Approximate Dadda Multiplier

Shrihari Gokulachandran, Heejoo Roh, Gillian Yost, Daniel Peralta Velazquez
EID: sg56476, hr8742, gay234, dup83

***Abstract*—Approximate computation circuits are a popular way to increase performance and reduce area in applications with error resilience. In this work we propose a way to possibly improve Dadda multipliers by using approximate full adders. We replaced the full adders in four different configurations: replacing all full adders in the last stage, replacing all the full adders in the stage with the least amount of full adders, replacing all the full adders in the stage with the most number of full adders, and replacing 1/4 of the full adders in the least significant bits. We used three different approximate full adders and compared the results of the configurations and approximate full adders for 8 bit and 16 bit Dadda multipliers. We used System Verilog test benches to compare average error and synthesized the Dadda multipliers to compare power, performance, and Area. From our results we found that the most accurate configuration was replacing 1/4 of the full adders in the least significant bits. The 16 bit Dadda multiplier had the least error overall error and replacing the least significant bits had very little difference in error across the different approximate adders. Overall, LSB achieved up to 20.2% power reduction, 13.5% complexity reduction, and 15% less area while maintaining the same gate delay and 0.3% accuracy drop.**

I. Problem

Approximate computational circuits have emerged as a popular way to efficiently improve the performance and reduce the area in error resilient applications [1]. Modern day applications perform several multiply operations, especially matrix multiplications while performing CNN and MLP computations. Such applications are resilient to errors and besides the software techniques to approximate the Neural Networks, performing approximate computation on the fundamental unit of computation would significantly improve the performance and area at the cost of increased error which does not affect the end result. In edge computing, we are limited by the area of dedicated hardware units for performing matrix multiplications, so apart from experimenting with data flows and novel architectures for performing matrix computations, we focus on the fundamental unit for matrix operations - Multiplier [2] [3]. Approximate computing can reduce the design complication with an increase in performance.

II. Background

A. Approximate Adders

Adders are the building blocks of operations in computing. They are especially important in multiplication and are the speed-limiting element of multipliers as well. Ramasamy et al. [4] proposed four different approximate full adders. The sum term is calculated by inverting the carry out in each of the four approximate adders. The truth table for each is shown in Table 1.

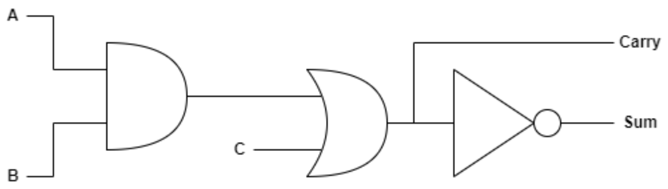

Fig. 1: Carry Based approximate Adder1 (CBAA1)

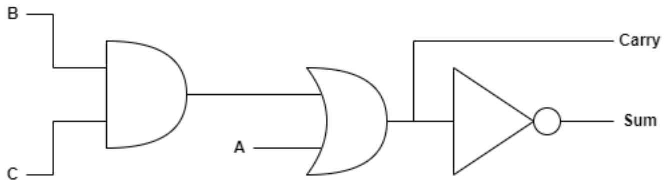

Fig. 2: Carry Based approximate Adder2 (CBAA2)

Carry based approximate adder1 (CBAA-type I)

This uses 1 AND gate, one OR gate, and one NOT gate. The gate diagram is shown in Figure 1. The equations for the approximate full adder are shown in equations 1 and 2.

$$Carry = (a.b) + c \tag{1}$$

$$Sum = \ carry \tag{2}$$

There is 1 error in the carry value and 3 errors in the sum value with a total error distance of 4.

Carry based approximate adder2 (CBAA-type II)

This uses 1 AND gate, one OR gate, and one NOT gate. The gate diagram is shown in Figure 2. The equations for the approximate full adder are shown in equations 3 and 4.

$$Carry = (b.c) + a \tag{3}$$

$$Sum = \ carry \tag{4}$$

There is 1 error in the carry value and 3 errors in the sum value with a total error distance of 3.

Inputs			Accurate outputs		Approximate outputs											
					CBA 1			CBA2			CBA3			CBA4		
A	B	C	SUM	CARRY	S	C	ED	S	C	ED	S	C	ED	S	C	ED
0	0	0	0	0	1	0	1	1	0	1	1	0	0	1	0	1
0	0	1	1	0	0	1	2	1	0	0	0	1	2	1	0	0
0	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0
0	1	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0
1	0	0	1	0	1	0	0	0	1	2	1	0	0	1	0	0
1	0	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0
1	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0
1	1	1	1	1	0	1	1	0	1	1	1	0	1	0	1	1

TABLE I: Truth Table for Conventional Adder and Carry Based Approximate Adders 1-4

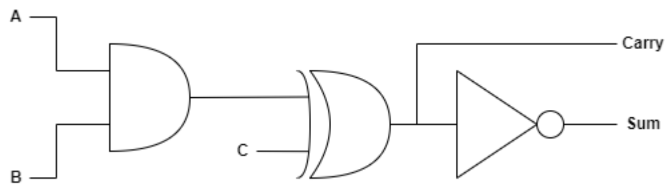

Fig. 3: Carry Based approximate Adder3 (CBAA3)

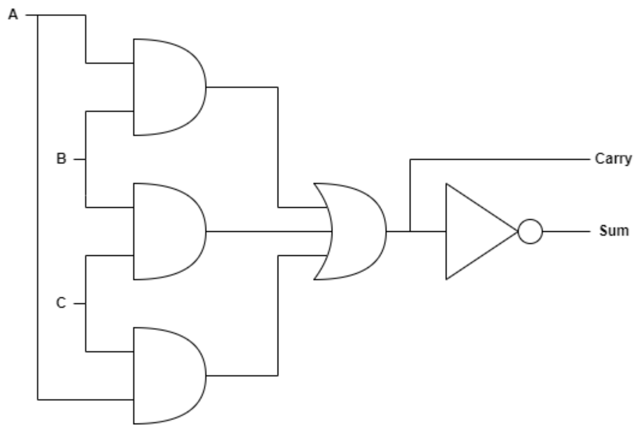

Fig. 4: Carry Based approximate Adder3 (CBAA4)

Carry based approximate adder3 (CBAA-type III)

This uses 1 AND gate, one XOR gate, and one NOT gate. The gate diagram is shown in Figure 3. The equations for the approximate full adder are shown in equations 5 and 6.

$$Carry = (a.b) \oplus c \tag{5}$$

$$Sum = \ carry \tag{6}$$

There is 2 errors in the carry value and 1 error in the sum with a total error distance of 3.

Carry based approximate adder4 (CBAA-type IV)

This uses 3 AND gates, two OR gates, and one NOT gate. The gate diagram is shown in Figure 4. The equations for the approximate full adder are shown in equations 7 and 8.

$$Carry = (a.b) + (b.c) + (c.a) \tag{7}$$

$$Sum = \ carry \tag{8}$$

There is no errors in the carry value and 2 errors in the sum with a total error distance of 2.

B. Dadda Multipliers

In the field of fast-multipliers Dadda multipliers are a strong option for tree-like multiplication [5]. Dadda makes use of full adders and half adders as their building block for the multistage pipeline of partial product computation and reduction.

The full adders and half adders logic is presented in Figures 5 6. Full adders and half adders yield high precision with a corresponding 9 gate count for full adders and 4 gates for half adders. Unlike other fast multiplier approaches (e.g. Wallace)

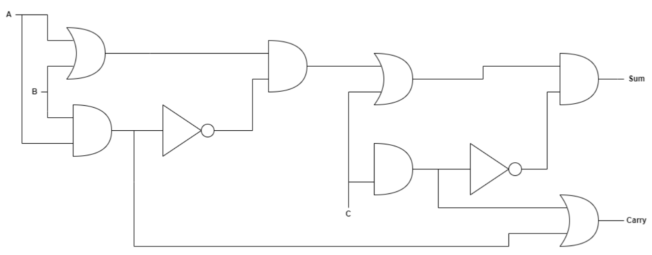

Fig. 5: Full Adder Logic

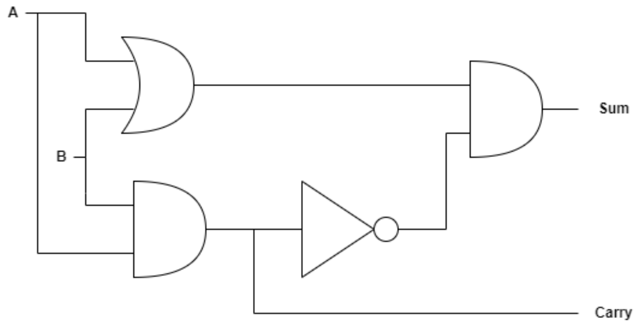

Fig. 6: Half Adder Logic

Dadda's approach consists of reducing the number of full and half adders needed to execute each stage. The novel idea behind Dadda consists of having a limited height set for each stage 9. By using (3,2) full adders we are effective reducing the height from 3 to 2 or $2/3$. On larger input the number of stages increase with all input sizes converging to a 2 input reduction, the last height reduction is then feed to a carry propagate adder logic block to obtain the final product.

$$height = \{2, 3, 4, 6, 9, 13, .., 1.5 * (n - 1)\} \tag{9}$$

C. Approximate Multipliers

Researches have reported work on approximate multipliers. Approximate Wallace Tree Multiplier (AWTM) [6] was presented by Bhardwaj et el. [7] In order for the approximate multiplier to exhibit high accuracy, certain extent of MSBs of the final product needs to be accurate. Breaking down the multiplicand and multiplier bits into the two section equally in the bit size, A_{H} X_{H}, the higher b bits, and A_{L} X_{L}, the lower b bits. AWTM distinguished partial products for A_{H} X_{H}, A_{H} X_{L}, A_{L} X_{H}, and A_{L} X_{L}. It had accurate adders for partial products of A_{H} X_{H}, but approximate adders for other partial products, and achieved the mean accuracy of 99.85% with 39% reduction in power and 30% reduction in area.

In our approach, we replaced the full adders with approximated adders in 8 bit and 16 bit Dadda multiplier [5]. We examined different design approaches for selecting which adder to replace.

III. Approach

In this study, we have implemented 8 and 16-bit Dadda multipliers using System Verilog. Additionally, we have introduced CBAA2, CBAA3, and CBAA4 approximate adders as replacements for specific full adders in the Dadda multipliers. Four different configurations were created by replacing the full adders with approximate adders to produce different approximate multipliers. Two of the configurations aimed to minimize the delay, while the other two aimed to maximize the area reduction in complexity. To evaluate the performance of the configurations, simulations were conducted for all four configurations of 8 and 16 bit multiplier with the three approximate adders and a regular Dadda multiplier of 8 and 16 bits.

A. Approach to Reducing Delay

For having a delay benefit in the Dadda multiplier an entire stage of full adders would need to be replaced with the approximate full adders. If the entire stage is not replaced then there is no delay benefit overall because the multiplier would still need to wait for those full adders to finish. Therefore for maximizing delay we have two configurations to compare.

The configurations are:

- Replace all full adders in the stage with the least amount of full adders (LFA).
  - **8bit:** 3 full adders in the 1st stage in Dadda multiplier has been replaced with approximate adders
  - **16bit:** 8 full adders in the 1st stage in Dadda has been replaced with approximate adders
- Replace all full adders in the last stage with approximate full adders (LS).
  - **8bit:** 11 full adders in the 4th stage in Dadda multiplier has been replaced with approximate adders
  - **16bit:** 27 full adders in the 6th stage in Dadda has been replaced with approximate adders

The changes in number of gates in the 8 and 16 bit Dadda multipliers are shown in Table II.

B. Approach to Reducing Area and Power

In order to design area focused and power focused design we have to aggressively use approximate full adders. At the same time, it is essential to keep good number of accurate full adders for accuracy. We simulated two different configurations that replaces around 1/4 of full adders in each 8bit and 16 bit design. One of the configurations does not necessarily accelerate on the delay ,because only the partial adders of the whole stage are replaced.

The configurations are:

- Replace all full adders in the in the stage with the most amount of full adders (MFA)
  - **8bit:** 12 full adders in the 2nd stage in Dadda multiplier has been replaced with approximate adders.
  - **16bit:** 51 full adders in the 3rd stage has been replaced with approximate adders.
- Replace 1/4 of the full adders in the least significant bits. The approximate adders were added vertically across the every stages.
- **8bit:** The full adders regarding last least significant 7 bits across the stages have been replaced with approximate adders. Total of 9 adders were replaced. Depicted in **Figure 7**
- **16bit:** The full adders regarding least significant 13 bits across the stages have been replaced with approximate adders. Total of 53 adders were replaced. Depicted in **Figure 8**

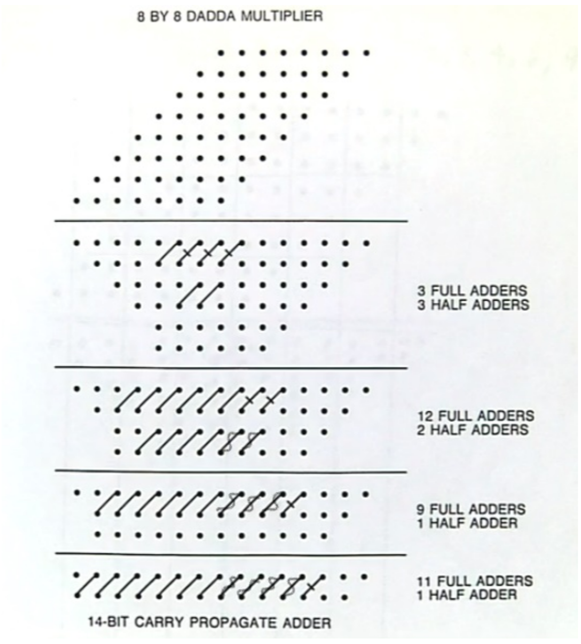

Fig. 7: Showing Which Full Adders are replaced by approximate adders in 8 bit LSB (squiggles is the approximate adder)

The changes in number of gates in the 8 and 16 bit Dadda multipliers are shown in Table III.

TABLE II: Delay Optimization Focus

8-bit Dadda Multiplier Gate Count

Config	Precise Dadda	CBAA#2	CBAA#3	CBAA#4
LFA	525	507	507	513
LS	525	459	459	481

16-bit Dadda Multiplier Gate Count

Config	Precise Dadda	CBAA#2	CBAA#3	CBAA#4
LFA	2218	2170	2170	2186
LS	2218	2056	2056	2110

TABLE III: Complexity Optimization Focus

8-bit Dadda Multiplier Gate Count

Config	Precise Dadda	CBAA#2	CBAA#3	CBAA#4
MFA	525	453	453	477
LSB	525	471	471	489

16-bit Dadda Multiplier Gate Count

Config	Precise Dadda	CBAA#2	CBAA#3	CBAA#4
MFA	2218	1912	1912	2014
LSB	2218	1918	1918	2018

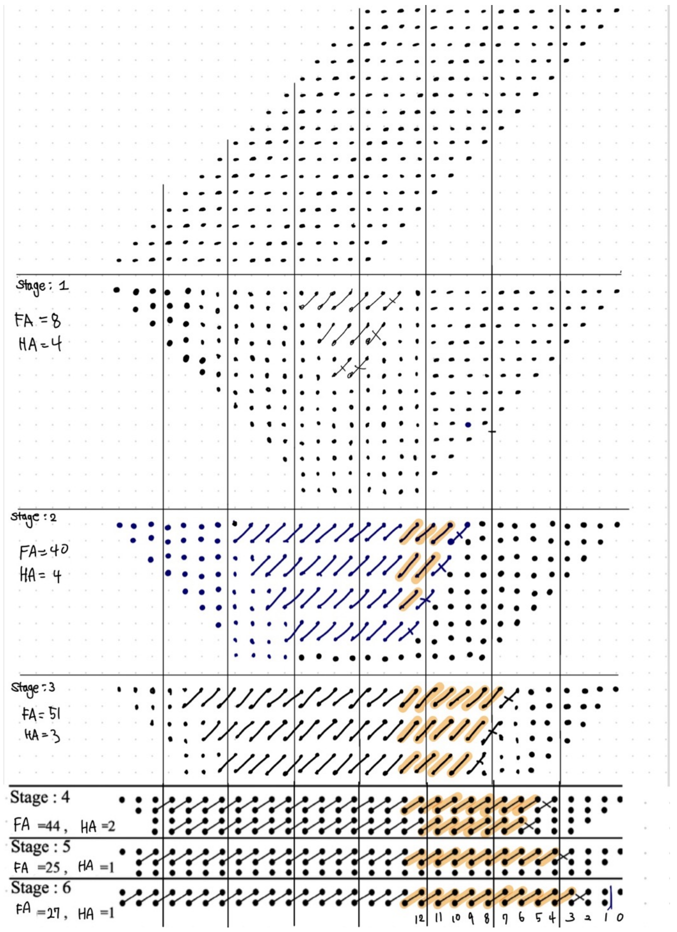

Fig. 8: Showing Which Full Adders are replaced by approximate adders in 16 bit LSB approach, the highlighted adders were replaced

IV. Methodology

A. Simulation

For the 8-bit and 16-bit accurate and approximate Dadda multipliers, we used System Verilog test benches to carry out Constrained Random Vector based simulations with identical seed value across configurations. The constraints for the random test vectors were based on the maximum possible values for the unsigned inputs and the products for the 8-bit and 16-bit Dadda Multipliers. The Relative Error for each test vector is evaluated and accumulated to obtain the Average Error over all the random test vectors for the given configuration. Siemens QuestaSim was used to perform simulations

The range of inputs for the 8-bit Dadda Multiplier were constrained in the interval 1-255 and also with the condition that the resulting product would be less than or equal to 32,767. The 16-bit Dadda Multiplier had ihe inputs constrained in the interval 1-65,535 and also with the condition that the resulting product would be less than or equal to 2,147,483,647 to avoid overflows while estimating the relative errors. 100 constrained random test vectors were generated and the relative error *RE*

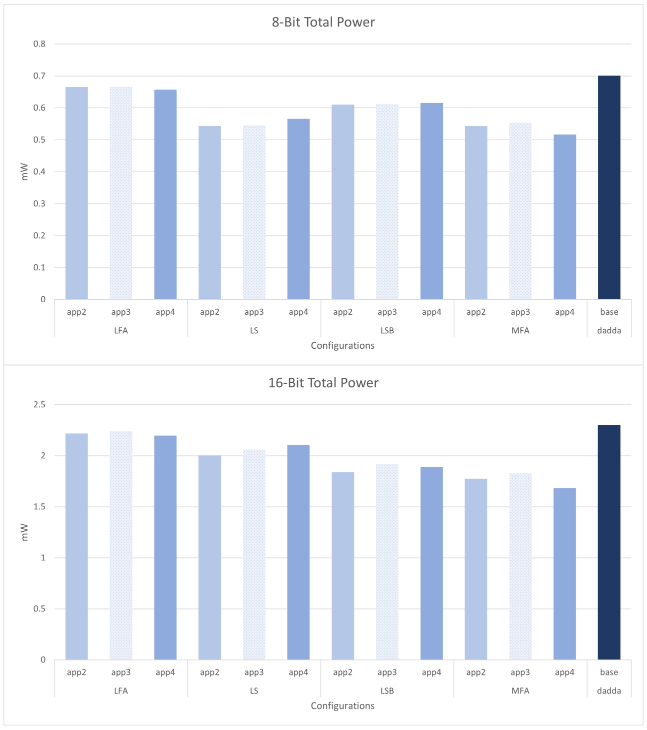

Fig. 9: Power Results

for each test vector i was estimated using the equation 10.

$$RE_i = \frac{|Product_{actual} - Product_{approximate}|}{Product_{actual}} * 100 \quad (10)$$

The average relative error over all the test vectors *AE* is calculated using the equation 11

$$AE = \frac{\sum_{i=0}^{i=n} RE_i}{n} \quad (11)$$

B. Power, Performance, and Area Estimation

The accurate Dadda multipliers and various configurations of approximate multipliers mentioned in the previous section were synthesized using Synopsys Design Compiler with the 45nm NanGate FreePDK45. A custom version of FuseSoC, a package manager was used to interface with the tools.

V. Results

A. Average Error

As seen in Table IV we can observe that the error is the least for LSB configuration, while LS produces the wrong results. While the approach of using MFA reduces the delay for a stage, there is a considerable error. However, this error would be well acceptable in certain applications like machine learning inference. LSB exhibits the least error and emerges as the best configuration for designing approximate full adders. This shows that while designing approximate full adders, introducing approximate adders from the left in all stages is better than just replacing the adders completely in a stage. This comes at the cost of latency as the critical path would still be that of the Full Adder, but the a similar reduction in area can be achieved.

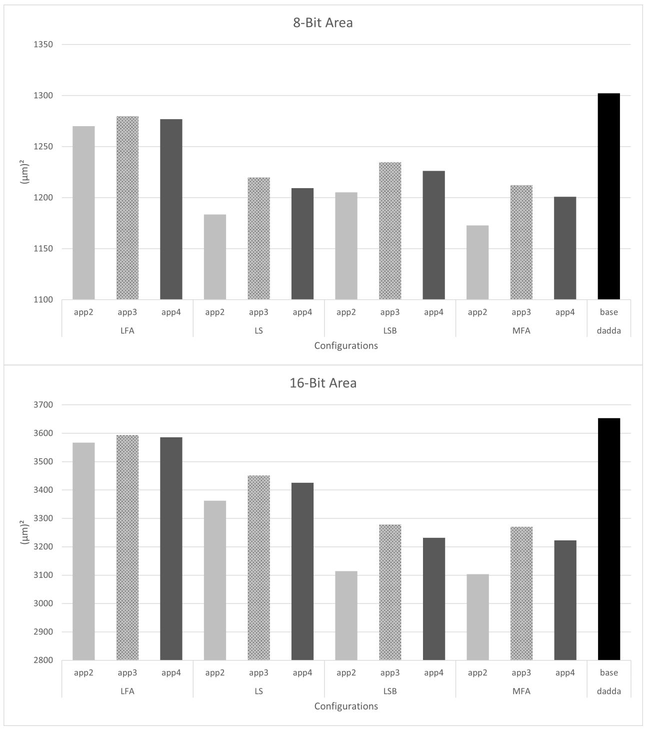

Fig. 10: Area Results

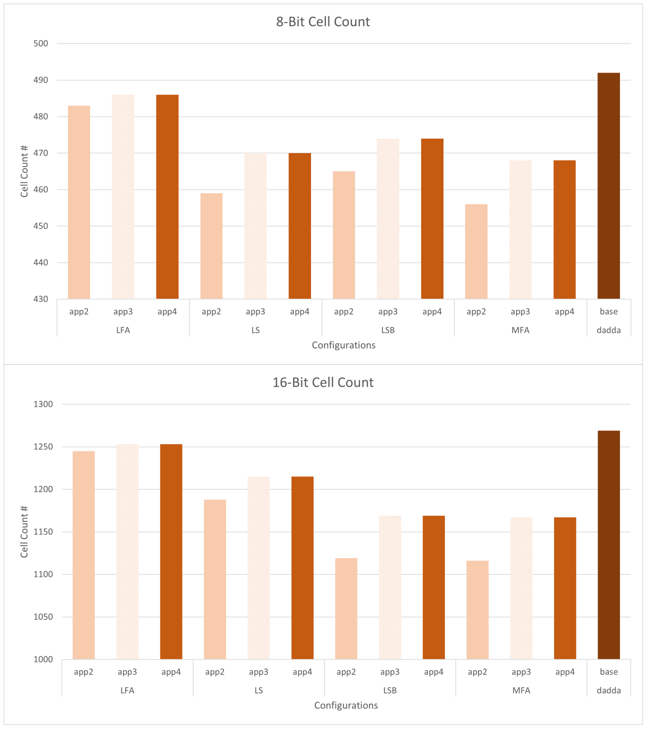

Fig. 11: Cell Count Results

TABLE IV: Average Error

8-bit Dadda Multiplier

Configuration	CBAA#2	CBAA#3	CBAA#4
LFA	54.19%	57.37%	52.70%
LS	1075.18%	1074.52%	1063.85%
MFA	385.5%	386.25%	875.23%
LSB	7.45%	14.73%	6.85%

16-bit Dadda Multiplier

Configuration	CBAA#2	CBAA#3	CBAA#4
LFA	0.24%	0.23%	0.21%
LS	421.49%	422.56%	410.81%
MFA	39.78%	40.50%	35.18%
LSB	0.03%	0.03%	0.03%

B. *Complexity and Delay*

The number of gates was calculated for each configuration, approximate adders, and width of the Dadda Multiplier to compare our approach with a typical Dadda Adder. This is shown in Tables II and III. The delay of each configuration was calculated as well and shown in Tables V and V.

TABLE V: Calculated Gate Delay Results

8-bit Dadda Multiplier

Configuration	**Delay**
Baseline	31
LFA	28
LS	28
MFA	28
LSB	31

16-bit Dadda Multiplier

Configuration	**Delay**
Baseline	43
LFA	41
LS	41
MFA	41
LSB	43

C. *Power, Area and Cell Count*

As seen in Figures 9, 10 and 11, for both 8-bit and 16-bit Dadda Multipliers, the MFA configuration consumes the least power, and area and has the least cell count. LSB closely follows MFA, but the result analysis proves that LSB configuration that performs the best in reducing the Error in addition to the Power Performance and Area metric.

VI. Conclusion

Approximate multipliers in error-resilient applications are best optimized for error, performance, area, and delay when designed by replacing the full adders from the left. Higher the bit-width of the multiplier the errors are less pronounced. We hence put forward the strategies for designing space exploration of approximate Dadda multipliers with importance on sweeping the configuration with approximate adders from the LSB in the last stage and extending to the stages above. This can be further followed by exploring the replacement of the full adders with approximate full adders proceeding towards the MSB but not more than half the bit-width. In this work, we were able to demonstrate different approximate adder configurations to build a Dadda approximate multiplier and yield an overall average error ranging from 1000% - 7% in 8-bit input size and 400% - 0.2% in 16-bit. We conclude from these results, the tradeoff between accuracy and resources is to be considered and carefully designed in noise resilient applications taking our proposed designs into account. The results presented showcase Least Significant Bit (LSB) replacement design as the optimal configuration with 13.5% complexity reduction, 15% area reduction and 20.2% power reduction while maintaining equal gate delay and $\tilde{1}$% accuracy error.

References

[1] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, 2004.

[2] E. Swartzlander, "Truncated multiplication with approximate rounding," in *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers (Cat. No.CH37020)*, vol. 2, 1999, pp. 1480–1483 vol.2.

[3] S. Venkatachalam and S.-B. Ko, "Design of power and area efficient approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1782–1786, 2017.

[4] M. Ramasamy, G. Narmadha, and S. Deivasigamani, "Carry based approximate full adder for low power approximate computing," *2019 7th International Conference on Smart Computing amp; Communications (ICSCC)*, 2019.

[5] L. Dadda, *Some Schemes for Parallel Multipliers*, 1965, vol. 34.

[6] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, 1964.

[7] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power- and area-efficient approximate wallace tree multiplier for error-resilient systems," in *Fifteenth International Symposium on Quality Electronic Design*, Santa Clara, CA, USA, 2014, pp. 263–269.