speed and area usage [3].
Despite these inefficiencies, this design demonstrates the feasibility of the Dadda multiplier architecture under practical constraints. Analytical calculations and Cadence simulations are presented to evaluate key performance metrics, including delay and power consumption. Although not the most optimized solution, this project lays the groundwork for future enhancements, such as integrating Booth encoding for partial product generation and designing more efficient full adders.

II. Methodology

The implementation of the 8×8-bit Dadda multiplier (fig. 1) was carried out through a systematic process involving partial product generation, Dadda reduction, and final addition using a custom-designed Manchester adder.

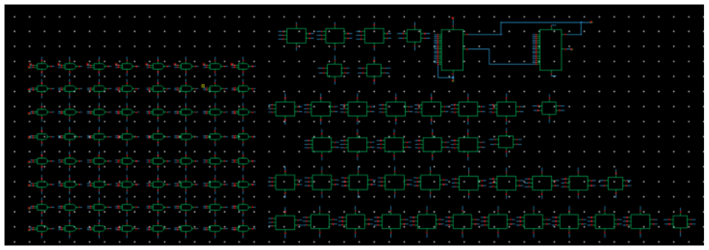


Figure 1. Top level hierarchy of 8x8 Dadda multiplier

A. Partial Product Generation

The partial product generation stage was implemented using 64 *AND* gates, where each gate computes the logical *AND* of one bit from the multiplicand (A) and one bit from the multiplier (B). The resulting partial product terms are arranged into an 8×8 matrix, as shown below:

$$P_{i,j} = A[i] \wedge B[j], \qquad \forall i,j \in \{0,1,..,7\}.$$

These terms serve as inputs to the reduction tree in the next stage. The generation of the partial products ensures the completeness of the input data for further processing, with all $A[i] \wedge B[j]$terms included.

B. Dadda Reduction

The Dadda reduction stage reduces the partial product matrix to two rows suitable for final addition. This is done in five stages, progressively lowering the row count according to the Dadda algorithm:

$$d_{k+1} = \left\lfloor \frac{3}{2} \cdot d_k \right\rfloor$$

Starting with d = 6, the thresholds decrease as d=6,4,3,2, ensuring efficient use of hardware and minimal delay.
The reduction process involves full adders (FAs) and half adders (HAs) shown in fig. 2. to minimize the row count in each column.

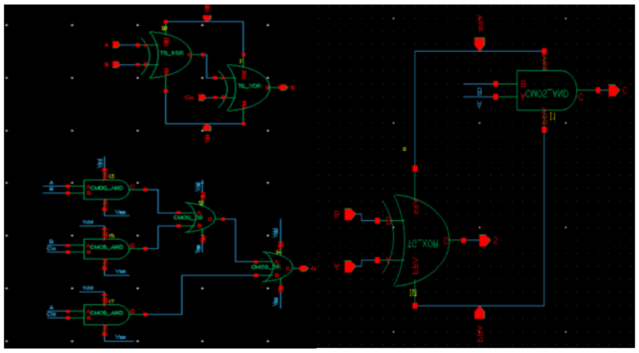
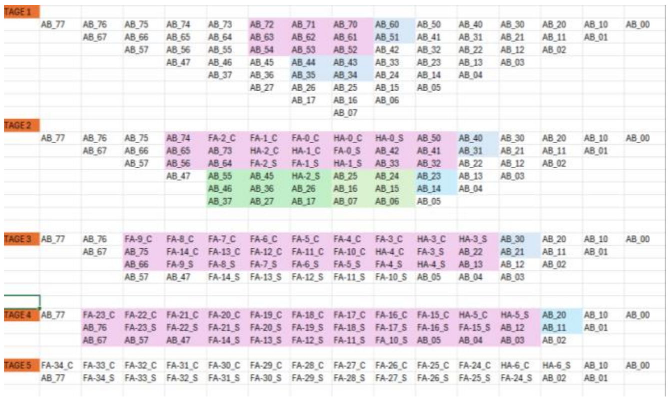


Figure 2. Full adder (left) and Half adder (right).



TAGE 1															
	AB_77	AB_76	AB_75	AB_74	AB_73	AB_72	AB_71	AB_70	AB_60	AB_50	AB_40	AB_30	AB_20	AB_10	AB_00
		AB_67	AB_66	AB_65	AB_64	AB_63	AB_62	AB_61	AB_51	AB_41	AB_31	AB_21	AB_11	AB_01	
			AB_57	AB_56	AB_55	AB_54	AB_53	AB_52	AB_42	AB_32	AB_22	AB_12	AB_02		
				AB_47	AB_46	AB_45	AB_44	AB_43	AB_33	AB_23	AB_13	AB_03			
					AB_37	AB_36	AB_35	AB_34	AB_24	AB_14	AB_04				
						AB_27	AB_26	AB_25	AB_15	AB_05					
							AB_17	AB_16	AB_06						
								AB_07							
TAGE 2															
	AB_77	AB_76	AB_75	AB_74	FA-2_C	FA-1_C	FA-0_C	HA-0_C	HA-0_S	AB_50	AB_40	AB_30	AB_20	AB_10	AB_00
		AB_67	AB_66	AB_65	AB_73	HA-2_C	HA-1_C	FA-0_S	AB_42	AB_41	AB_31	AB_21	AB_11	AB_01	
			AB_57	AB_56	AB_64	FA-2_S	FA-1_S	HA-1_S	AB_33	AB_32	AB_22	AB_12	AB_02		
				AB_47	AB_55	AB_45	HA-2_S	AB_25	AB_24	AB_23	AB_13	AB_03			
					AB_46	AB_36	AB_26	AB_16	AB_15	AB_14	AB_04				
					AB_37	AB_27	AB_17	AB_07	AB_06	AB_05					
TAGE 3	AB_77	AB_76	FA-9_C	FA-8_C	FA-7_C	FA-6_C	FA-5_C	FA-4_C	FA-3_C	HA-3_C	HA-3_S	AB_30	AB_20	AB_10	AB_00
		AB_67	AB_75	FA-14_C	FA-13_C	FA-12_C	FA-11_C	FA-10_C	HA-4_C	FA-3_S	AB_22	AB_21	AB_11	AB_01	
			AB_66	FA-9_S	FA-8_S	FA-7_S	FA-6_S	FA-5_S	FA-4_S	HA-4_S	AB_13	AB_12	AB_02		
			AB_57	AB_47	FA-14_S	FA-13_S	FA-12_S	FA-11_S	FA-10_S	AB_05	AB_04	AB_03			
TAGE 4	AB_77	FA-23_C	FA-22_C	FA-21_C	FA-20_C	FA-19_C	FA-18_C	FA-17_C	FA-16_C	FA-15_C	HA-5_C	HA-5_S	AB_20	AB_10	AB_00
		AB_76	FA-23_S	FA-22_S	FA-21_S	FA-20_S	FA-19_S	FA-18_S	FA-17_S	FA-16_S	FA-15_S	AB_12	AB_11	AB_01	
		AB_67	AB_57	AB_47	FA-14_S	FA-13_S	FA-12_S	FA-11_S	FA_10_S	AB_05	AB_04	AB_03	AB_02		
TAGE 5	FA-34_C	FA-33_C	FA-32_C	FA-31_C	FA-30_C	FA-29_C	FA-28_C	FA-27_C	FA-26_C	FA-25_C	FA-24_C	HA-6_C	HA-6_S	AB_10	AB_00
	AB_77	FA-34_S	FA-33_S	FA-32_S	FA-31_S	FA-30_S	FA-29_S	FA-28_S	FA-27_S	FA-26_S	FA-25_S	FA-24_S	AB_02	AB_01	

Table 1. Stages of the Dadda Reduction Process. This table illustrates the systematic reduction of the partial product matrix at each stage, with clear annotations for the use of FAs and HAs.

Table 1 illustrates this process:

- Blue cells: Represent HAs (reduce two rows to a sum and carry bit).
- Purple and green cells: Represent FAs (reduce three rows to a sum and carry bit).

Once reduced, the Manchester adder computes the final 16-bit product. Although less efficient than advanced parallel-prefix adders like Kogge-Stone, the Manchester adder provides a lightweight and practical solution under the project's time constraints. The final adder outputs the weighted sum of the two rows, resulting in the final product.

III. Simulation Results

A. *Waveform Analysis*

The transient simulation results for the product (P) are shown in fig. 3, where the expected results for various input combinations of A and B are compared against the observed waveform output. The expected product values were derived from the vector file data, while the observed values were extracted from the waveform simulation.

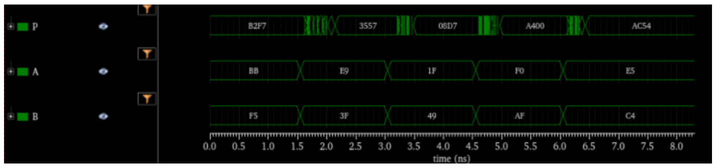


Figure 3. Waveform for P, A, and B

Table 2 highlights discrepancies between the expected and observed results for specific input cases. While the circuit produces correct outputs for some cases (e.g., 0.0–2.0 ns and 4.0-6.0 ns), other cases exhibit deviations. These mismatches could be attributed to noise or propagation issues within the reduction tree and final addition stages.

Table 2: Expected vs. Observed Results for Product (PPP)

Time (ns)	**AAA (Input)**	**BBB (Input)**	**Expected PPP (Hex)**	**Observed PPP (Hex)**
0.0–2.0	BB	F5	B2F7	B2F7
2.0–4.0	E9	3F	3957	3557
4.0–6.0	1F	49	08D7	08D7
6.0–8.0	F0	AF	A410	A400
8.0–10.0	E5	C4	AF54	AC54

B. *Power Dissipation*

The red graph in fig. 4 represents the instantaneous power dissipation of the multiplier circuit during simulation. Using Cadence's calculator tool, the average power dissipation was calculated by averaging the instantaneous power values over the entire simulation period. The result is as follows:

- Average Power Dissipation: 1.209 µW

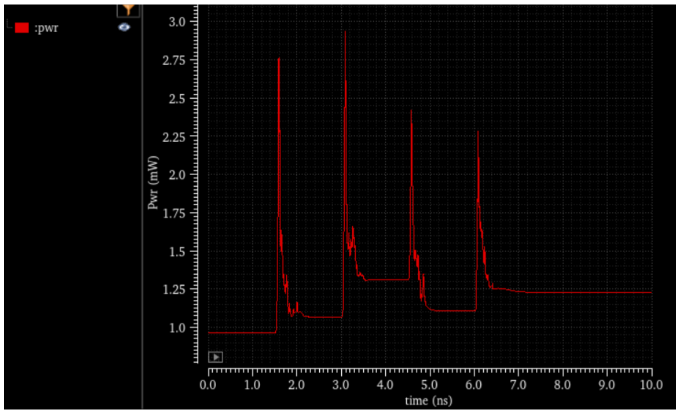
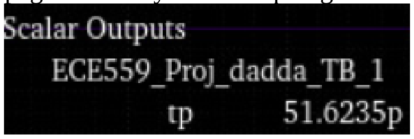


Figure 4. Power dissipation graph over time, highlighting instantaneous power peaks during transitions.

C. *Propagation Delay*

The propagation delay was measured by comparing the transition of input A[0] (0th bit) to the output P[0] (0th bit) using the scalar tool in Cadence. The measured delay was:

- Propagation Delay: 51.6235 ps fig. 5.



```
Scalar Outputs
    ECE559_Proj_dadda_TB_1
        tp          51.6235p
```

Figure 5. Scalar output showing the calculated propagation delay.

IV. Observations and Discussion

The waveforms for the product (P) exhibited significant noise during transitions, as shown in fig. 6. These transient instabilities are likely caused by glitches in the intermediate stages of the Dadda multiplier, particularly in the reduction tree. The noise indicates that not all signals stabilize simultaneously, which is a common issue in combinational circuits with multiple levels of logic. While the circuit eventually stabilizes to correct results for some cases, the transient instability impacts correctness in others, as highlighted in Table 2.

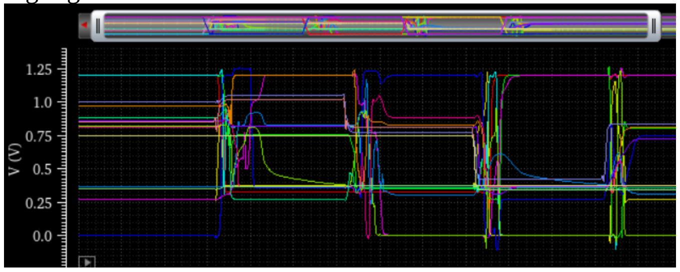


Figure 6. Waveform for Product (p) Output Showing Transient Noise During Transitions.

This behavior underscores the need for:

1. Improved Synchronization: Introducing pipeline registers between stages could mitigate glitches and ensure stable intermediate outputs.
2. Optimized Reduction Logic: Reducing propagation delays in the reduction tree could help minimize transient inconsistencies.

The transient noise suggests that additional design optimizations are necessary to ensure reliable operation across all test cases. Future iterations could address these issues by implementing a more robust reduction tree.

V. Conclusion

In this project, we designed and implemented an 8×8 Dadda multiplier to evaluate its performance and feasibility under practical constraints. Key results included an average power dissipation of 1.209 µW, and a propagation delay of 51.6235 ps. While the multiplier produced correct results for most test cases, transient noise during transitions caused incorrect outputs for certain inputs. These glitches were attributed to propagation delays and intermediate stage instability in the reduction tree and final addition logic.

Although the design demonstrated the viability of the Dadda multiplier architecture, it revealed several areas for improvement. Future iterations could incorporate Booth encoding to reduce partial product terms, optimize adders to minimize transistor count, and introduce pipelining to ensure stable outputs. This project provides a solid foundation for further enhancements, balancing feasibility and performance in multiplier design.

References

[1] S. Knowles, “A family of adders,” in Proc. 14th IEEE Symp. Computer Arithmetic, Apr. 1999, pp. 277–281.

[2] A. Kumar, A. Kumar, and D. Nand, "Design and Study of Dadda Multiplier by using 4:2 Compressors and Parallel Prefix Adders for VLSI Circuit Designs," 2021 2nd International Conference for Emerging Technology (INCET), Belagavi, India, 2021, pp. 1–5, doi: 10.1109/INCET51464.2021.9456283.

[3] G. Rohith, P. Jagadeesh, and N. Meenakshisundaram, "Implementation of 8-bit Dadda Multiplier by Using 4:2 Compressors and Brentkung Adder for Reducing Propagation Delay in Comparison with Kogge Stone Adder," 2024 Ninth International Conference on Science Technology Engineering and Mathematics (ICONSTEM), Chennai, India, 2024, pp. 1–6, doi: 10.1109/ICONSTEM60960.2024.10568807.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.