The report committee for Wesley Donald Chu
Certifies that this is the approved version of the following report:

Wallace and Dadda Multipliers Implemented Using Carry Lookahead Adders

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: _____
Earl E. Swartzlander, Jr.

_____
Mircea D. Driga

Wallace and Dadda Multipliers Implemented Using Carry Lookahead Adders

By:

Wesley Donald Chu, B.S.E.E

Report

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2013

Acknowledgements

I would like to express my deep gratitude to Professor Earl Swartzlander for his patient guidance and enthusiastic encouragement for this report. I would also like to thank the entire University of Texas at Austin ECE department for their continued excellence in teaching. Finally, I would like to thank my family for their support and encouragement throughout my study.

Abstract

Wallace and Dadda Multipliers Implemented Using Carry Lookahead Adders

By Wesley Donald Chu, MSE

The University of Texas at Austin, 2013

Supervisor: Earl E. Swartzlander, Jr.

In a previous paper it was shown that the use of carry lookahead adders can reduce the delay it takes to compute the product with Wallace and Dadda multipliers. In this report, a more detailed analysis is provided on the use of carry lookahead adders to implement Dadda multipliers and Wallace multipliers. These two styles present different ways to use the arithmetic components and offer different benefits. Implementations of each style of multiplier are presented in this report. The performance delay and complexity of the implementations is compared and analyzed.

Table of Contents

List of Tables

List of Figures

1.0 Introduction

Multiplication is the second most used arithmetic operation after addition, which has resulted in a large research interest in developing ways to improve the performance of multipliers. Multipliers have complex designs as a result of the large number of partial products that are formed during a multiplication; however, the general process can be broken down into three steps. The first step is to generate the partial product matrix. An example of a partial product matrix is shown in Figure 1. Each partial product is generated with an AND gate. As a result, N^2 AND gates are required in an N by N multiplier. The second step (referred to as the "reduction" step) is to reduce the N rows of partial products to 2 rows that have an equivalent value. This step has the most delay in a multiplier and is where most of the research effort, this report included, focuses on improving. The third step is to use a carry propagating adder (CPA) to add the 2 rows and obtain their sum which is the product of the two input operands.

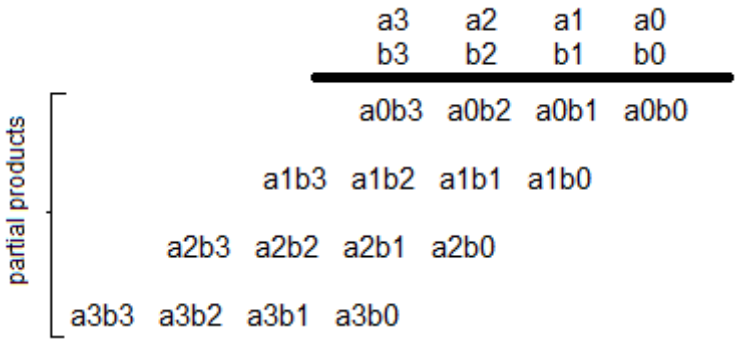


Fig 1. Partial Product Matrix for a 4-Bit by 4-Bit Multiplication

There are two widely used approaches, Wallace[1] and Dadda[2], which are currently used in high speed multipliers to perform the reduction step. These multipliers are constructed with half adders and full adders. By using these arithmetic components in parallel, a result can be obtained

quickly. Moreover, by using carry lookahead adders to replace some of the full adders, the delay of the second step (the reduction) can be improved by up to 30% [3]. The drawback is that the complexity of the design increases by up to 25%.

At this time, there are two approaches one can take to implement the carry lookahead adders in the multiplier: aggressively reducing partial products, similar to a Wallace multiplier, and conservatively reducing partial products, similar to a Dadda multiplier. The two approaches offer different benefits with regards to complexity and performance. This report analyzes the differences between the two multipliers with CLAs.

2.0 Background

Most modern high speed multipliers are either Wallace or Dadda multipliers. Both of these approaches use full adders and half adders in the reduction step to achieve a respectable performance. The difference in these two multipliers lies in how each multiplier utilizes the full adders and half adders. To understand this, a background of the arithmetic components and dot diagrams follows.

2.1 Arithmetic Components Used in High Speed Multipliers

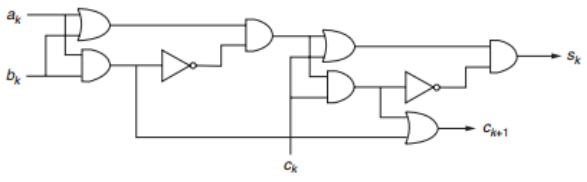


Fig. 2a Full Adder[4]

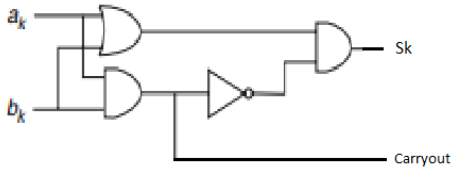


Fig. 2b Half Adder[4]

Full and half adders are the arithmetic components used in the Wallace and Dadda multipliers. The full adders and half adders used in this report are depicted in the Fig 2a and Fig 2b. The full adder has a complexity of 9 gates and takes in 3 inputs to produce a sum and a carry out. Throughout this report the latency is measured in terms of gate delays. That is, each gate is assumed to have roughly the same delay. This holds true for any gate with a fan-in of less than or equal to 4. For example, the full adder has a latency of 6 gate delays. This can be seen by tracing the inputs, a_k or b_k, to the s_k output. In contrast, the half adder has a complexity of 4 gates. Half adders take in 2 inputs to produce a sum and a carry-out. The half adder has a latency of 3 gate delays. Similarly, the delay can be found by tracing the inputs to the s_k output.

2.2 Dot Diagrams

The sheer number of partial products that arises through multiplication calls for an efficient way to visualize how the multiplier is implemented. Through the use of dot diagrams, in which partial products are represented by dots, one can see how Wallace and Dadda multipliers are implemented. The Dadda dot notation [2] is used in this report. The following notations represent the partial product term, half adder and full adder:

- A partial product term (AND gate output)

 The outputs of a half adder

 The outputs of a full adder

The dot diagram is divided into “stages”. Each stage contains a number of additions that happen in parallel. The delay of each stage is equal to the maximum delay of the arithmetic components

in that stage. In the case of a Wallace or Dadda multiplier, this is equal to the delay of a full adder. A horizontal line is used to delineate each stage in the Wallace and Dadda multiplier in the dot diagram.

2.3 Wallace Approach

Wallace multipliers implement an aggressive approach to the reduction process where the maximum possible reduction is done at every stage. A dot diagram for the first two steps (array formation and reduction) of a 12-bit by 12-bit Wallace multiplier is shown in Figure 3. Wallace seeks to reduce any 3 rows of partial products into 2 rows [1]. For each column in the partial product matrix, groups of 3 dots form the inputs for a full adder. Meanwhile, groups of 2 dots form the inputs for a half adder. It is common in Wallace Multipliers to have groups of 2 dots surrounding a range of group of 3 dots; because of this, half adders usually appear on the outer edges of the dot diagrams.

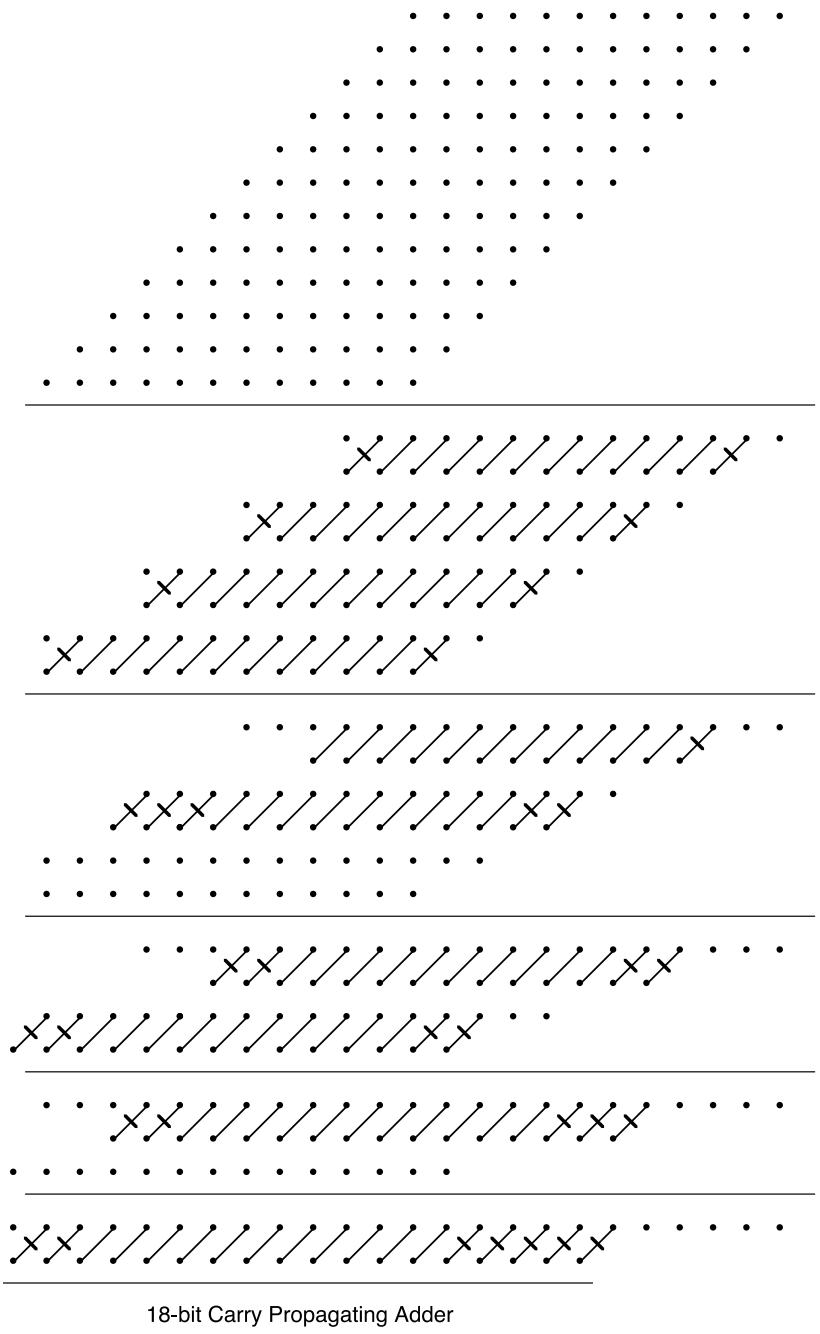


Fig 3. Wallace 12-Bit by 12-Bit Reduction

The number of arithmetic components was then recorded into Table I to analyze the complexity of the Wallace 12-bit by 12-bit. There is a total of 102 full adders and 34 half adders in the entire

dot diagram which gives the Wallace 12-bit by 12-bit a total complexity of 1054 gates. The Wallace 12x12 multiplier has a total of 5 reduction stages. Each reduction stage has 6 gate delays which gives the 12-bit by 12-bit Wallace multiplier a total delay of 30 gate delays.

Table I. Arithmetic Component Count in a 12-Bit by 12-Bit Wallace Reduction

Stage	**1**	**2**	**3**	**4**	**5**	**Total**
Full Adders	40	20	20	11	11	102
Half Adders	8	6	8	5	7	34

As seen in Table I, most of the reductions happen towards the initial stages which present a benefit when using a pipelined ALU. Thus, if the outputs from each stage are latched into registers, the Wallace approach will use fewer latches.

2.4 Dadda Approach

The Dadda approach differs from the Wallace approach using the minimum number of full and half adders necessary to meet predetermined heights for each stage. The stage height limits are: {#2,#3,#4,#6,#9,#13,#19,#28,#42, etc.} [4]. Each stage is 1.5 times larger than the subsequent stage.

The 1.5 reduction ratio results from the use of full adders which take 3 partial products and reduces them to 2 partial products. Thus the maximum height of the succeeding stage is ⅔ that of the previous stage. In doing this, Dadda seeks to optimize the area of the multiplier by using the fewest number of adders to reach the CPA stage.

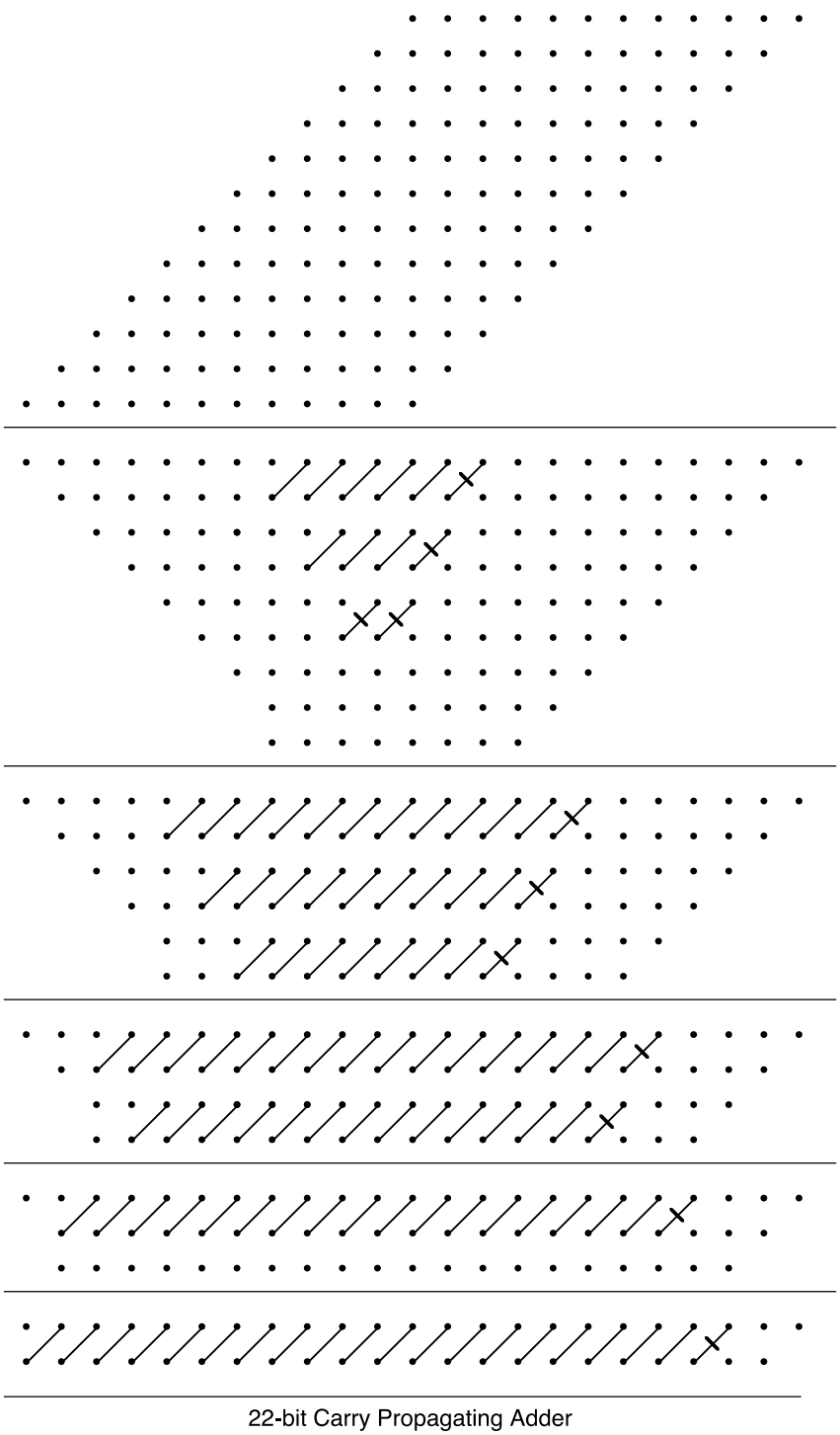


Fig 4. Dadda 12-Bit by 12-Bit Reduction

A dot diagram is shown in Figure 4 detailing the Dadda reduction for a 12-bit by 12-bit multiplier. Because the closest Dadda number to 12 is 9, the first reduction stage has a height of 9 dots. The next stages proceed to follow the Dadda numbers, 9 to 6, 6 to 4, etc. The Dadda multiplier has the same number of reduction stages as a Wallace multiplier; in this case, the 12-bit by 12-bit Dadda multiplier has 5 reduction stages. Thus, the delay of the Dadda approach for the 12 bit word size is 30 gate delays which is the same delay as the Wallace approach.

Table II shows the number of arithmetic components in each of the 5 stages of the 12-bit by 12-bit Dadda reduction. It can be seen that the number of arithmetic components is greatest in the middle stages of its reduction. There is a total of 99 full adders and 11 half adders which gives a total complexity of 945 gates. Because Dadda reduction follows a set of stage heights, the number of full adders in an N-bit by N-bit Dadda multiplier can be found with: N^2-4N+3; the number of half adders can be found with: N-1.

Table II. Arithmetic Component Count in a 12-Bit by 12-Bit Dadda Reduction

Stage	**1**	**2**	**3**	**4**	**5**	**Total**
Full Adders	8	27	28	17	19	99
Half Adders	4	3	2	1	1	11

The complexity of the reduction step in a Wallace multiplier is approximately 10% larger than that of Dadda for a given input size. Despite this reduction in hardware, Dadda multipliers contain the same number of reduction stages compared with Wallace multipliers. As a result, they are expected to have the same delay in the reduction step. However, the overall delay of a Wallace multiplier might differ from the Dadda multiplier as the size of the CPA is smaller in the

Wallace multiplier. The CPA observes increases in delays every time its input word size increases by a factor of 4.

3.0 Multipliers Implemented with Carry Lookahead Adders

The original Wallace and Dadda multipliers, described in the previous sections, are able to achieve respectable performance with the sole use of full adders and half adders. The performance can be improved by using Carry Lookahead Adders (CLAs) in the reduction process. Implementations for both the Wallace and Dadda were done with CLAs in the following subsections along with a background on CLAs.

3.1 Arithmetic Components Used in Multipliers Implemented with Carry Lookahead Adders

CLAs take in up to 9 partial product bits and reduce them to 5 partial products. Using this concept, 4 bit CLAs are used in parallel in a reduction stage and observe the same delay as a reduction stage using full adders and half adders. Figure 5 shows a 4 bit CLA, which is constructed with 1 Full Adder (FA), 3 Modified Full Adders (MFAs), and a lookahead logic block.

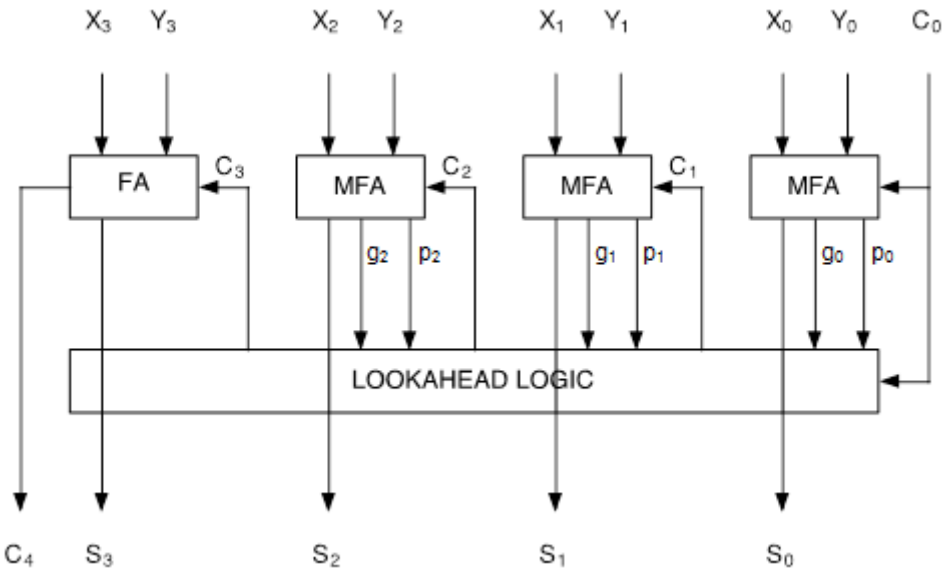


Fig 5. 4-Bit Carry Lookahead Adder

CLAs calculate the sum 2 inputs quickly by calculating the carries in parallel. C_1,C_2, and C_3 are calculated simultaneously with the equations shown below. To perform this operation, the CLA first uses MFAs to output the generate bit, g_i, and the propagate bit, p_i. These 3 pairs of generate and propagate bits are then used to form the 3 carries used internally in the CLA. The last carry, C_3, is used as the carry input for the full adder to calculate the sum bit, S_3, and the carry bit, C_4.

$$g_i = x_i y_i \tag{1}$$

$$p_i = x_i + y_i \tag{2}$$

$$c_{i+1} = g_i + p_i c_i \tag{3}$$

$$c_{i+2} = g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i \tag{4}$$

$$c_{i+3} = g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i \tag{5}$$

The critical path through the CLA can be found by starting at any of the inputs to the MFAs and using the final sum bit, S_4, as the sink. The propagate and generate bits are both available after 1 gate delay. The carries are generated after 2 gate delays since they involve an AND operation as well as an OR operation. It takes 3 gate delays to travel from the carry input to the sum output in a full adder. This gives a total of 6 gate delays which is the same as a 9 gate full adder.

Modified full adders are used to reduce the complexity of the overall CLA seen in Figure 5. This is because carry-outs are not needed from the first three full adders in the CLA. A modification was done to the full adder by removing the OR gate that outputted the carry-out. By doing this, each of the 3 modified full adders uses 8 gates which is 1 less than a normal full adder. The modified full adder is shown in Figure 6. From Equations (2)-(5), the lookahead logic block can be constructed with 9 gates. This gives the CLA a complexity of 42 gates.

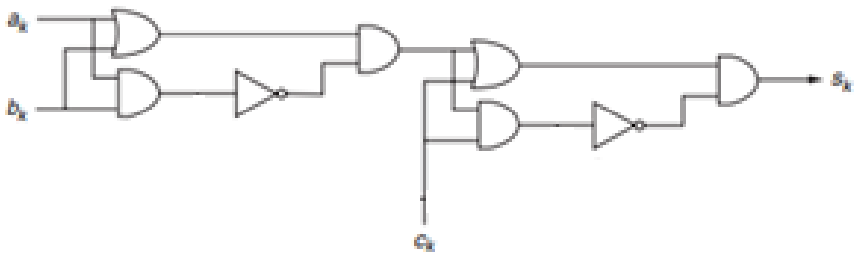


Fig. 6 Modified Full Adder

One other possible design for a CLA block is shown in Figure 7. It was found throughout the course of using CLAs in multipliers that 9 partial products are not always available to use as

inputs. The purpose of this design is to accommodate instances where the 9 dot inputs are not available. This design uses 8 inputs by omitting the carry-in used in the previous CLA design.

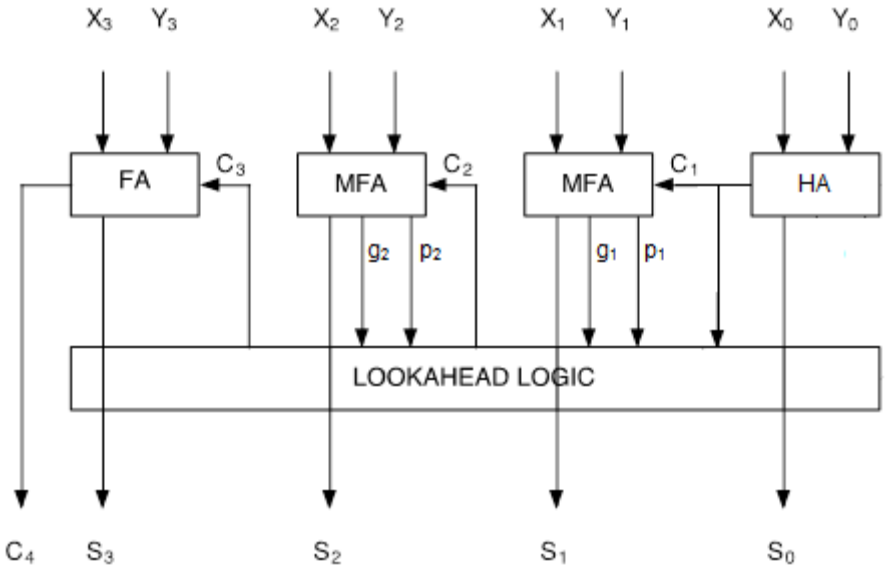


Fig 7. CLA Without Carry-in

The result of using the CLA without carry-in is lower complexity since C1 is formed in the half adder in one gate delay compared to the original CLA with carry-in in which C1 is formed in the lookahead logic block. Equations (4) and (5) can be rewritten as (6) and (7) as a result. The critical path has the same latency at 6 gate delays. However, the complexity is reduced to 34 gates.

$$c_{i+2} = g_{i+1} + p_{i+1}c_{i+1} \quad (6)$$

$$c_{i+3} = g_{i+2} + p_{i+2}g_{i+1} + p_{i+2}p_{i+1}c_{i+1} \quad (7)$$

The reason behind why CLAs are able to improve the performance in a multiplier lies in its better reduction ratio. In the case of the CLA with carry-in, the CLA is able to input 9 dots and output 5 dots giving this CLA a reduction ratio of 1.8. Meanwhile, the CLA without carry-in is able to input 8 dots and output 5 dots giving this CLA a reduction ratio of 1.6. The reduction ratio for a full adder is 1.5. Thus, CLAs are able to reduce more dots in the same time step.

CLAs are represented in the dot diagrams as shown in Figure 8. It should be noted that the line does not necessarily need to be horizontal as long as it connects 5 dots in different columns. The first 4 dots are the sum outputs from the CLA and the last dot is the carry-out from the CLA. A slash is added to indicate that no carry-in is used for the CLA.

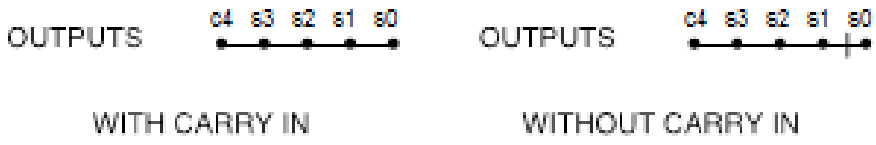


Fig 8. Depiction of 4-Bit CLA in Dot Diagram

3.2 Wallace CLA Approach

This section introduces the implementation of the Wallace CLA multiplier. The dot diagram, shown in Figure 9, is similar to a regular Wallace dot diagram except that the most used arithmetic component is the carry lookahead adder. Due to the fact that the CLA can take in a different number of inputs at its LSB compared to the other 3 pairs of inputs, this gives the dot diagram an unorganized look where the number of dots in each column does not monotonically

increase before monotonically decreasing as the behavior in the normal Wallace and Dadda reduction. Throughout this report, light lines are used to group columns of 4 together to better organize the look of the dot diagrams with CLAs.

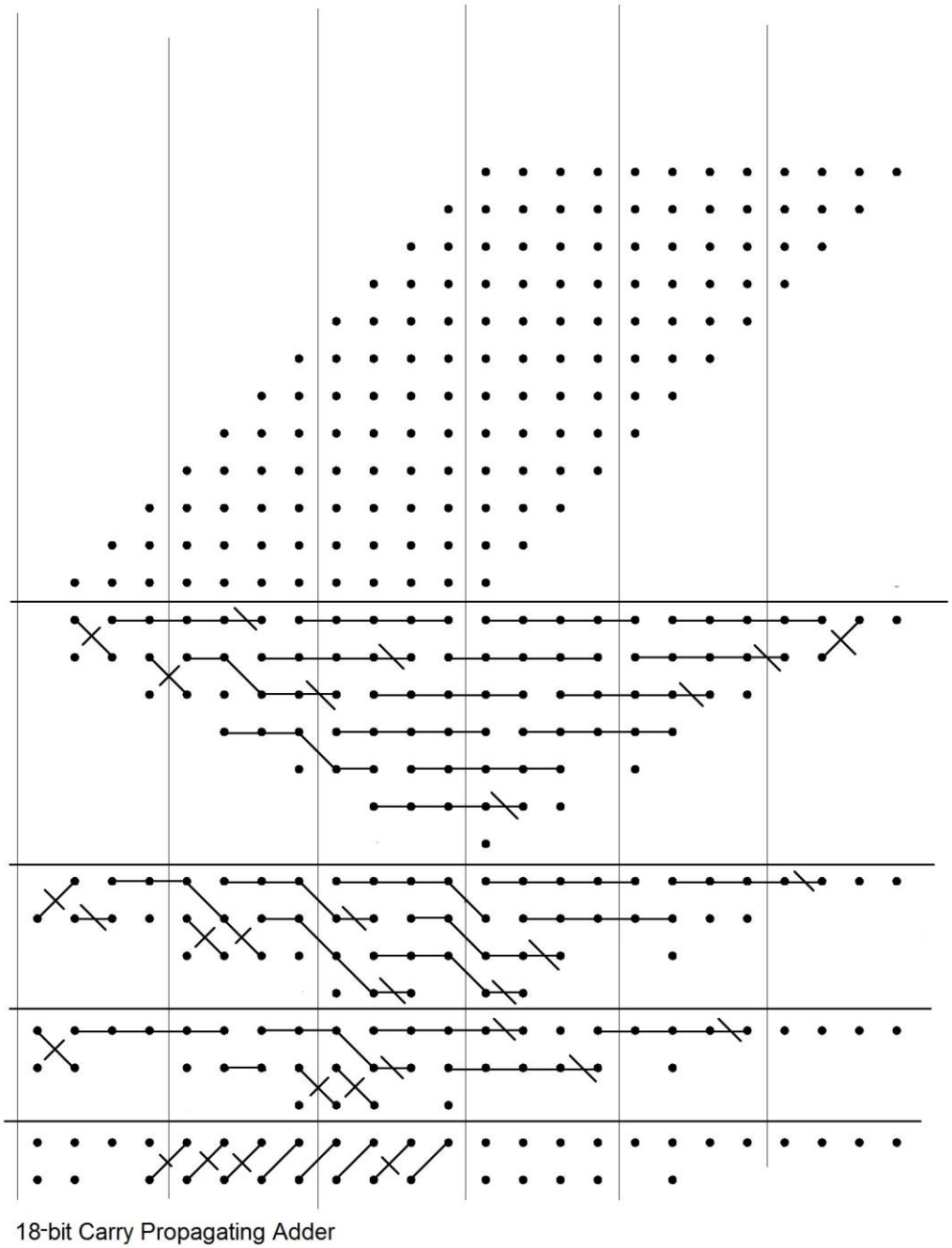


Fig 9. Wallace CLA 12-Bit by 12-Bit Reduction

An analysis of the Figure 9 is shown with Table III. The number of full adders is greatly reduced compared to the original Wallace multiplier. All but one full adder are used in the final reduction stage. A total of 1187 gates are used in the 12-bit by 12-bit Wallace CLA reduction. There are 4 reduction stages giving this multiplier a latency of 24 gate delays.

Table III. Arithmetic Component Count in a 12-Bit by 12-Bit Wallace CLA Reduction

Stage	**1**	**2**	**3**	**4**	**Total**
CLA w/carry	9	3	1	0	13
CLA w/o carry	6	6	4	0	16
Full Adders	0	0	1	4	5
Half Adders	3	3	3	4	13

Similar to the standard Wallace multiplier, CLAs, FAs, and HAs are aggressively used in the implementation of the Wallace CLA multiplier. CLAs were placed where a sufficient number of inputs for the respective CLA were present. The number of carry lookahead adders in each subsequent stage is fewer than the previous stage. This is because the Wallace multiplier computes most of the intermediate partial products in the initial reduction stages. Some optimizations were done in this 12-bit by 12-bit Wallace CLA multiplier. To make efficient use of complexity, CLAs were not used in the last reduction stage, in going from a stage of height 3 to a stage of height 2.

In the 12-bit by 12-bit Wallace CLA diagrams, each stage height was reduced to a subsequent stage height of $\left\lceil \frac{N}{1.8} \right\rceil$. This is a greater reduction than that of the conventional Wallace multiplier

in which each subsequent stage height is $\left\lceil \frac{N}{1.5} \right\rceil$. It was found that 1.8 was the maximum reduction possible as discussed in the section below.

3.2.1 Determining the Maximum Reduction

To find the maximum reduction of a given stage height N, a repeatable pattern of CLAs is found that can be arranged to apply the reduction. A worst case scenario for a given input size is assumed so that the pattern will hold for all cases of the same input size. Since a CLA spans 5 columns of dots then a partial product matrix with 5 columns is sufficient to generalize the case.

The first step is to consider the first column, column *i*, in the partial product matrix. Begin by starting 1 CLA on this column. Then for each pair of the remaining dots in column *i* assume that there is a preexisting CLA. Increase the number of starting CLAs in column *i* to a value *k* until the number of preexisting CLAs that can fit in column *i* is 3*k* or 3*k-1*. Finally, to get this pattern to repeat, for each CLA that was started in column *i* there exists a corresponding number of carry outs in column *i.* Repeat this process for columns *i+1* to column *i+3* keeping in mind that the number of preexisting CLAs in column *i* must eventually produce carry outs distributed over these columns. The fifth column in the partial product matrix, *i+4*, has to mirror the first column, column *i.*

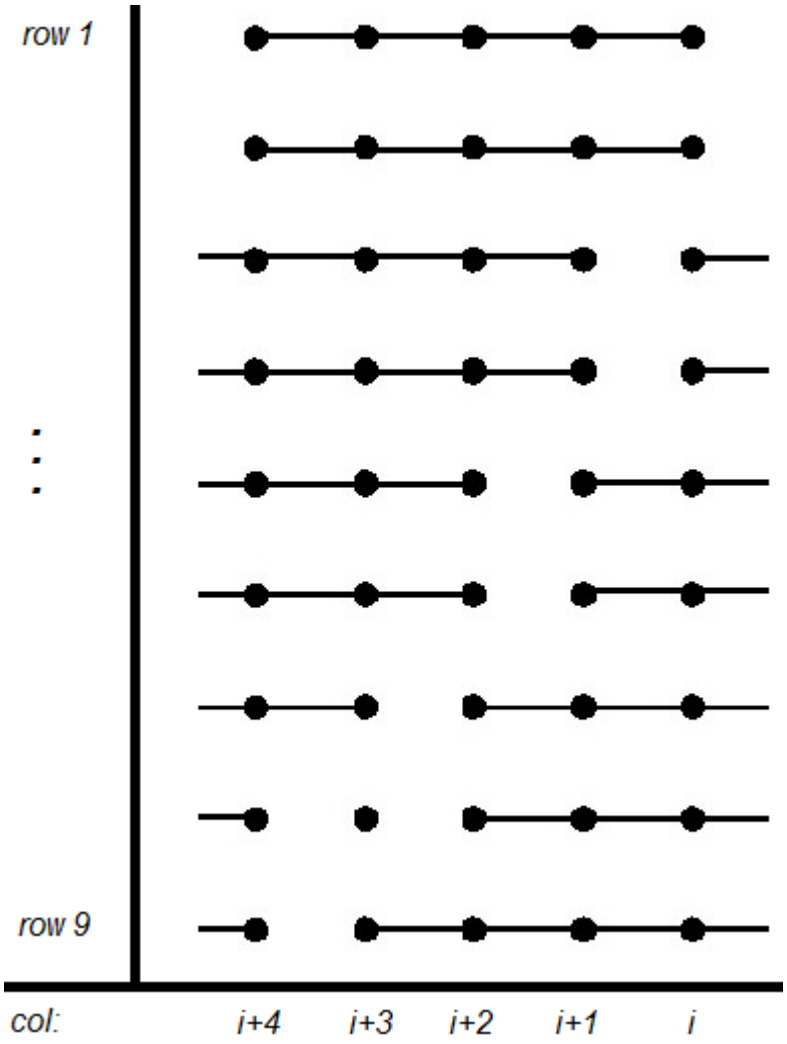


Fig 10. Example of a Worst Case Reduction of a 16-Bit Stage Height

This is best illustrated with the example shown in Figure 10, which shows the maximum reduction from a stage height of 16 dots. Following the steps described before, 2 CLAs are started at column *i* which gives *k* a value of 2. There are also 7 preexisting CLAs in column *i* which satisfies the 3*k*-1 requirement. These preexisting CLAs all have their final carry-out bit outputted before column *i*+4. Finally, column *i*+4 has the same number of preexisting CLAs and the same number of starting CLAs as column *i*. This means that this pattern can be repeated.

There is a single dot in column *i+3* that is neither an input to a CLA or an output to a CLA. The simplest explanation is that the total number of dots, 80 dots, does not divide evenly into the 1.8 reduction ratio. An extra dot can be expected as a result in each of the 5 column groups as a result.

Through experimentation, the stage height reduction ratio was found to be $\left\lceil \frac{N}{1.8} \right\rceil$. It should be noted that this reduction is for the worst case scenario. An example of a case where the stage height might have a greater reduction is where there is only one column of max height in the reduction stage. This is occasionally seen in the Wallace CLA multiplier and explains why the reduction might exceed the 1.8 reduction.

3.3 Dadda CLA Approach

The Dadda CLA multiplier differs from the Wallace CLA multiplier by doing the minimum reduction required at each stage. Predetermined stage heights are used similar to how the original Dadda reduction uses predetermined stage heights. To find these predetermined stage heights, each stage can be no higher than 1.8 times the subsequent stage height. Since all multipliers must reduce eventually to two, the Dadda CLA stage heights were calculated to be: 2, 3, 5, 9, 16, 28, 50, etc.

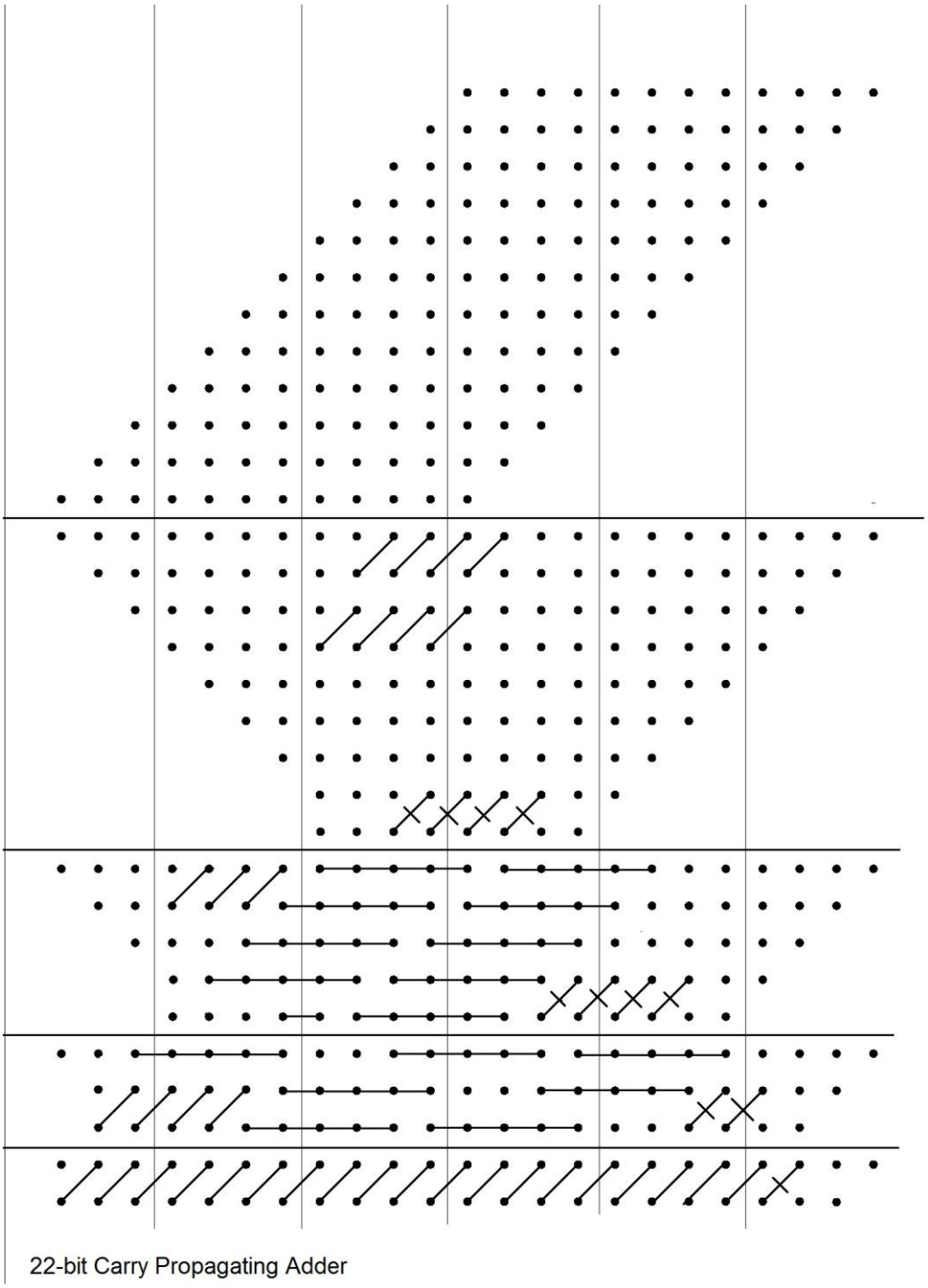


Fig 11. Dadda CLA 12-Bit by 12-Bit Reduction

A dot diagram for a Dadda CLA 12-bit by 12-bit multiplier is shown in Figure 11. Like the 12-bit by 12-bit Wallace CLA reduction, the Dadda CLA reduction achieves the same performance by completing the reduction in 4 stages. Table IV summarizes the number of arithmetic components in each stage for the Dadda CLA reduction. The total complexity of this 12-bit by 12-bit Dadda CLA reduction is 1031 gates which is approximately 5% lower than its Wallace CLA counterpart.

Table IV. Arithmetic Component Count in a 12-Bit by 12-Bit Dadda CLA Reduction

Stage	**1**	**2**	**3**	**4**	**Total**
CLA w/carry	0	9	7	0	16
CLA w/o carry	0	0	0	0	0
Full Adders	8	4	4	19	35
Half Adders	4	4	2	1	11

Since the Dadda CLA multiplier is not always seeking the maximum reduction of each column, it requires fewer CLAs than the Wallace CLA multiplier. Table IV shows how the Dadda CLA multiplier uses a greater number of full and half adders. For instance, in any stage where the reduction is less than 1.5 times the preceding stage height, the Dadda CLA approach can take advantage of using full and half adders for the reduction; this usually occurs, if at all, on the first and last reduction stage.

The Dadda CLA reduction comes with a similar drawback to that of the original Dadda multiplier. Compared to the Wallace CLA multiplier, the Dadda CLA approach does most of its reduction in the middle of its reduction stages; this makes it less attractive for pipelined multiplication. In addition, the Dadda CLA multiplier also has a slightly larger CPA width than the Wallace CLA multiplier.

4.0 Results

This section analyzes how the two CLA approaches, Wallace and Dadda, differ from the original Wallace and Dadda multipliers. A comparison for the delay and complexity is presented. The two CLA approaches are also compared to each other.

4.1 Analysis of Delay

Both Wallace and Dadda CLA multipliers have the same number of reduction stages and, thus, the same number of unit gate delays. Using CLAs in the multipliers results in fewer reduction stages and, as a result, a lower delay compared to the regular Wallace and Dadda multipliers. Over the range of sizes presented in Table V, there is a maximum difference of 2 reduction stages between the Wallace CLA/Dadda CLA multipliers and the original Wallace/Dadda multipliers. For the 24 bit word size, this translates to a saving in delay of 33% compared to the original multipliers.

Table V. Gate Delays for Different Multiplier Word Sizes

	Gate Delays to CPA Stage			
Word Size	**12**	**16**	**24**	**32**
Wallace CLA/Dadda CLA	24	24	30	36
Wallace/Dadda	30	36	42	48

Using the Dadda numbers, the delays for the CLA multiplier were extrapolated for all word sizes in Figure 12. From Figure 12, the Wallace CLA/Dadda CLA always have fewer reduction stages than the original Wallace/Dadda multipliers for any word size greater than 10 bits. The greatest difference in the number of reduction stages is 2 in the range for which the data is presented.

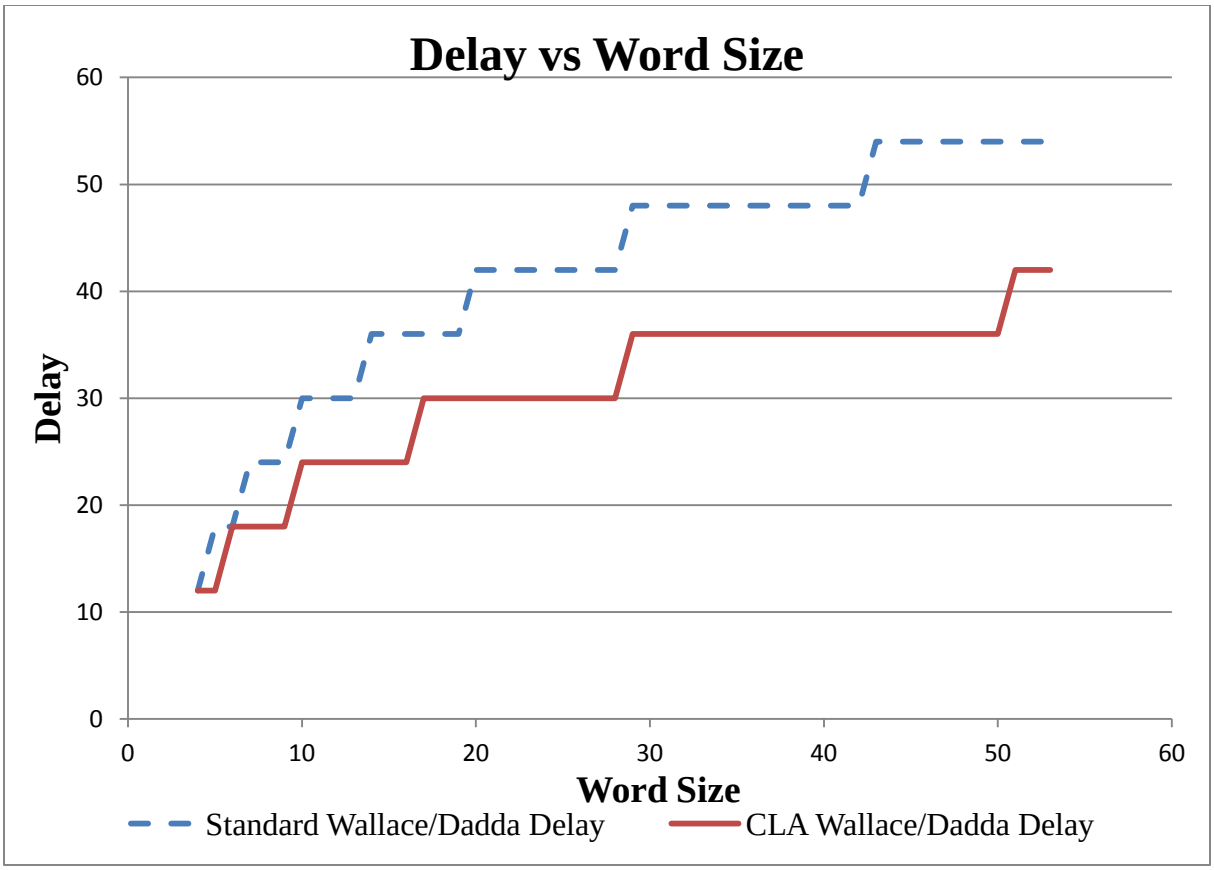


Fig. 12. Delay vs Multiplier Word Size

4.2 Analysis of Complexity

The tradeoff for the use of CLAs is a higher complexity. Each CLA with carry-in is built with 42 gates and each CLA without carry-in is built with 34 gates. To do the equivalent amount of work as a CLA with carry-in, (to remove 4 dots from a dot diagram) 4 full adders are required which is built with 36 gates. Analyzing the complexity difference at this level translates to a 17% increase in complexity for the CLA with carry-in and a 6% decrease in complexity for the CLA without carry-in. The increase in the complexity of the overall multiplier can be expected to be roughly less than the 17% increase. Table VI shows the gate count for Wallace, Wallace CLA, Dadda, and Dadda CLA reduction. The increase in complexity ranges from 9-13% for a Wallace CLA

multiplier compared to a regular Wallace multiplier. In contrast, the Dadda CLA multiplier has a 10-15% increase in complexity over a regular Dadda multiplier.

Table VI. Gate Count for Different Multiplier Word Sizes

	Complexity			
Word Size	**12**	**16**	**24**	**32**
Wallace	1054	2008	4801	8778
Wallace CLA	1187	2194	5353	9977
Dadda	935	1815	4439	8215
Dadda CLA	1031	2091	4955	9277

4.3 Analysis of Wallace CLA vs Dadda CLA

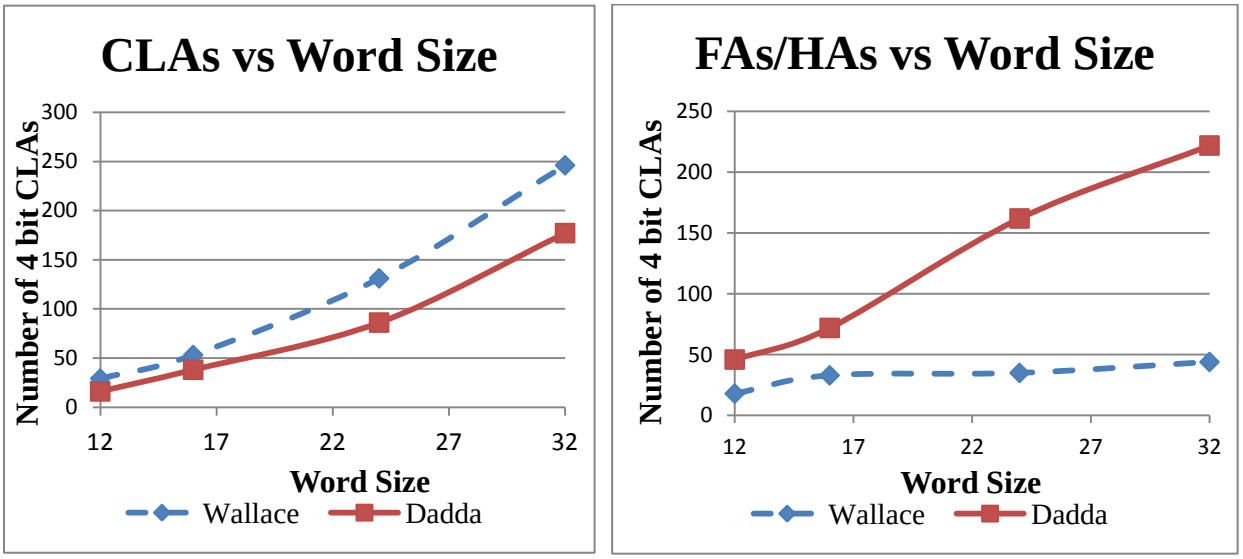


Fig. 13. Comparison of Arithmetic Components Used in Wallace CLA and Dadda CLA Multipliers

The number of CLAs grows almost exponentially with word size. Because of this exponential relationship, the Wallace CLA reduction has nearly 40% more CLAs than the Dadda CLA reduction for the 32 bit word size. In contrast, the relationship for the full adders and half adders

is linear. Since the Dadda CLA multiplier uses a lot more full adders and half adders, this will cause a difference in complexity between Wallace and Dadda CLA multiplier to exponentially increase as the word size increases.

Table VII. Carry Propagating Adder Width for Different Multiplier Word Sizes

Stage	**12**	**16**	**24**	**32**
Wallace	18	25	40	55
Wallace CLA	18	26	41	56
Dadda	22	30	46	62
Dadda CLA	22	31	46	62

As mentioned before, although the number of reduction stages is the same in both Wallace and Dadda, the size of the carry propagating adder is not always the same. This effect is seen in Table VII which shows the sizes of the carry propagating adder for Wallace and Dadda multipliers with CLAs and without CLAs. The size of the CPA in the Dadda multiplier is always larger than the size of the CPA in the Wallace multiplier. Despite this increase in size, the delay of the CPA is still the same for both the Wallace and the Dadda at each of the sizes shown in the table.

5.0 Conclusion

New designs of Wallace and Dadda reduction were presented in this report. Through the use of dot diagrams, the implementations of the designs were done. From these dot diagrams, the complexity and latency were analyzed. It was seen that by using CLAs the latency can be lowered from the original Wallace and Dadda reduction. As with most engineering problems, this comes with a tradeoff. In this case, the tradeoff is a higher complexity. The complexity increases by 10-15% for the Dadda CLA compared to the standard Dadda reduction. In contrast,

the complexity increases by 9-13% for the Wallace CLA compared to the standard Wallace reduction. Both the Wallace and Dadda CLA reductions observed the same amount of latency for each given input size. Using CLAs reduces the number of reduction stages required by up to two which can be as much as a 33% saving in delay.

Even though the Dadda CLA reduction has a lower complexity and the same latency, some applications might be suitable for the Wallace CLA reduction. For instance, pipeline multipliers can take advantage of the fact that the majority of the reduction happens early in the reduction steps. Thus, fewer latches are needed for a Wallace multiplier compared to a Dadda multiplier.

Appendix

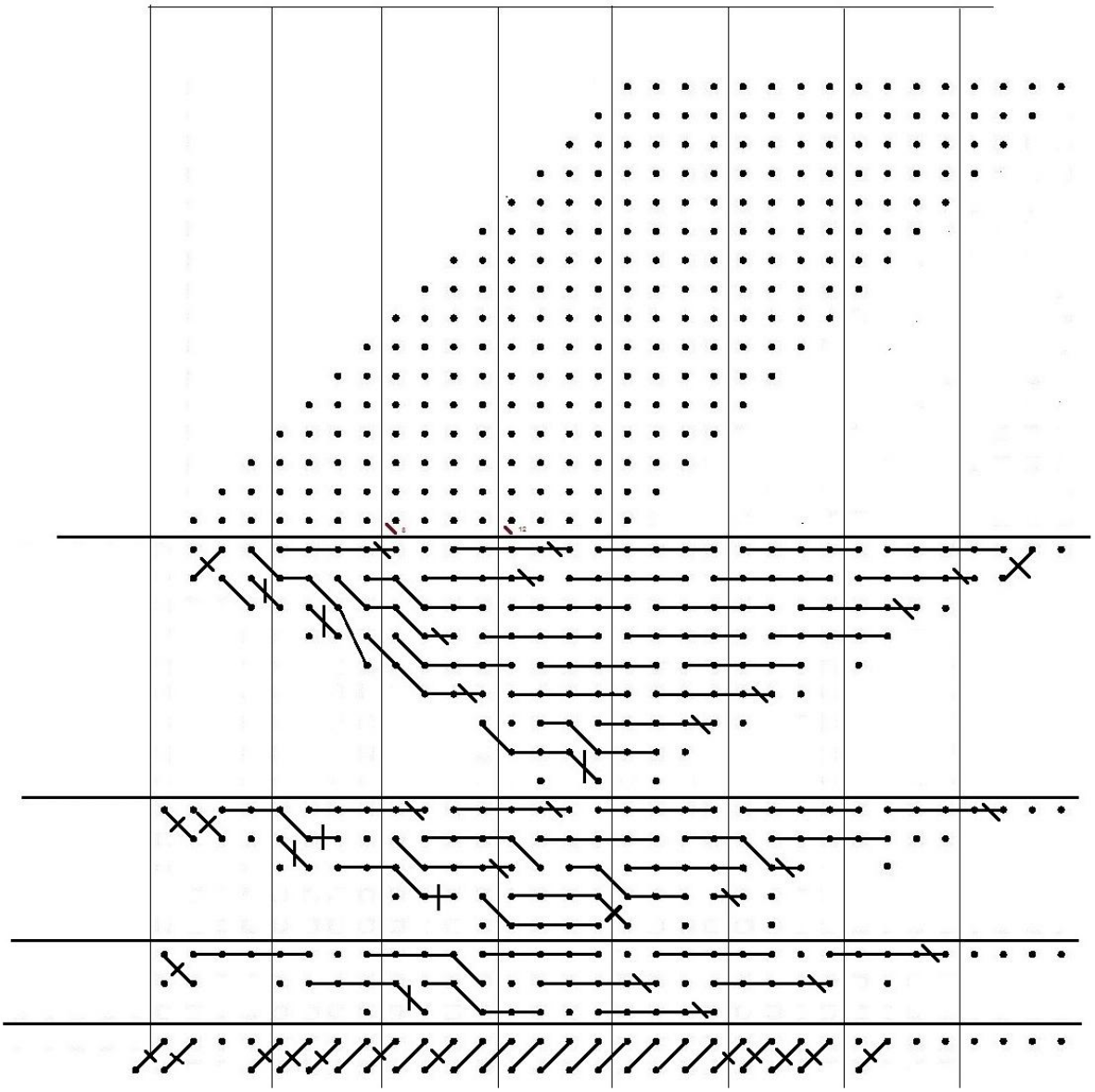
Wallace Multiplier Complexities

Size	9 bit CLAs	8 bit CLAs	Full Adders	Half Adders	Gate Count	CPA Width
12x12	13	16	5	13	1187	18
16x16	30	23	12	21	2194	26
24x24	88	43	11	24	5353	41
32x32	174	72	9	35	9977	56
53x53	540	138	49	95	28193	99

Dadda Multiplier Complexities

Size	9 bit CLAs	8 bit CLAs	Full Adders	Half Adders	Gate Count	CPA Width
12x12	16	0	35	11	1031	22
16x16	37	1	43	29	2091	31
24x24	86	0	139	23	4955	46
32x32	177	0	191	31	9277	62

Wallace CLA 16-Bit by 16-Bit Reduction

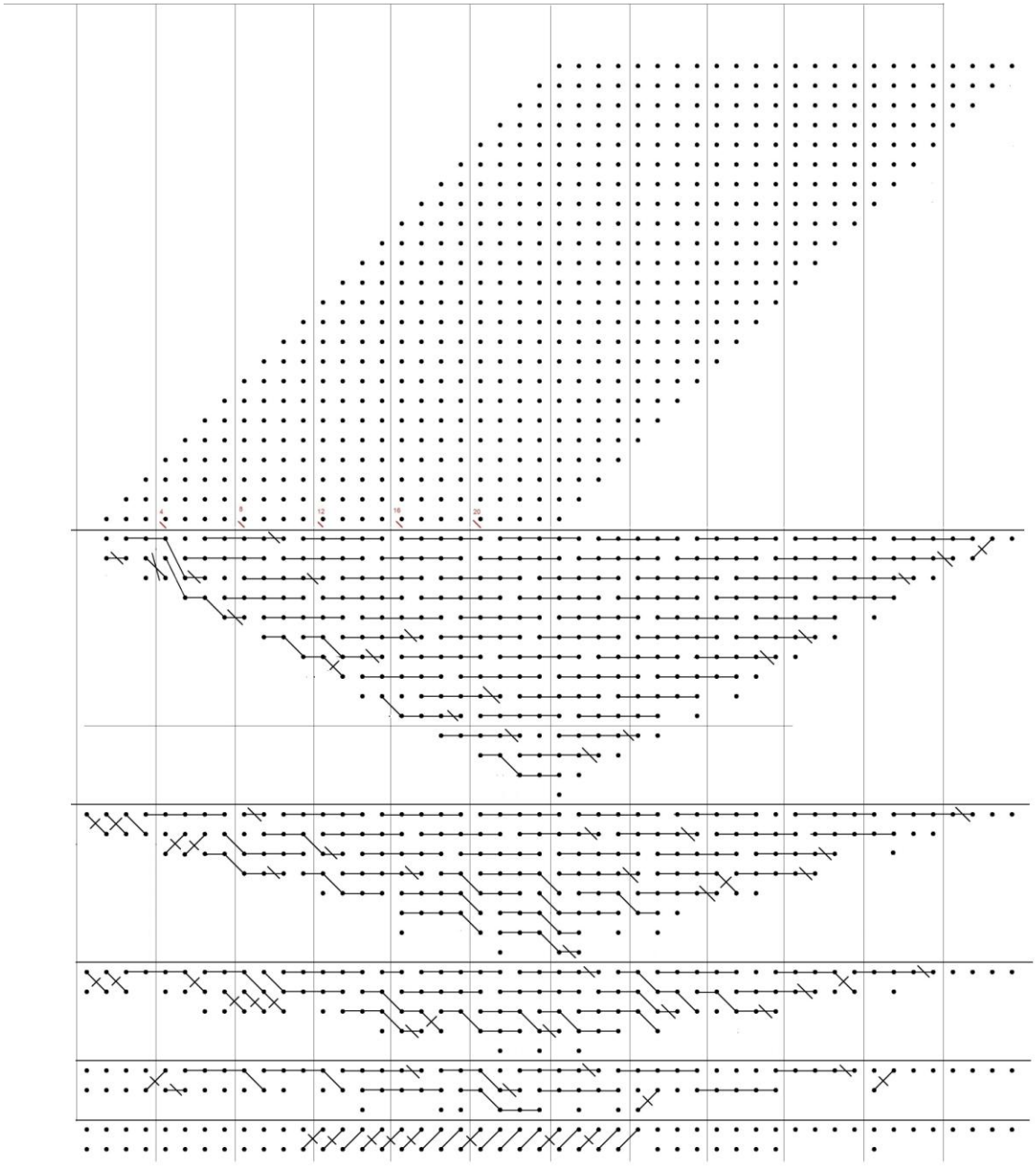


26-bit Carry Propagating Adder

Total:
30 x 9 bit CLAs
23 x 8 bit CLAs
12 FAs
11 HAs

Gate Count: 2194

Wallace CLA 24-Bit by 24-Bit Reduction

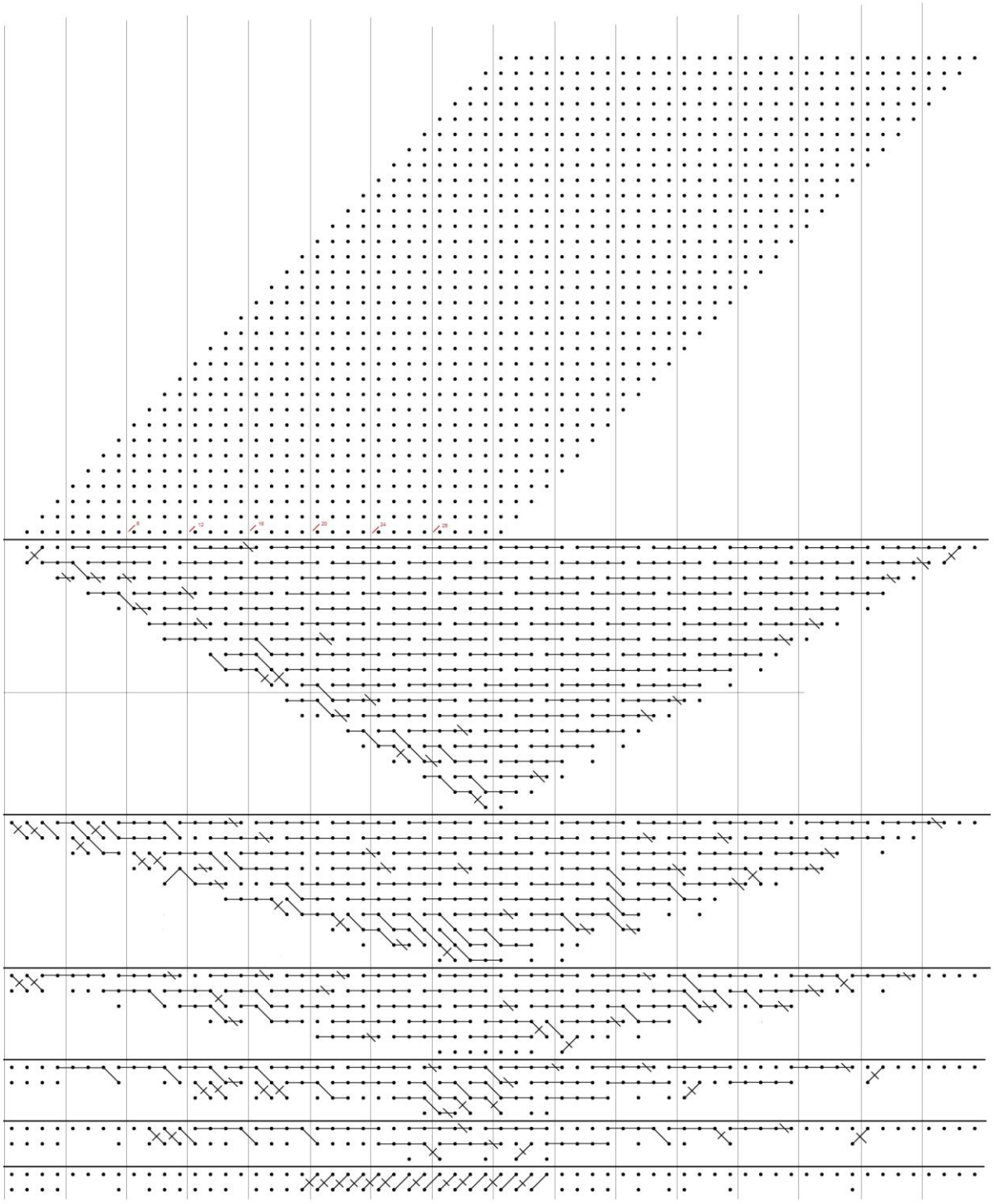


88x9 bit CLAs
43x8 bit CLAs
11 FAs
24 HAs

Total Gate Count: 5353

41-bit Carry Propagating Adder

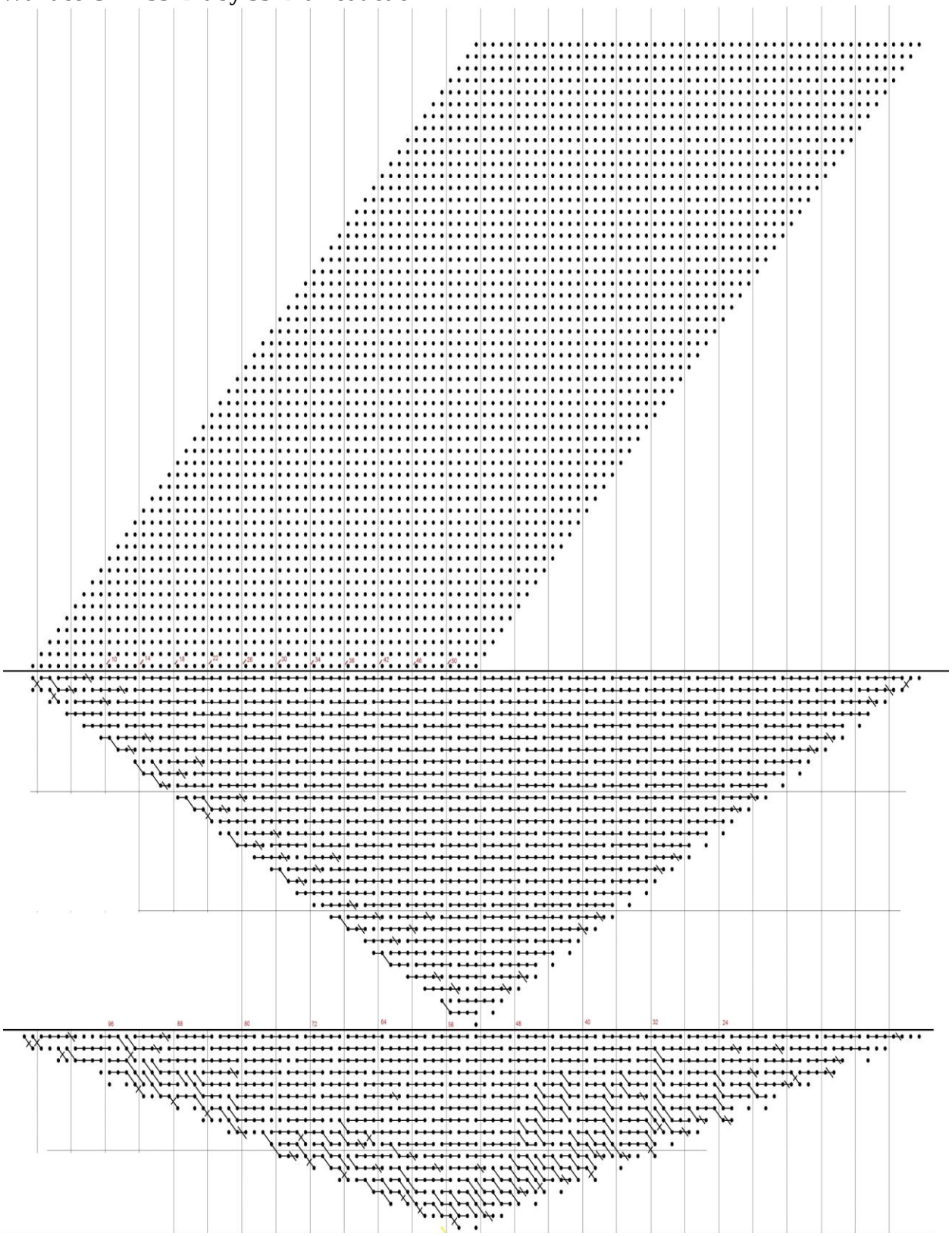
Wallace CLA 32-Bit by 32-Bit Reduction



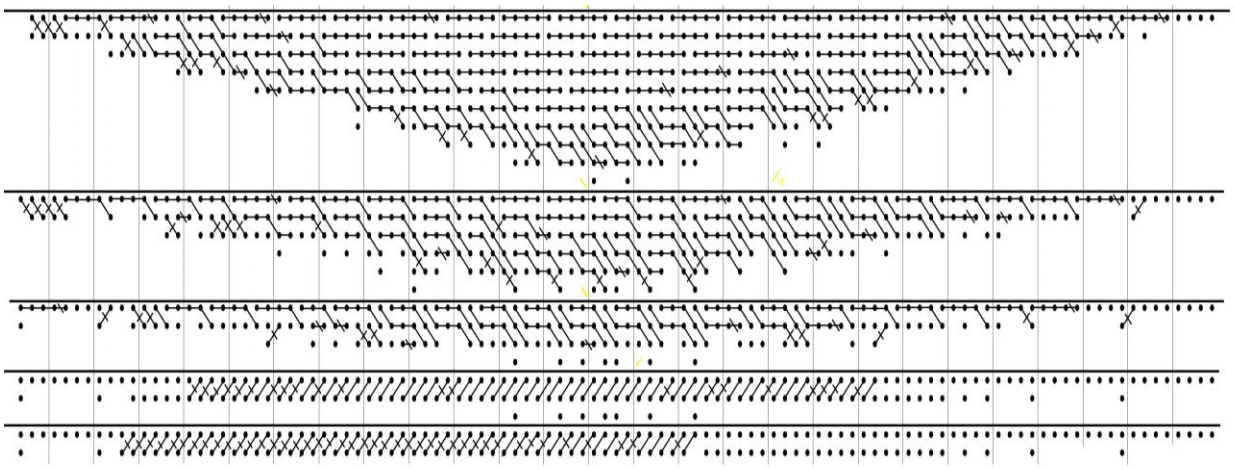
174 x 9 bit CLAs
72 x 8 bit CLAs
9 FAs
35 HAs
Total gate count: 9977

56-bit Carry Propagating Adder

Wallace CLA 53-Bit by 53-Bit Reduction



Wallace CLA 53-Bit by 53-Bit Reduction (cont.)

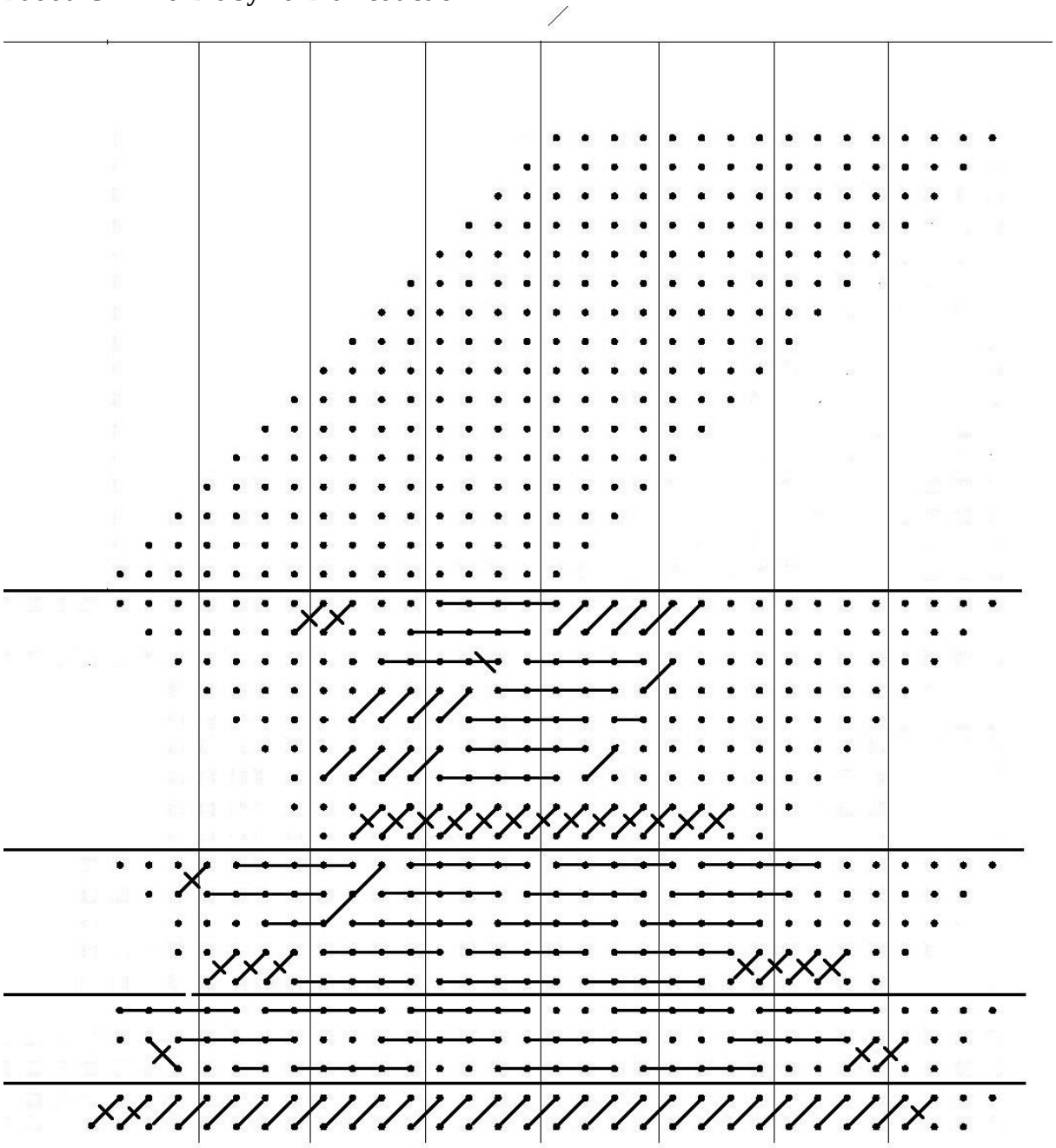


540 x 9 bit CLAs
138 x 8 bit CLAs
49 FAs
95 HAs

Total gate count: 28193

99-bit Carry Propagating Adder

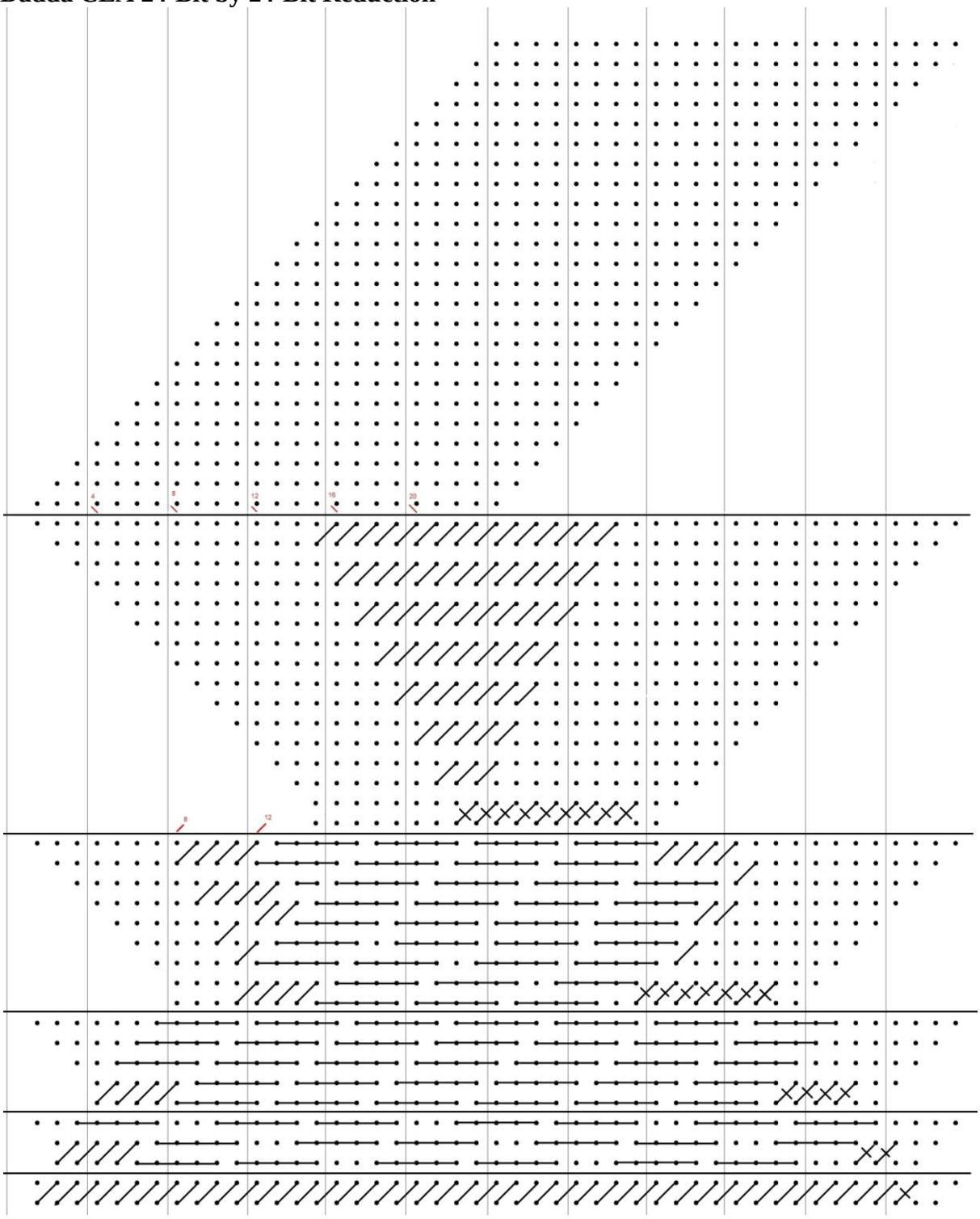
Dadda CLA 16-Bit by 16-Bit Reduction



37 x 9 bit input CLA
1 x 8 bit input CLA
43 FAs
29 HAs
2091 Gate Count

31-bit Carry Propagating Adder

Dadda CLA 24-Bit by 24-Bit Reduction

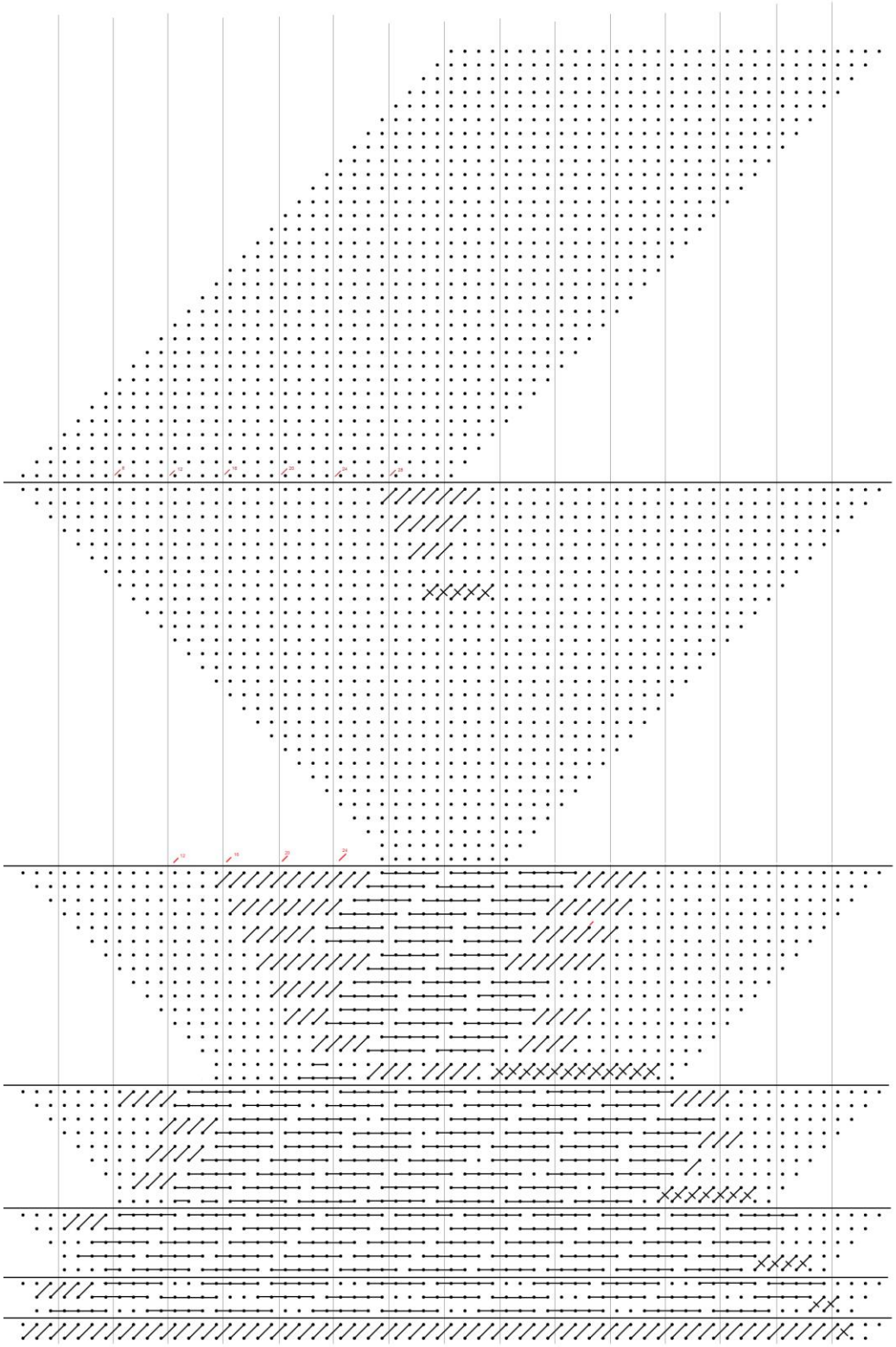


86 x 9 CLAs
139 FAs
23 HAs

Total: 4955

46-bit Carry Propagating Adder

Dadda CLA 32-Bit by 32-Bit Reduction



62-bit Carry Propagating Adder

177 x 9 bit CLAs
191 FAs
31 HAs
Total Gate Count: 9277

Bibliography

[1] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, vol. 13, pp. 14-17, 1964

[2] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, vol. 34, pp. 349-356, 1965.

[3] Wesley Chu, Ali Unwala, Pohan Wu, and Earl E. Swartzlander, Jr., **"**Implementation of a High Speed Multiplier Using Carry Lookahead Adders", Proc. Asilomar Conf. on Signals, Systems, and Computers, Nov. 3-6, 2013, Pacific Grove, CA.

[4] Earl E. Swartzlander, Jr., "High-Speed Computer Arithmetic," Chapter 22 in 2ed. *Computer Science Handbook,* Boca Raton, FL: Chapman & Hall/CRC, 2004.